REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA

Faculté de **T**echnologie

Département du **G**énie **E**lectrique

Mémoire de fin d'études de Master

Filière : *Télécommunications.*

Spécialité : *Systèmes de Télécommunications.*

Thème :

Implémentation efficace d'un réseau de neurones convolutif (CNN) sur circuit FPGA.

Présentés par :

1. MOUSSA El Habib.

2. CHAHBOUB Mohammed Ridha.

Proposés et Dirigés par :

Dr. DALI Mohammed.

Promotion 2022

Dédicace

Nous dédions cet humble travail

A nos chers parents qui ont donné leur vie,

Pour notre succès et atteindre les plus hauts niveaux

Et toute notre famille,

Sans oublier tous les enseignants qui ont contribué à notre formation

Depuis le primaire à ce jour,

A nos amis et frères,

Et aussi à tous ceux qui nous ont aidés à compléter ce mémoire

Enfin, à tous ceux que nous aimons.

Remerciement

Tout d'abord, Louange à Allah, Seigneur de l'univers qui nous a armés de patience et de force tout au long de ces longues années et nous a bénis avec succès dans l'accomplissement de cet œuvre.

Nous offrons également nos sincères remerciements et gratitude à notre encadrant Dr. DALI Mohammed, pour son aide tout au long de la période de travail, et pour tous les conseils, instructions et documents précieux qui ont grandement contribué à la mise en œuvre de ce travail.

Nous remercions également tous les enseignants de l'Université Yahia-Farés qui ont travaillé sur notre formation dès le premier jour, en particulier les enseignants du département de génie électrique. Nous les remercions tous, sans exception.

Et nous n'oublions pas de remercier tous ceux qui ont contribué, directement ou indirectement, à la réalisation de ce travail.

Nous exprimons également notre gratitude absolue à nos familles et parents pour tout ce qu'ils nous ont donné comme soutien financier et psychologique tout au long des années scolaires, et aussi à tous les amis dont les noms n'ont pas été mentionnés.

Merci à tous.

ملخص

في مجال الذكاء الاصطناعي والتعلم العميق ، تعد الشبكات العصبونية التلافيفية **CNN** نماذج قوية تستخدم في الرؤية الألية مثل التعرف التلقائي على الأرقام والحروف والنصوص المكتوبة بخط اليد والأشكال والأشخاص. تتطلب هذه النماذج معالجات قوية وحجم ذاكرة كبيرة لتنفيذها. يتمثل أحد الحلول الفعالة في استخدام دوائر قابلة لإعادة التكوين من نوع **FPGA** لإجراء تنفيذ فعال لشبكات **CNN** وهو هدف عملنا.

لقد قمنا بتصميم الدوائر الحسابية للطبقات المختلفة ضمن بيئة **System Generator** والتنفيذ في بيئة **Vivado**. الشبكة المصممة مخصصة لتطبيق التعرف التلقائي على الأرقام المكتوبة بخط اليد.

الكلمات المفتاحية : التعلم العميق ، **CNN** ، دائرة **FPGA** ، التعرف على الأرقام المكتوبة بخط اليد، **System Generator** ، **Vivado** ، المحاكاة المشتركة **.**

Résumé

Dans le domaine de l'Intelligence Artificielle et de l'Apprentissage Profond, les Réseaux de Neurone Convolutif (CNN) sont des modèles puissants utilisés dans la vision par ordinateur comme la reconnaissance automatique de chiffres, de lettres, de textes manuscrits, la reconnaissance de formes et de visages. Ces modèles nécessitent des processeurs puissants et un espace mémoire important pour leur mise en œuvre. L'une des solutions efficaces consiste à utiliser des circuits reconfigurables de type FPGA pour faire une implémentation efficace des réseaux CNN ce qui est l'objectif de notre travail.

Nous avons fait la conception des circuits de calcul des différentes couches sous l'environnement System Générateur et l'implémentation sous l'environnement Vivado. Le réseau conçu est destiné à l'application de reconnaissance automatique de digit manuscrit.

Mots clés : Apprentissage Profond, CNN, FPGA, reconnaissance digits manuscrits, System Générateur, Vivado, Co-simulation.

Abstract

In the field of Artificial Intelligence and Deep Learning, Convolutional Neural Networks (CNN) are powerful models used in computer vision such as automatic handwritten digits, latters and text recognition, face recognition and pattern recognition. These models require powerful processors and large memory space for their implementation. An efficient solution consists in using a Field-programmable Gate Array (FPGA) circuit to implement the CNN network, which is the objective of our work.

We carried out the design of the calculation circuits for the different layers under the System Generator environment and the implementation under the Vivado environment. The designed network used for the application of automatic recognition of handwritten digits.

Keywords: Deep Learning, CNN, FPGA circuits, handwritten digits recognition, System Generator, Vivado, Co-simulation.

Table des matières

Dédicace I
Remerciement II
ملخص III
Liste des figures VIII
Liste des tableaux X
Liste des abréviations XI
Introduction générale 1
Chapitre I : Deep Learning et réseaux CNN.
I.1. Introduction 4
I.2. L'apprentissage en profondeur (Deep Learning) 5
I.2.1. Définitions et concept de l'apprentissage en profondeur 5
I.2.2. Fonctionnement du Deep Learning 6
I.2.3. Les algorithmes du Deep Learning 7
I.2.3.1. L'algorithme de rétro-propagation du gradient : (back-propagation). 7
I.2.3.2. Fonction Coût (Cost Function) 7
I.2.3.3. L'algorithme de descente de gradient 8
I.3. Les réseaux neuronaux à convolution (CNN) 9
I.3.1. Introduction 9
I.3.2. Architecture des réseaux CNN 9
I.3.2.1. Couches de convolution 10
I.3.2.2. Les fonctions d'activations 12
I.3.2.3. Couche sous-échantillonnage (Pooling) 14
I.3.2.4. Couches complètement connectées (Fully Connected FC) 15
I.3.2.5. La fonction softmax 15
I.3.3. Apprentissage et inférence de réseau CNN 16
I.3.4. Exemples d'architectures de réseaux CNN 17
I.3.4.1. AlexNet 17
I.3.4.2. VGG 17
I.3.4.3. GoogleNet 18
I.3.4.4. ResNet 18
I.4. Conclusion 19

Chapitre II : Les circuits logiques programmables FPGA.

II.1. Introduction 21
II.2. Évolution de la Logique Programmable 21
II.3. Architecteur de circuit FPGA 22
II.3.1. Structure de base d'un circuit FPGA 22
II.3.2. Les composants d'un circuit FPGA 23
II.3.2.1. Blocs logiques configurables (CLB) 23
II.3.2.2. Les blocs d'E/S (IOB) 24
II.3.2.3. Blocs d'interconnexion 25
II.3.2.4. Les blocs RAM (BRAM) 26
II.3.2.5. Les blocs DSP 26
II.3.2.6. Les processeurs embarqués 27
II.3.2.7. Gestionnaire d'horloge numérique (DCM) 27
II.4. Les types de configuration des circuits FPGA 27
II.4.1. Les FPGA anti-fusion 27
II.4.2. Les FPGA basés sur SRAM 28
II.4.3. Les FPGA à base de Flash 28
II.5. La famille FPGA Virtex-7 28
II.6. Environnement de travail 31
II.6.1. Vivado 31
II.6.2. System Générateur (XSG) 32
II.7. Réseaux CNN sur FPGA 32
II.7.1. Exemples de travaux en relation 32
II.7.2. Les techniques d'optimisation de CNN sur FPGA 34
II.7.2.1. Optimisations algorithmiques 34
II.7.2.2. Technique d'optimisation de boucle 35
II.7.2.3. Optimisation du modèle CNN 36
II.8. Conclusion 36

Chapitre III : Implémentations, optimisation et résultats.

III.1. Introduction 38
III.2. Le réseau CNN proposé 38
III.3. Conception des circuits de calcul 41
III.3.1. Circuit de calcul de la convolution 1 41
III.3.2. Circuit de calcul ReLU 1 et Max pooling 1 42
III.3.3. Circuit de calcul de la convolution 2 43

III.3.4. Les circuits de calcul de ReLU 2 et de max pooling 2 44
III.3.5. Circuit de calcul de la convolution 3 44
III.3.6. Le circuit de calcul de ReLU 3 44
III.3.7. Le circuit de calcul de la couche complètement connectées 44
III.4. Optimisation des circuits de calcul 46
III.4.1. Technique de réutilisation de ressources 46
III.4.1.1. Circuit de calcul série de la convolution 1 46
III.4.1.2. Circuit de calcul série de ReLU 1 et de Max pooling 1 47
III.4.1.3. Circuit de calcul série de la convolution 2 48
III.4.1.4. Circuit de calcul série de ReLU 2 et Max pooling 2 48
III.4.1.5. Circuit de calcul série de la convolution 3 48
III.4.1.6. Circuit de calcul série de ReLU 3 49
III.4.1.7. Circuit de calcul série de couche complètement connectées 49
III.4.2. Représentation des données en virgule fixe 49
III.5. Résultats d'implémentation sur circuit FPGA 50
III.6. Vérification des résultats par Co-simulation sur la carte FPGA 54
III.7. Conclusion 55
Conclusion générale 56
Bibliographie 58

Liste des figures

Figure I.1 : Relation entre intelligence artificielle, Machine Learning, Deep Learning et CNN 4
Figure I.2 : Architecture d'un réseau de neurone (NN). 5
Figure I.3 : Structure d'un perceptron linéaire. 6
Figure I.4 : Illustration du principe de rétro-propagation 7
Figure I.5 : Optimisation de la fonction coût par descente de gradient. 8
Figure I.6 : Architecture générale de réseau CNN. 9
Figure I.7 : Les boucles de la couche convolution. 10
Figure I.8 : Exemple de calcul de la convolution 2D. 11
Figure I.9 : Les graphes des fonctions d'activation. 13
Figure I.10 : L'effet et les types de Couche pooling. 14
Figure I.11: Illustration des couches entièrement connectées. 15
Figure I.12 : Les deux phases d'implémentation d'un réseau CNN. 16
Figure I.13 : Architecture d'AlexNet. 17
Figure I.14 : Architecture de VGG 18
Figure I.15 : Structure de base d'un bloc du réseau GoogleNet. 18
Figure I.16 : Principe de l'architecture du réseau ResNet 19

Figure II.1 : Schéma simplifié de l'architecture de base d'un FPGA classique. 22
Figure II.2 : Composants d'un FPGA. 23
Figure II.3 : Configuration de base d'une cellule CLB d'un FPGA. 24
Figure II.4 : Structure d'un IOB de Xilinx 24
Figure II.5 : Interconnexion interne d'un circuit FPGA. 25
Figure II.6 : Architecture simplifiée d'un bloc DSP48E. 26
Figure II.7 : Structure générale d'une matrice de routage pour un FPGA à base de SRAM. . 28
Figure II.8 : Architecture interne de la famille Virtex 7 de Xilinx. 29
Figure II.9 : Arrangement des slice dans un CLB pour la série 7 de Xilinx. 29
Figure II.10 : L'environnement de visualisation de l'IDE de Vivado. 31
Figure II.11 : Les différentes techniques d'optimisation. 34
Figure II.12 : Les différentes configurations de déroulement de boucle. 35

Figure III.1 : La structure du réseau proposé. 39
Figure III.2 : Fenêtre de progression de l'apprentissage. 40
Figure III.3 : Schéma bloc du circuit de calcul de la convolution 1. 41
Figure III.4 : Structure des huit branches du circuit de calcul du ReLu 1 et Max pooling 1.. 42
Figure III.5 : Schéma bloc du circuit de calcul de la convolution 2. 43
Figure III.6 : La structure détaillée d'un seul groupe du circuit de calcul de la convolution 43
Figure III.7 : La structure du circuit de calcul de la couche complètement connectées. 45
Figure III.8 : Schéma détaillé de l'une des dix branches de calcul des scores de classification de la partie entièrement connectée. 45
Figure III.9 : Schéma détaillé du circuit de calcul de la classe finale et de son score. 45
Figure III.10 : Schéma bloc du circuit de calcul série de la convolution 1. 47
Figure III.11 : Schéma détaillé du circuit de calcul de la convolution série..... 47
Figure III.12 : Structure du circuit de calcul série de ReLU 1 et Max pooling 1. 47
Figure III.13 : Formats de représentation en virgule fixe et en virgule flottante 32 bits. 49
Figure III.14 : Format de représentation en virgule fixe choisi pour notre architecture..... 50
Figure III.15 : Fenêtre System Generator de génération du HDL de l'un des circuits développé. 50
Figure III.16 : Le schéma du circuit de convolution 1 sur le circuit Virtex 7 XC7VX485T-2FFG1761. 51
Figure III.17 : Estimé de la puissance consommée par le circuit de la convolution 1 l'implémenté sur Virtex 7 XC7VX485T-2FFG1761. 51
Figure III.18 : La consommation des ressources du réseau implémenté sur Virtex 7 XC7VX485T-2FFG1761. 53
Figure III.19 : Génération du modèle de gestion de la Co-Simulation. 54
Figure III.20 : Schéma du modèle de gestion de la Co-simulation du circuit de convolution 1 54
Figure III.21 : Image de la carte VC707 en exécution de la co-simulation. 55

Liste des tableaux

Tableau II.1 : Nombre de ressources disponibles dans chaque type de slice de la série 7 de Xilinx.....30
Tableau II.2 : Ressources disponibles pour les différentes versions de Virtex-7. 30

Tableau III.1 : Nombre total des éléments utilisés par chacun des circuits de calcul des trois couches de convolution. 44
Tableau III.2 : Ressources du circuit Virtex 7 XC7VX485T-2FFG1761 consommées par la convolution 2 en virgule flottante. 46
Tableau III.3 : Nombre total des éléments utilisés par chacun des circuits de calcul série pour les trois couches de convolution. 49
Tableau III.4 : Les ressources et la puissance dynamique consommées par les différents circuits de calcul implémenté sur Virtex 7 XC7VX485T-2FFG1761. 52

Liste des abréviations

AI	Artificielle Inteligence
ASIC	Application Specific Integrated Circuit
BRAM	Block Random Access Memory
CNN	Convolutional Neural Network
CPU	Central Processing Unit.
CONV	Convolutional Layer
CMOS	Complementary metal oxide Semiconductor
CLB	Configurable Logic Block
DL	Deep Learning
DSP	Digital Signal Processing
DCM	Digital Clock Manager
EPROM	Erasable programmable read only memory
FPGA	Field Programmable Gate Array
FC	Fully-connect Layer
FF	Flip flop
FIFO	First in first out
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IOB	Input Output Block
JTAG	Joint Test Action Group
LUT	Look-up Table
MAC	Multiply and accumulate

ML	Machine learning.
MNIST	Modified National Institute of Standards and Technology
MUX	Multiplexer
NN	Neural Network
PLD	Programmable logic device
RAM	Random Access Memory
ROM	Read Only Mémory
ReLU	Unités Rectifié linéaires
RTL	Registre Transfert Level
SRAM	Static Random Access Memory
TTL	Transistor-transistor logic
VGG	Visual Geometry Group
VHDL	Very high speed Integrated Circuits Hardware Description Language
XSG	Xilinx System Generator

Introduction générale

Dans notre monde moderne, monde numérique où la technologie se développe à grande échelle, l'intelligence artificielle a fait un grand pas en se manifestant comme un atout majeur dans la société moderne. Le terme IA (Intelligence Artificielle) désigne toutes les techniques et méthodes qui permettent de doter des systèmes informatiques de capacités proches de celles de l'être humain.

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.

Le réseau de Neurone Convolutif (CNN) est une architecture d'apprentissage en profondeur bien connue inspirée du réseau de neurones artificiels. Il a été principalement utilisé dans diverses applications, notamment la classification d'images, la détection d'objets et la reconnaissance automatique de chiffres, de lettres, de textes manuscrits, de personnes. Contrairement aux algorithmes d'apprentissage automatique traditionnels, les algorithmes CNN sont extrêmement coûteux en calcul et en mémoire. Les algorithmes CNN nécessite généralement des millions de paramètres et des milliards d'opérations pour traiter une seule entrée d'image. C'est un grand défi pour implémenter des CNN sur un système embarqué en raison de contraintes matérielles sévères telles que les ressources de calcul, la bande passante mémoire et la mémoire sur puce.

Par conséquent, ces dernières années, le FPGA (Field Programmable Gate Array) est devenu une solution alternative attrayante pour les algorithmes basés sur CNN en raison de ses performances relativement élevées, de sa flexibilité, de son efficacité énergétique et de son cycle de développement rapide par rapport à CPU et GPU. Cependant, le FPGA est limité en ressources (mémoire limitée, bande passante d'E/S limitée et ressources de calcul limitée). Par conséquent, le défi consiste à trouver une correspondance entre le modèle de CNN et le modèle d'implémentation.

Notre projet de fin d'étude s'inscrit dans le cadre d'une implémentation efficace d'un réseau de neurone convolutif sur circuit FPGA. Pour faire ça, nous avons utilisé deux méthodes d'optimisation. Le réseau étudié est constitué de trois couches de convolution et une seule

couche complètement connectée. L'application visée par le réseau est la reconnaissance automatique des chiffres manuscrits.

Ce présent mémoire est organisé en 3 chapitres :

Le premier chapitre est consacré aux Deep Learning et les réseaux de neurone convolutif CNN. Dans ce contexte, nous présenterons les concepts de base et les notions importantes qui sont en relation avec l'apprentissage profond (définition, Architectures...etc.). Ensuite, nous présenterons en détailler les réseaux CNN car c'est la base de notre projet.

Le deuxième chapitre est consacré à la présentation des circuits logiques programmables FPGA, environnement matérielle et logicielle. Nous commençons par l'explication d'évolution de la Logique Programmable et l'architecture interne des circuits FPGAs avec la présentation des principaux composants constituant un circuit FPGA. A la fin du deuxième chapitre nous abordons le problème de l'implémentation des réseaux CNN sur les circuits FPGA.

Dans le troisième chapitre, nous présentons les différents schémas de conception, les différentes optimisations sur les schémas de conception, les résultats de simulation ainsi que les résultats d'implémentation sur FPGA. Nous finissons le troisième chapitre par la présentation de la vérification expérimentale du réseau CNN sur la carte FPGA VC707 en utilisant la technique de Co-simulation matérielle et logicielle.

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

Chapitre I :

Deep Learning et réseaux CNN.

I.1. Introduction

Au cours de ces dernières années, l'intelligence artificielle (IA) joue un rôle essentiel pour combler le fossé entre les capacités des humains et des machines. Elle est basée sur l'imitation de l'intelligence humaine pour réaliser une multitude de tâches telles que la reconnaissance d'images et de vidéos, la classification d'images, les applications biomédicales et les véhicules autonomes. [1]

L'apprentissage en profondeur (DL) est le sous-domaine de l'intelligence artificielle (IA) dérivé de l'apprentissage automatique (ML), axé sur la création de réseaux de neurones à plusieurs couches capables de prendre des décisions précises basées sur une grande quantité des données. [2]

Les réseaux de neurone convolutif (CNNs) est parmi les méthodes les plus utilisée dans l'apprentissage en profondeur. En effet, la technique CNNs offre une précision proche de l'homme ou même meilleur dans de nombreuses applications liées à la vision par ordinateur (par exemple, la classification, la détection, la segmentation et la reconnaissance vocale et reconnaissance des images). [3]

Dans ce chapitre, nous allons détailler les principes de base de l'apprentissage en profondeur en mettant l'accent sur les réseaux de neurone convolutif (CNNs).

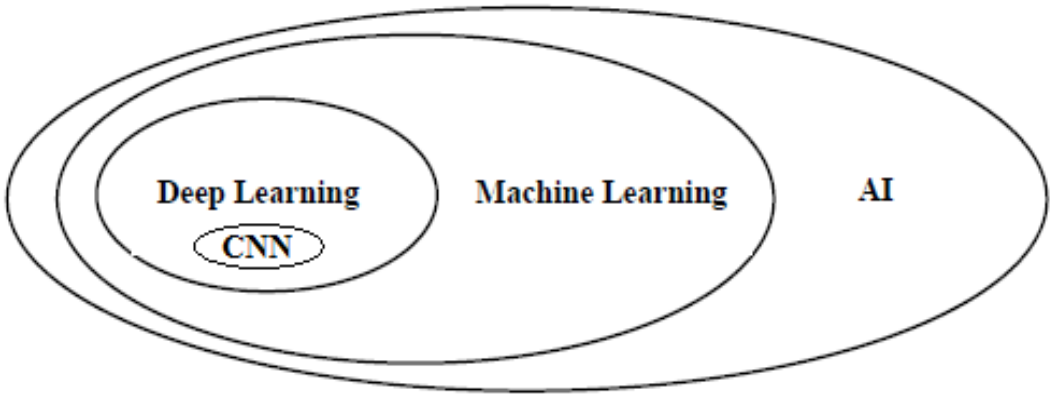


Figure I.1 : Relation entre intelligence artificielle, Machine Learning, Deep Learning et CNN.

I.2. L'apprentissage en profondeur (Deep Learning)

Depuis 2006, l'apprentissage structuré en profondeur, ou plus communément appelé apprentissage en profondeur ou apprentissage hiérarchique, est devenu un nouveau domaine de recherche sur l'apprentissage automatique. Les techniques développées à partir de la recherche sur l'apprentissage en profondeur ont déjà eu un impact sur un large éventail d'applications dans différents domaines de la vie quotidienne comme la médecine, la robotique, l'informatique, la technologie spatiale et militaire. [4]

I.2.1. Définitions et concept de l'apprentissage en profondeur

L'apprentissage en profondeur est un sous-domaine de l'apprentissage automatique qui traite des algorithmes qui ressemblent étroitement à une version trop simplifiée du cerveau humain pour résoudre des problèmes liés au traitement du signal, image et vidéo comme la classification, la reconnaissance et la détection.. C'est un ensemble d'algorithmes d'apprentissage automatique qui tentent d'apprendre à plusieurs niveaux. De nombreux exemples courants peuvent être trouvés dans des applications utilisées par les smartphones, les tablettes, les ordinateurs et de nombreux autres systèmes et équipements électroniques.[5]

Les réseaux de neurone sont les éléments constitutifs de l'apprentissage en profondeur. L'unité de base d'un réseau de neurone est le neurone appelé perceptron linéaire. Un réseau de neurone est composé de nombreux perceptron, fortement interconnectés par des liens d'intensité variable. Les réseaux de neurones sont généralement organisés en couches. Les couches de réseau de neurone sont constituées d'un certain nombre de nœuds interconnectés (qui représentent les neurones). Comme montré la Figure I.2.

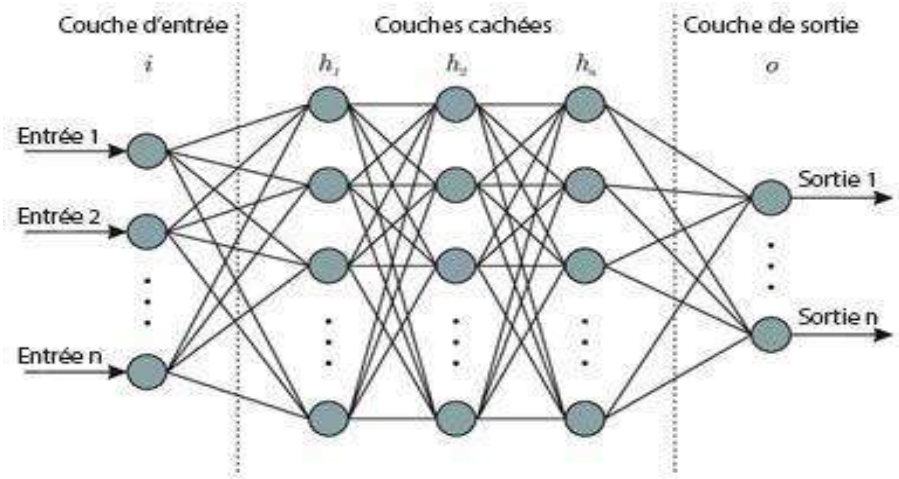


Figure I.2 : Architecture d'un réseau de neurone (NN).

Les nœuds interconnectés c'est les perceptrons et dont le but est de combiner linéairement les sorties des neurones de la couche précédente avec lesquels ils partagent une connexion à travers un ensemble de connexions pondérées. Ces connexions pondérées sont initialisées à une valeur généralement aléatoire, et seront ensuite ajustées par une phase d'apprentissage ou formation. Un biais b (agissant comme un seuil) est également appris pour chaque neurone et ajouté au résultat de la combinaison linéaire entre les poids $\mathcal{W}_n$ et les entrées x_n. La sortie est ensuite passée par une fonction d'activation f afin d'introduire des non-linéarités. La structure de perceptron linéaire dans la Figure I.3. [6]

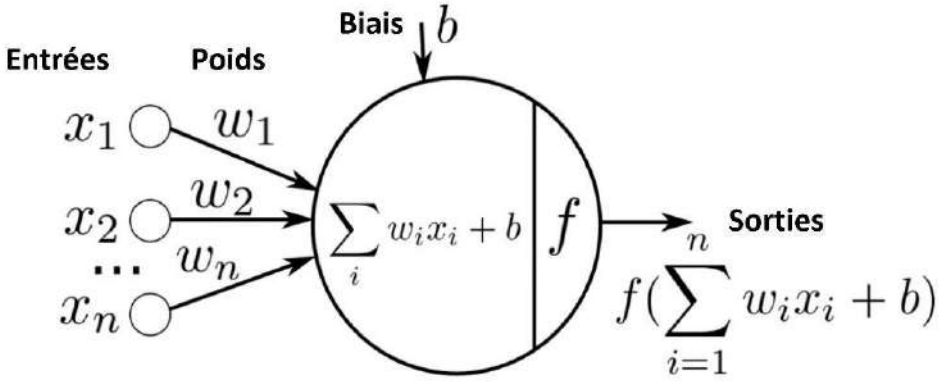


Figure I.3 : Structure d'un perceptron linéaire.

I.2.2. Fonctionnement du Deep Learning

Les signaux voyagent entre les neurones dans un cerveau artificiel de la même manière qu'ils le font dans un cerveau humain. Le secret de cette prouesse repose en grande partie sur les algorithmes.

Dans le cas de la reconnaissance visuelle, les systèmes d'apprentissage en profondeur doivent être capables de reconnaître toutes les formes existantes sous tous les angles pour être efficaces. Par conséquent, il sera capable de détecter les voitures sur les routes au milieu du paysage. Cela n'est concevable que si la machine est largement entraînée. Cela implique de regarder des milliers de photos de voitures de toutes formes et angles possibles.

Lorsque de nouvelles images apparaissent, elles sont envoyées au réseau de neurones chargé de les analyser et de déterminer si l'objet au milieu du plan est bien une voiture. La machine a-t-elle gagné ? Elle garde la bonne réponse au chaud car cela l'aidera à faire face à d'autres situations similaires le jour où elle devra identifier une autre voiture. [7]

I.2.3. Les algorithmes du Deep Learning

I.2.3.1. L'algorithme de rétro-propagation du gradient : (back-propagation).

Le bloc de construction le plus fondamental de l'apprentissage en profondeur est la rétro-propagation, abréviation de propagation en arrière des erreurs, un algorithme utilisé pour entraîner les réseaux de neurones en apprentissage supervisé. Bien que la rétro-propagation ait été inventée dans les années 1970, elle a été popularisée plusieurs années plus tard, en 1989, par Rumelhart, Hinton et Williams dans leur article "Learning representations by back-propagating errors". Cet algorithme permet de calculer l'erreur pour chaque neurone, de la dernière couche vers la première, et ce pour tous les paramètres du réseau. [5, 8]

L'algorithme de rétro-propagation du gradient se déroule en deux phases :

- calculer l'erreur δ^{sortie} en comparant la sortie avec le résultat attendu,
- propager l'erreur de couche en couche vers l'arrière.

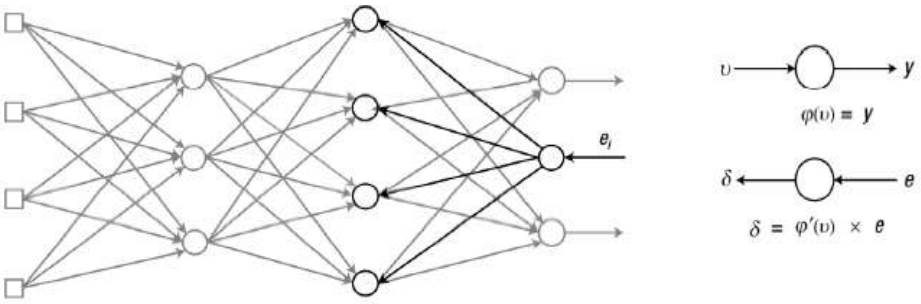


Figure I.4 : Illustration du principe de rétro-propagation.

I.2.3.2. Fonction Coût (Cost Function)

Nous avons introduit l'erreur entre le modèle et les données. L'ensemble de ces erreurs est représenté par une fonction coût (En anglais Cost ou loss function). La fonction coût est exprimée par la technique de Back propagation dans le réseau de neurones. Cette fonction peut être définie avec plusieurs fonctions, mais dans le cadre de l'apprentissage profond, les deux fonctions les plus utilisées sont : [9, 10]

- Le maximum de vraisemblance : Le maximum de vraisemblance cherche à trouver les valeurs optimales pour les paramètres en maximisant une fonction de vraisemblance dérivée des données d'apprentissage.
- L'entropie croisée (*Cross-Entropy*) : Dans le cas des problèmes de classifications utilisant les réseaux de Neurone Convolutif, c'est la fonction d'entropie croisée qui est favorisée.

I.2.3.3. L'algorithme de descente de gradient

L'algorithme de descente du gradient utilise le résultat de l'algorithme de rétro-propagation pour mettre à jour les paramètres. Il cherche à minimiser la fonction de coût $E(x)$ en modifiant x. Il est itératif et procède par améliorations successives. Son but est de trouver ou de s'approcher d'un point stationnaire ou minimum global, c'est à dire d'un point où le gradient de la fonction de coût est nul et ainsi aboutir à une configuration stable du réseau neuronal. Le principe de cet algorithme remonte à 1847 avec Cauchy.[8]

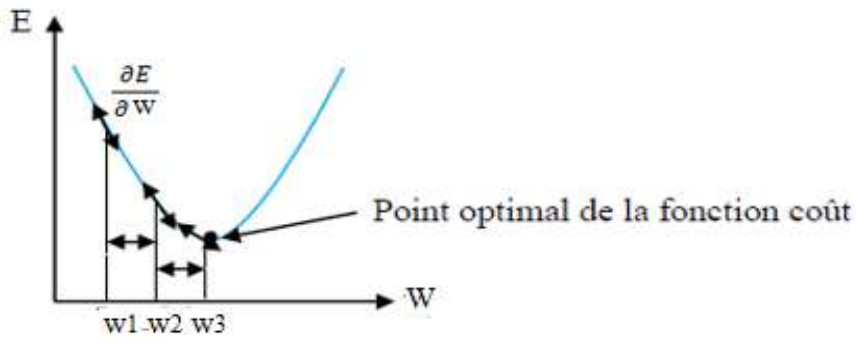


Figure I.5 : Optimisation de la fonction coût par descente de gradient.

Comme il est illustré sur la Figure I.5, l'algorithme utilise les dérivées partielles de l'erreur par rapport aux poids pour avancer vers le point optimal suivant un pas déterminé. [9]

La formule de mise à jour des paramètres est :

$$w_t = w_{t-1} - \Delta w_t \,, \quad \Delta w_t = \eta * \frac{\partial E}{\partial w} \tag{I.1}$$

Le paramètre W se met à jour de manière itérative à l'aide d'un taux d'apprentissage η strictement positif qui permet de régler la vitesse de convergence.

La descente de gradient peut s'effectuer de plusieurs manières :

- De manière globale (*batch gradient*) : on envoie au réseau toutes les données en une seule fois, puis le gradient est calculé et les poids ajustés.
- Par des lots (*mini-batch gradient*) : on envoie au réseau les données par petits groupes d'une taille définie par l'expérimentateur dont on calcule les erreurs, puis l'erreur moyenne est calculée et enfin les poids sont mis à jour.
- De façon unitaire, stochastique (*stochastic gradient*) : cette méthode envoie une seule donnée à la fois dans le réseau et donc les poids sont mis à jour à chaque fois juste après.

I.3. Les réseaux neuronaux à convolution (CNN)

I.3.1. Introduction

Les réseaux de Neurone Convolutif CNN est une technique du Deep Learning qui a été proposée pour la première fois par LeCun et autres. Contrairement aux réseaux conventionnels entièrement connectés (FC), les poids partagés et les connexions locales dans le CNN sont utilisés pour tirer pleinement parti des structures de données d'entrée 2D telles que les signaux d'image.

L'émergence des CNN est inspirée du traitement biologique car le schéma de connexion entre les neurones est similaire au tissu du cortex visuel des animaux. [11]

Les réseaux CNN sont conçu pour apprendre automatiquement et de manière adaptative les hiérarchies spatiales des caractéristiques des modèles de bas à haut niveau. Ils sont largement utilisés pour les tâches de classification d'images, de reconnaissance d'objets et d'étiquetage de scènes.[12]

I.3.2. Architecture des réseaux CNN

Comme tous les réseaux de neurones, les CNN sont composées des couches multiples, la première étant la couche d'entrée suivie d'un certain nombre des couches et puis par une couche de sortie qui donne le résultat de classification.

La structure typique est composée d'une succession de couches de convolutions de fonctions d'activations et de couches de sous échantillonnage ou pooling, suivie d'une couche entièrement connectée. La partie contenant les couches de convolution, d'activation et de pooling effectue l'extraction des caractéristiques, tandis que la partie contenant les couches entièrement connectées (fully connected) et la fonction softmax réalise la classification [13]**.** Le modèle commun de l'architecture CNN est illustré à la Figure I.6.

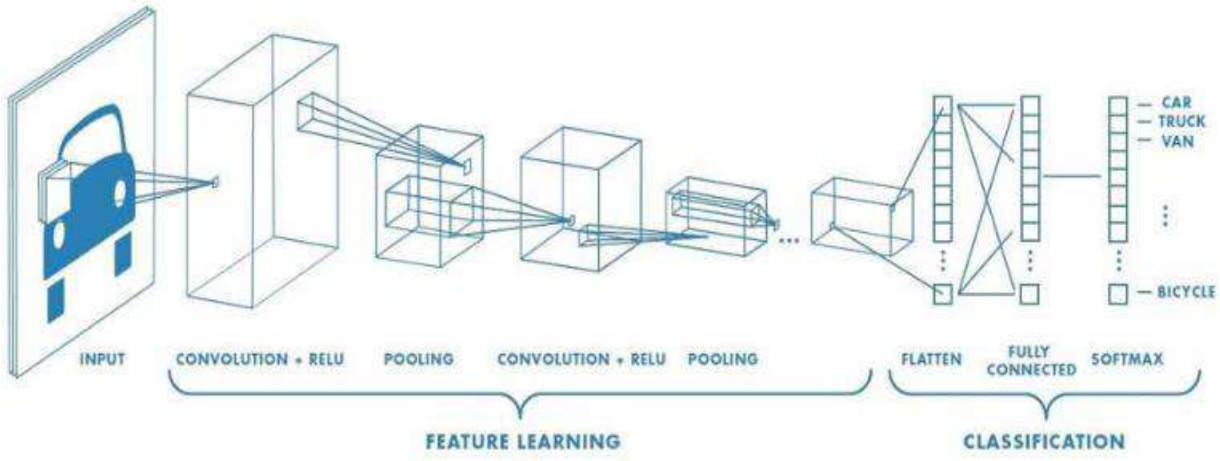


Figure I.6 : Architecture générale de réseau CNN.

I.3.2.1. Couches de convolution

Dans l'architecture CNN, le composant le plus important est la couche de convolution**.** Il se compose d'une collection de filtres de convolution (appeler noyau ou Kernel). Le noyau ou Kernel c'est une grille de valeurs ou de nombres discrets dont chaque valeur est appelée poids du noyau. Des nombres aléatoires sont assignés pour agir comme poids du noyau au début du processus de formation CNN. Il existe plusieurs méthodes différentes utilisées pour initialiser les poids. Ensuite, ces poids sont ajustés à chaque période d'entraînement ; ainsi, le noyau apprend à extraire des caractéristiques importantes.

Durant la phase de convolution, l'entrée, exprimée sous forme d'un ensemble de matrices, est convolée avec ces filtres pour générer la carte de caractéristiques de sortie (Output features maps ou Output FM). [14]

Comme le montre la Figure I.7, 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 « input feature maps (input FM)», un ensemble de groupe de filtre (poids conv) et une sortie sous forme de plusieurs matrices aussi de même dimension appelée « Output feature maps (Output FM)».

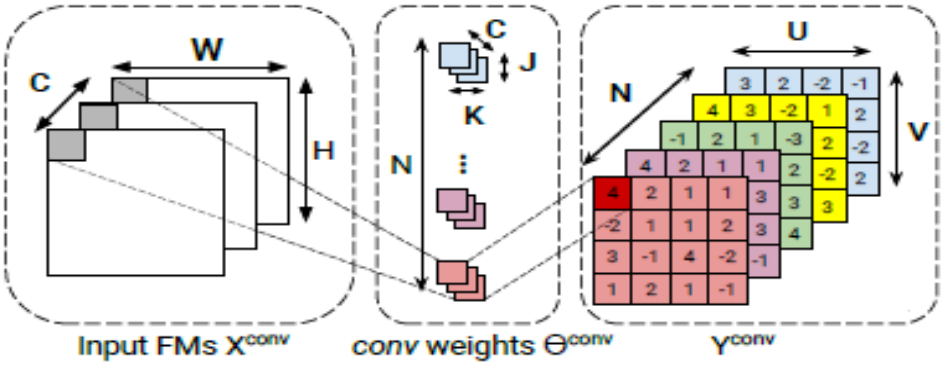


Figure I.7 : Les boucles de la couche de convolution. [3]

Tel que :

- W et H représentent les dimensions (nombre de colonnes et nombre de lignes) de chacune des matrices d'entrée et C est le nombre des matrices d'entrée (nombre de feature maps).
- K et J sont les dimensions d'un kernel, C est le nombre de kernel par groupe et N est le nombre total de groupes.
- U et V représentent les dimensions (nombre de colonnes et nombre de lignes) de chacune des matrices de sortie et N est le nombre de matrice de sorties.

La formule de calcul la convolution 3D est obtenue en sommant des convolutions 2D sur toutes les valeurs de la variable c. Pour calculer la valeur d'un pixel de position (*v*,*u*) d'une sortie de numéro *n* on doit effectuer le calcul suivant :

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

$$Y[b,n,v,u] = \beta[n] + \sum_{c=1}^{C} \sum_{j=1}^{J} \sum_{k=1}^{K} X[b,c,v.S+j,u.S+k].\theta[n,c,j,k] \quad \text{(I.2)}$$

Où : X : les maps de caractéristique d'entrée. Y : les maps de caractéristique de sortie.

β : Les biais appris. θ : Les filtres appris. B : Nombre de lot. S : le pas (stride).

Pour calculer le résultat de convolution pour une seule map de sortie on doit refaire le calcul de l'équation (I.2) pour toutes les valeurs de v,u (1 à V et 1 à U respectivement). Pour obtenir le résultat de toute une couche de convolution on doit refaire le calcul d'une sortie sur la totalité des sorties et pour la totalité des frames du lot (n de 1 à N et b de 1 à B). Et Pour bien comprendre la convolution 2D, nous avons présenté dans la figure I.8 un exemple pour Strid =1, K=2 et J=2 :

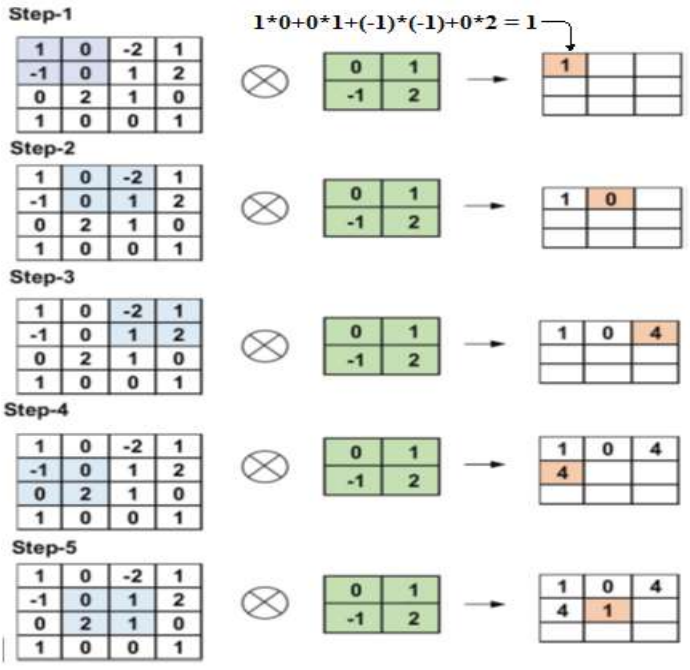


Figure I.8 : Exemple de calcul de la convolution 2D.

La taille des cartes de caractéristiques de sortie peut être formulée comme suit :

$$\begin{cases} V = (H - J) * S + 1 \\ U = (W - k) * S + 1 \end{cases} \quad \text{(I.3)}$$

La couche de convolution est caractérisée par trois hyper-paramètres : [15]

- La profondeur de la couche : C'est le nombre de noyaux de convolution utilisés pour le calcul de la même carte de caractéristique de sortie.

- Le pas de convolution (stride) : c'est le pas ou la distance à laquelle le filtre se déplace d'une position à la suivante. Par exemple dans le cas ou *S=1* 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. Le stride contrôle le chevauchement au niveau des cartes de caractéristiques de sortie.
- Le remplissage à zéro (Zéro padding) : Consiste à rajouter des zéros à la frontière du volume d'entrée. Ce remplissage permet le contrôle de la dimension spatiale du volume de sortie. La quantité de remplissage dépend de la taille du noyau.

I.3.2.2. Les fonctions d'activations

La sortie de chaque couche de convolution est envoyée à une couche de fonction d'activation. La couche de fonction d'activation se compose d'une fonction d'activation qui prend la carte de caractéristiques produite par la couche de convolution et génère la carte d'activation comme sortie. La fonction d'activation est utilisée pour transformer le niveau d'activation d'un neurone en un signal de sortie. Il spécifie la sortie d'un neurone vers une entrée donnée.

La couche d'activation accélère le processus d'apprentissage des cartes d'entités en introduisant une combinaison non linéaire d'entités. En général, 3 propriétés sont attendues de la fonction d'activation :

- ✓ Non linéarité : c'est la propriété cruciale de la fonction d'activation. Grace à cette propriété le réseau de neurones peut être utilisé pour résoudre des problèmes non linéaires.
- ✓ Différentiable : ce qui signifie que nous avons une dérivée continue du premier ordre. C'est une propriété souhaitable pour activer des méthodes d'optimisation basées sur un gradient comme la retro-propagation.
- ✓ Monotone : cette caractéristique aide le réseau de neurones à converger plus facilement vers un modelé plus précis.

Les types de fonctions d'activation les plus couramment utilisés dans les réseaux CNN et d'autres réseaux de neurone profonds sont : [16]

- **Sigmoïde :** L'entrée de cette fonction d'activation est constituée de nombres réels, tandis que la sortie est limitée entre zéro et un. La courbe de la fonction sigmoïde est en forme de S et peut être représentée mathématiquement par :

$$f(x)_{sigm} = \frac{1}{1+e^{-x}} \quad \text{(I.4)}$$

- **Tanh** : Elle est similaire à la fonction sigmoïde, car son entrée est constituée de nombres réels, mais la sortie est limitée entre − 1 et 1. Sa représentation mathématique est :

$$f(x)_{tanh} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \text{(I.5)}$$

- **ReLU :** C'est la fonction la plus couramment utilisée dans le contexte CNN. Elle convertit les valeurs entières de l'entrée en nombres positifs. L'utilisation de la fonction ReLU accélère le processus d'entrainement. Une charge de calcul plus faible est le principal avantage de ReLU par rapport aux autres : tous les éléments négatifs sont mis a 0, donc pas d'exponentiels, pas d'Operations de multiplication ou de division. Donc, cette fonction est facilement implémentée dans le matériel. [17].

  Le calcul du gradient est très simple (0 ou 1 selon le signe de x). De plus, le calcul d'un ReLU est facile. Sa représentation mathématique est :

$$f(x)_{ReLU} = \max(0, x) \quad \text{(I.6)}$$

- **ReLU6 :** Cette fonction d'activation a l'avantage d'être conçu pour les opérations en virgule fixe et les approches de quantification au lieu des opérations en virgule flottante. Cela signifie que le nombre de bits pour la partie entière peut être limité à 3 bits car la sortie est dans l'intervalle [0, 6]. Cette propriété permet à ReLU6 d'être utilisé dans des modèles qui visent des plateformes mobiles et embarquées comme MobileNet. En effet, les opérations en virgule fixe, en particulier sur FPGA, sont plus efficaces en termes d'utilisation du matériel et de consommation d'énergie. ReLU6 est donnée par l'expression suivante :

$$h(x) = \max(0, x, 6) \quad \text{(I.7)}$$

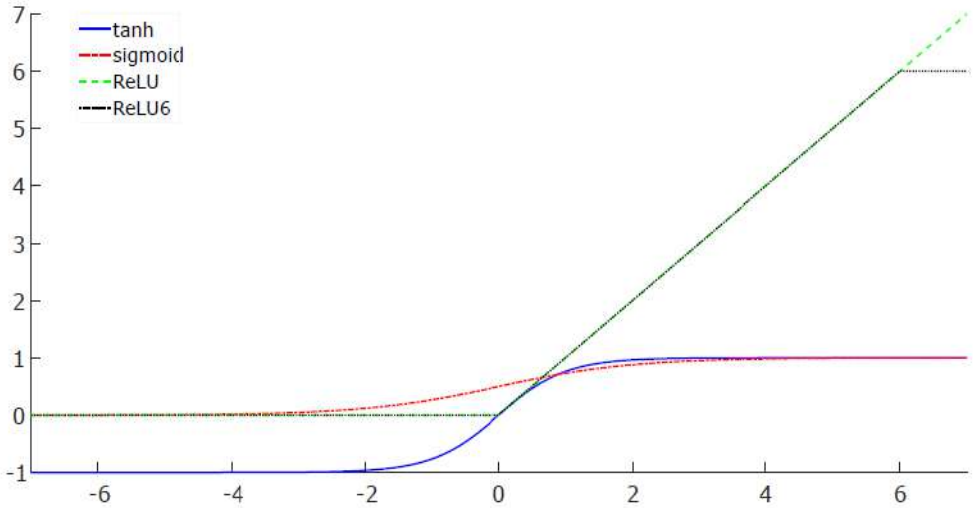


Figure I.9 : Les graphes des fonctions d'activation.

I.3.2.3. Couche sous-échantillonnage (Pooling)

Les couches de sous-échantillonnage ou pooling sont placées généralement entre les couches de convolution. Cette couche vise à réduire progressivement la dimensionnalité des cartes de caractéristiques, et donc à réduire encore le nombre de paramètres et la complexité de calcul du modèle, mais conserve les caractéristiques importantes. Cette opération remplace un carré de pixels (généralement de dimension 2×2 ou 3×3) par une seule valeur. [15]

La Figure I.10 explique les deux techniques de pooling les plus courantes qui sont :

- Max Pooling : il prend le pixel qui a la valeur maximale entre tous les pixels de la sélection. la formule mathématique du max Pooling est :

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

$$Y^{pool}[b,n,v,u] = \max_{p,q\in[1:k]} (X^{pool}[b,n,v+p,u+q]) \quad \text{(I.8)}$$

- Mean Pooling : il prend la moyenne de tous les pixels de la sélection. L'expression du Mean Pooling est :

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

$$Y^{pool}[b,n,v,u] = \operatorname{mean}_{p,q\in[1:k]} (X^{pool}[b,n,v+p,u+q]) \quad \text{(I.9)}$$

Où : p et q : c'est les dimensions de carré de pixels de pooling.

X^{pool} : c'est l'entrée de pooling. Y^{pool} : c'est la sortie de pooling.

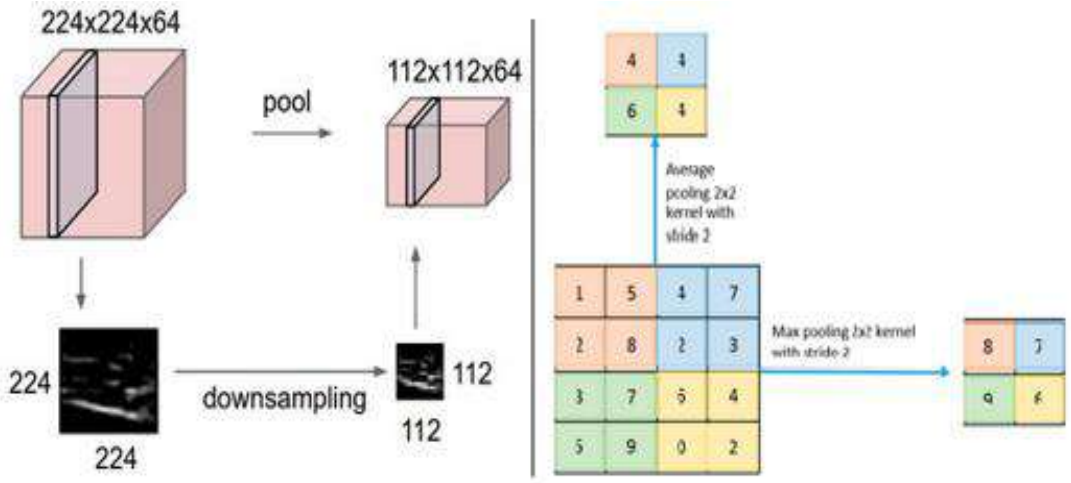


Figure I.10 : L'effet et les types de Couche pooling.

I.3.2.4. Couches complètement connectées (Fully Connected FC)

Il s'agit de la couche classique des NN entièrement connectés, dans laquelle tous les neurones d'une couche sont connectés à tous les neurones de la couche précédente. Il est généralement utilisé pour la classification de l'image à partir des caractéristiques résultant des couches de convolution et de sous-échantillonnage (pooling). Une ou plusieurs couches FC sont donc placées à la fin du réseau.

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 comme le montre la Figure I.11. 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.

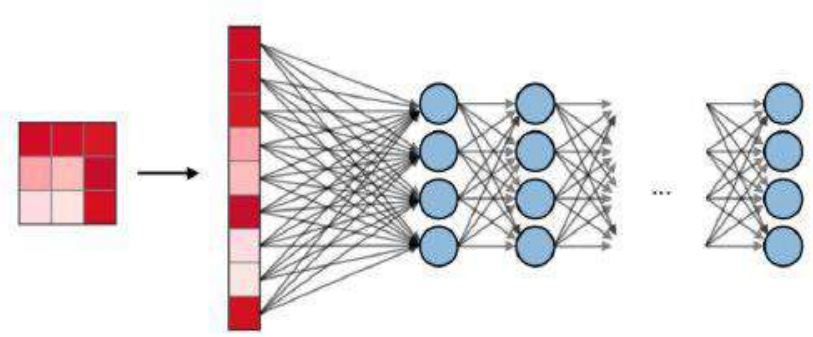


Figure I.11 : Illustration des couches entièrement connectées.

I.3.2.5. La fonction softmax

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 tâche 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 :

$$p_i = \frac{e^{a_i}}{\sum_{K=1}^{N} e^{a_{i_k}}} \tag{I.10}$$

Où : p_i est la sortie de la fonction softmax pour la classe *i*, a_i désigné la $i^{ième}$ sortie de la dernière couche FC correspondant à la classe *i* et *N* est le nombre de classes du réseau.

I.3.3. Apprentissage et inférence de réseau CNN

L'implémentation d'un réseau CNN est réalisée en deux phases. La première phase est l'apprentissage, où le modèle apprend à faire la classification en utilisant l'algorithme de rétro-propagation. La deuxième phase est l'inférence, où le modèle prédit la sortie de nouveaux échantillons de données, en utilisant le modèle appris. Nous pouvons voir les différents processus dans la Figure I.12. Habituellement, l'apprentissage est effectué une seule fois alors que l'inférence est exécutée chaque fois que le réseau CNN doit traiter une nouvelle entrée. Ainsi, la plupart des efforts doivent être concentrés sur l'accélération de la phase d'inférence. [16]

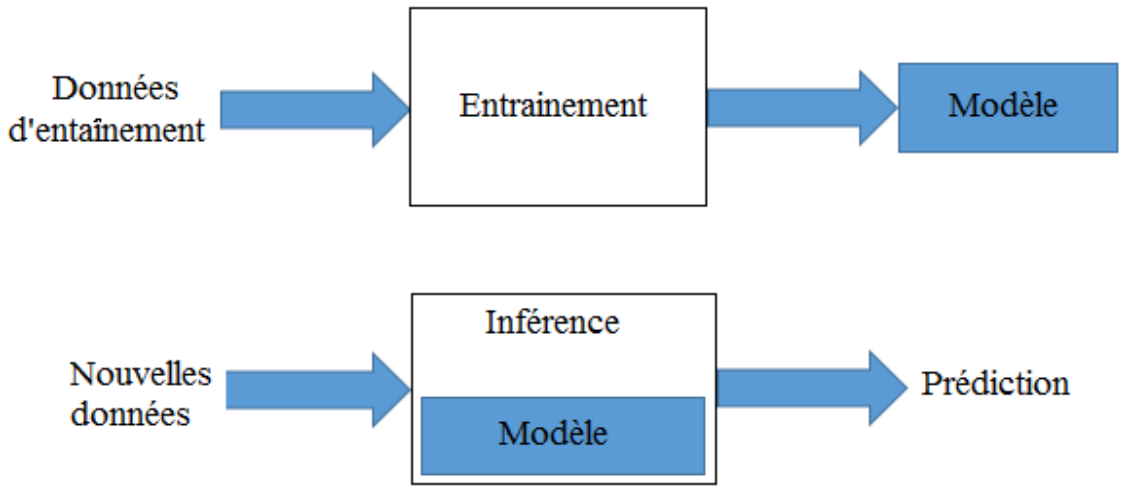


Figure I.12 : Les deux phases d'implémentation d'un réseau CNN.

Durant la phase d'apprentissage l'algorithme détermine les valeurs des filtres et des biais à utiliser pour les différentes couches de convolution et les couches entièrement connectées. Durant cette phase, l'algorithme utilise les paramètres suivants :

- **Epochs :** un epoch correspond à un seul passage en avant et en arrière de la totalité des données d'apprentissage à travers le réseau. Puisque on utilise un ensemble de données limité et le Gradient Descent qui est un processus itératif, la mise à jour des poids avec un seul passage ou une seule epoch ne suffit pas. Donc il faut plusieurs epochs pour arriver aux valeurs optimales des poids.
- **Batch :** pour les bases de données de grande taille, on ne pout pas transmettre l'intégralité de l'ensemble de données au réseau neuronal en une seule fois. Ainsi, on divise l'ensemble de données en nombre de lots ou parties appelé batch.
- **Iteration :** une itération correspond au passage d'un seul batch. Donc, le nombre total d'itération est égale au nombre de batch par epoch multiplier par le nombre total d'epochs.

I.3.4. Exemples d'architectures de réseaux CNN

Différentes améliorations de l'architecture CNN ont été apportées depuis 1989 à ce jour. Cette section détaille certains des réseaux CNN bien connus. Ce sont des exemples de la façon de construire un CNN efficace. [14, 18]

I.3.4.1. AlexNet

L'histoire des CNNs profonds a commencé avec l'apparition de LeNet. À cette époque, les CNN étaient limités aux tâches de reconnaissance de chiffres manuscrits, qui ne peuvent pas être mises à l'échelle pour toutes les classes d'images.

AlexNet est considérée comme la première architecture CNN profonde, qui a montré des résultats révolutionnaires pour les tâches de classification et de reconnaissance d'images. AlexNet a été proposé par Krizhevsky et al. Ce qui a amélioré la capacité d'apprentissage du CNN en l'approfondissant et en appliquant plusieurs stratégies d'optimisation des paramètres. La conception architecturale de base d'AlexNet est illustrée à la Figure I.13.

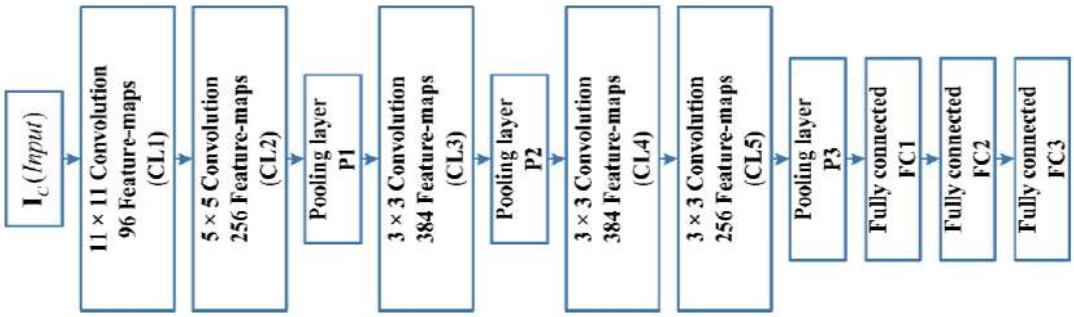


Figure I.13 : Architecture d'AlexNet.

I.3.4.2. VGG

Après que la technique CNN a été déterminée efficace dans le domaine de la reconnaissance d'images, un principe de conception simple et efficace pour CNN a été proposé par Simonyan et Zisserman. Cette conception innovante s'appelait Visual Geometry Group (VGG).

L'architecture VGG utilise des filtres 3 × 3 plutôt que les filtres 5 × 5 et 11 × 11. Cela a montré expérimentalement que l'utilisation de ces filtres de petite taille pouvait produire la même influence que les filtres de grande taille.

En général, VGG a obtenu des résultats significatifs pour les problèmes de localisation et de classification d'images. Cependant, le coût de calcul de VGG était excessif en raison de son utilisation d'environ 140 millions de paramètres, ce qui représentait son principal défaut.

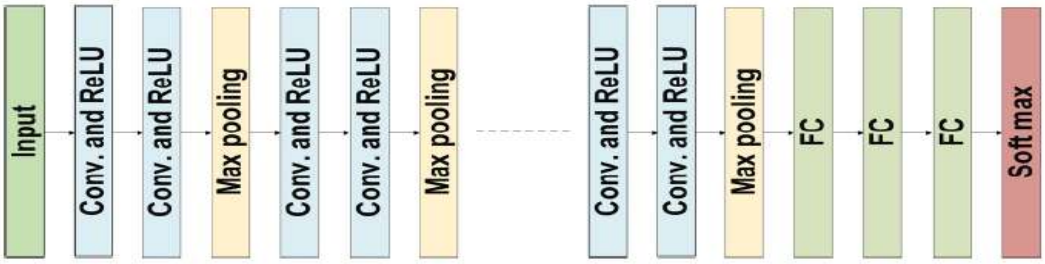


Figure I.14 : Architecture de VGG.

I.3.4.3. GoogleNet

GoogleNet a été le gagnant du concours 2014-ILSVRC et également connu sous le nom d'Inception-V1. L'objectif principal de l'architecture GoogleNet était d'atteindre une grande précision avec un coût de calcul réduit (Szegedy et al. 2015). Il a introduit le nouveau concept de bloc de création dans CNN, par lequel il incorpore des transformations convolutives à plusieurs échelles en utilisant des idées de division, de transformation et de fusion. La Figure I.15 illustre l'architecture du bloc de base de l'architecture de GoogLeNet.

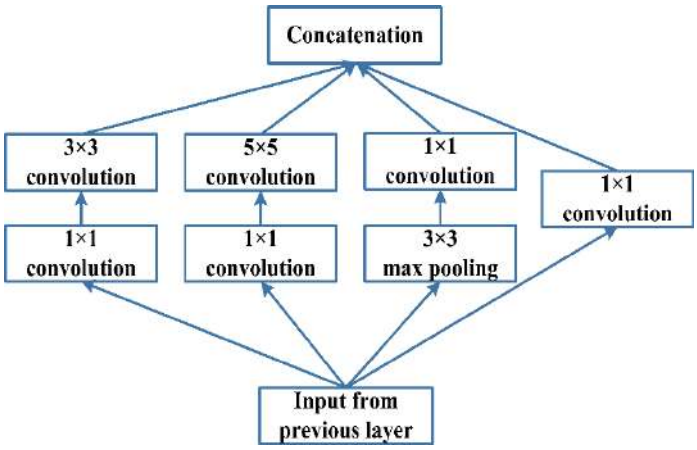


Figure I.15 : Structure de base d'un bloc du réseau GoogleNet.

I.3.4.4. ResNet

He et al ont développé le réseau ResNet (Residual Network), qui a remporté le prix ILSVRC 2015. Leur objectif était de concevoir un réseau ultra-profond par rapport aux réseaux précédents. Plusieurs types de ResNet ont été développés en fonction du nombre de couches (à partir de 34 couches et jusqu'à 1202 couches). Le type le plus courant était ResNet 50, qui comprenait 49 couches convolutifs plus une seule couche FC. Le nombre total de poids du réseau ResNet 50 est de 25,5 M, tandis que le nombre total de MAC est de 3,9 M. La nouvelle idée de ResNet est son utilisation du concept de voie de contournement, comme le montre la Figure I.16 pour résoudre le problème de la formation d'un réseau plus profond en 2015.

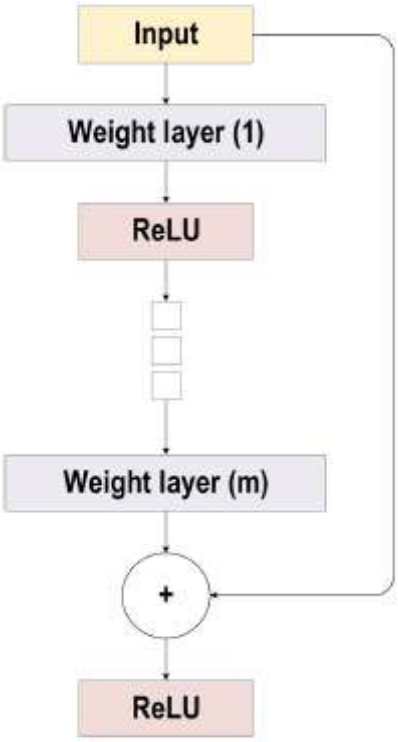


Figure I.16 : Principe de l'architecture du réseau ResNet.

I.4. Conclusion

Nous avons commencé ce chapitre par la présentation des concepts de l'apprentissage profond (définition, Architectures...etc.).

Ensuite, nous avons étudié une classe importante de réseaux de neurone qui ont connu un grand succès dans le domaine de classification des images. Il s'agit de réseaux de neurone convolutif CNN. Ce type de réseau est basé essentiellement sur l'opération de convolution qui permet d'extraire les caractéristiques significatives d'une image. Généralement, une couche de convolution est suivie d'une couche de pooling pour réduire la taille des maps de caractéristiques toute en gardant les informations essentielles. La partie de convolution est suivie d'une série de couches entièrement connectées. Le rôle de ces dernières et de faire la classification en se servant des caractéristiques extraites par la partie de convolution. 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.

En se basant sur ce principe architecturelle une multitude de versions de réseaux CNN ont été proposées ces dernières années. L'utilisateur trouve une large panoplie de choix pour répondre aux recommandations de ces applications.

Chapitre II :

Les circuits logiques programmables FPGA.

II.1. Introduction

Au cours des trois dernières décennies, les circuits FPGAs (field-programmable gate arrays) sont passés de petites puces avec quelques milliers de blocs logiques à un système sur puce à des milliards de transistors. Les FPGA modernes offrent un parallélisme massif à grain fin, une efficacité énergétique élevée et des performances plus prévisibles que les processeurs CPU et GPU. Plus important encore, ils peuvent être reconfigurés pour implémenter une architecture à usage spécial qui est spécifiquement personnalisée en fonction des caractéristiques clés d'une application (ou d'un domaine) cible. Pour ces raisons, ces dernières années ont vu une utilisation croissante des FPGA pour l'implémentation des algorithmes avancés de traitement de signal, où une pléthore de nouvelles architectures, compilateurs et applications accélérées par FPGA sont en cours de démonstration et de développement actif. [19]

Dans ce chapitre, nous allons étudier les circuits logiques programmables FPGA, leurs matérielles et leurs logicielles.

II.2. Évolution de la Logique Programmable

Les premiers dispositifs programmables par l'utilisateur appelés PLD (programmable logic devices) qui ont été développés en 1978 par MMI pouvaient remplacer une dizaine de portes TTL et étaient programmables une fois. Cela a conduit aux PLD reprogrammables basés sur les technologies EEPROM ou EPROM. En 1985, la lithographie avancée a permis une nouvelle classe d'appareils, les FPGA.

Les FPGA ont introduit deux nouvelles fonctionnalités d'architecture importantes : un routage programmable pour interconnecter le nombre croissant de portes sur un périphérique et un bloc programmable appelé LUT ou table de recherche avec un registre associé. Les dispositifs initiaux de Xilinx contenaient jusqu'à une centaine de paires LUT et flip-flop dans un élément logique de base appelé CLB ou bloc logique configurable. Plutôt que d'utiliser une mémoire EPROM ou EEPROM programmée en permanence, les FPGA Xilinx s'appuyaient sur des mémoires CMOS pour contenir les informations de programmation. Le FPGA a pris sa place en tant que composant central dans les systèmes numériques, remplaçant les PLD et TTL pour la mise en œuvre de la logique de colle. Dans les années 1990, de nouvelles utilisations ont commencé à émerger pour les FPGA. L'Internet émergeant est devenu un moteur de croissance pour les FPGA, les FPGA étant utilisés pour le prototypage, le déploiement initial et la production à grande échelle de commutateurs et de routeurs Internet. En 2000, les systèmes de communication constituaient le principal marché des FPGA. D'autres nouveaux marchés

pour les FPGA ont également émergé pour le prototypage ASIC et des systèmes DSP (processeur de traitement numérique du signal) à haute performance. Les FPGA ont également commencé à être utilisés pour mettre en œuvre des processeurs de contrôle logiciel tels que le MicroBlaze Xilinx et les architectures PicoBlaze. [20]

II.3. Architecteur de circuit FPGA

II.3.1. Structure de base d'un circuit FPGA

Un FPGA classique se compose d'une matrice de blocs logiques configurables (CLB) entourés de blocs d'entrée/sortie (E/S). L'ensemble de tous les blocs est connecté par un réseau d'interconnexions programmables, comme montré sur la Figure II.1. [21]

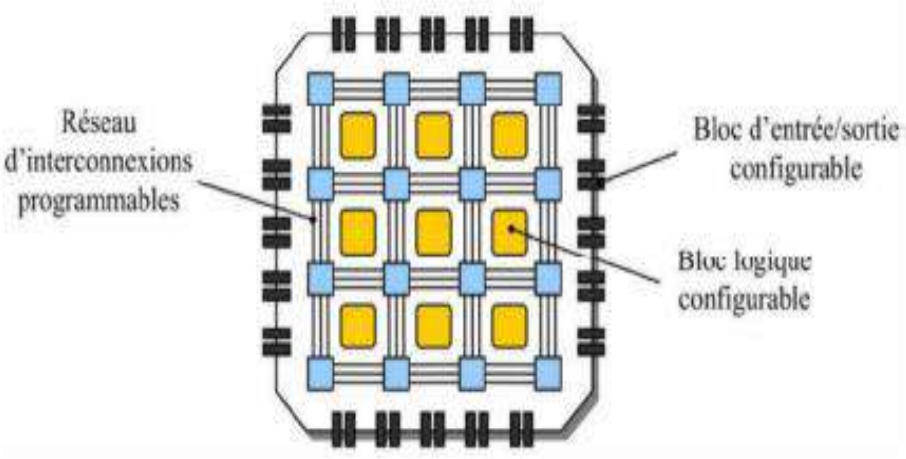


Figure II.1 : Schéma simplifié de l'architecture de base d'un FPGA classique.

- **Les blocs logiques** constituent le cœur du FPGA. Ces cellules sont constituées d'éléments logiques programmables disposés sous forme matricielle. Chaque bloc logique est identique aux autres et peut être relié à ses voisins par le biais d'un réseau d'interconnexion.
- **Le réseau d'interconnexion des blocs** relie les éléments logiques entre eux afin de réaliser des fonctions complexes. Au sein d'un FPGA, les interconnexions sont souvent hiérarchisées, où chaque niveau de hiérarchie supporte une fréquence de transmission différente.
- **Les entrées/sorties configurables** sont des cellules permettant d'interfacer le FPGA avec l'environnement extérieur. Chaque bloc d'entrée/sortie contrôle une broche du composant et peut être défini en entrée, en sortie, en signal bidirectionnel ou être inutilisé (haute impédance).

II.3.2. Les composants d'un circuit FPGA

En général, les FPGA sont des puces de silicium programmables avec un ensemble de blocs. La figure II.2 représenté les déférents blocs et composants d'un circuit FPGA. [22]

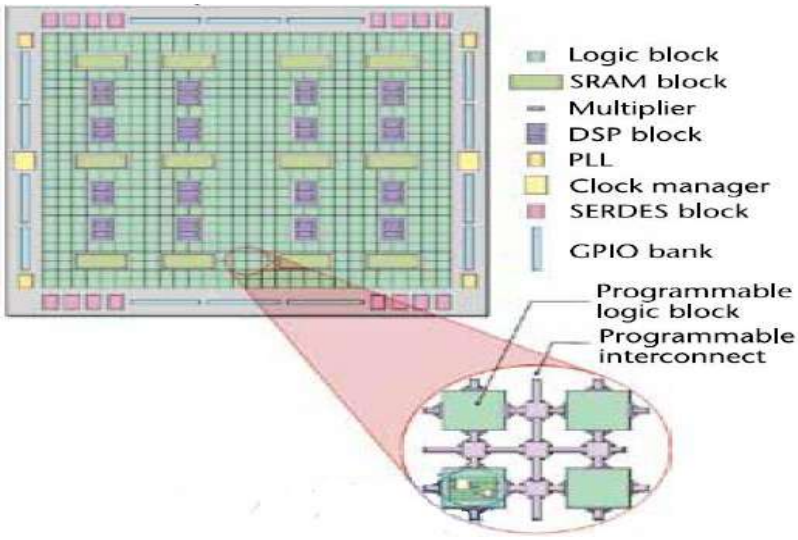


Figure II.2 : Composants d'un FPGA. [23]

II.3.2.1. Blocs logiques configurables (CLB)

Sont les ressources logiques répétitives de base sur un FPGA. Lorsqu'ils sont reliés entre eux par les blocs de routage programmables, les CLB peuvent exécuter des fonctions logiques complexes, implémenter des fonctions de mémoire et synchroniser le code sur le FPGA. Les CLB contiennent des composants plus petits, notamment des bascules Flip-Flop (FF), des tables de correspondance (Look-Up Table : LUT) et des multiplexeurs (MUX). Une bascule FF est la plus petite ressource de stockage sur le FPGA.

- Chaque FF dans un CLB est un registre binaire utilisé pour enregistrer les états logiques entre les cycles d'horloge sur un circuit FPGA.
- Une LUT stocke une liste prédéfinie de sorties pour chaque combinaison d'entrées. Les LUT fournissent un moyen rapide de récupérer la sortie d'une opération logique car les résultats possibles sont stockés puis référencés plutôt que calculés.
- Un MUX est un circuit qui sélectionne entre deux entrées ou plus, puis renvoie l'entrée sélectionnée.

Toutes les opérations logiques peuvent être implémentées en utilisant la combinaison de FF, LUTs et MUX. La figure II.3 montre la configuration de base d'un bloc CLB. [24]

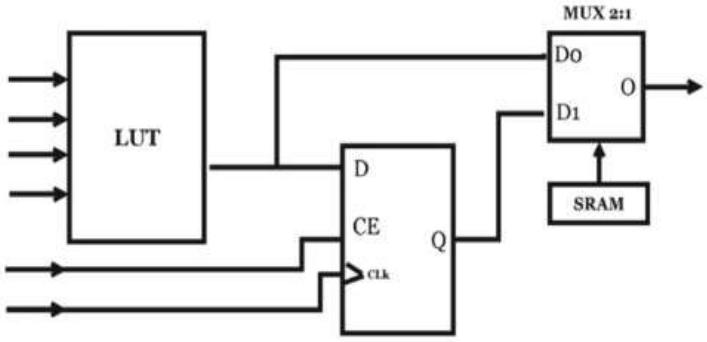


Figure II.3 : Configuration de base d'une cellule CLB d'un FPGA. [25]

II.3.2.2. Les blocs d'E/S (IOB)

Les blocs d'entrées sorties (Input Output Blocks IOB) sont utilisés pour acheminer les signaux externes vers l'intérieur du circuit FPGA (signaux d'entrée) et pour renvoyer les signaux produits à l'intérieur du circuit vers la sortie (signaux de sortie). Un IOB se compose d'un tampon d'entrée et d'un tampon de sortie avec des commandes de sortie à trois états. La polarité de la sortie peut généralement être programmée pour une sortie active haute ou active basse. Il existe des bascules typiques sur les sorties afin que les signaux synchronisés puissent être émis directement sur les broches sans rencontrer de retard important, répondant plus facilement aux exigences de temps de configuration pour les périphériques externes. Le schéma bloc d'un IOB de Xilinx est illustré dans la figure II.4.

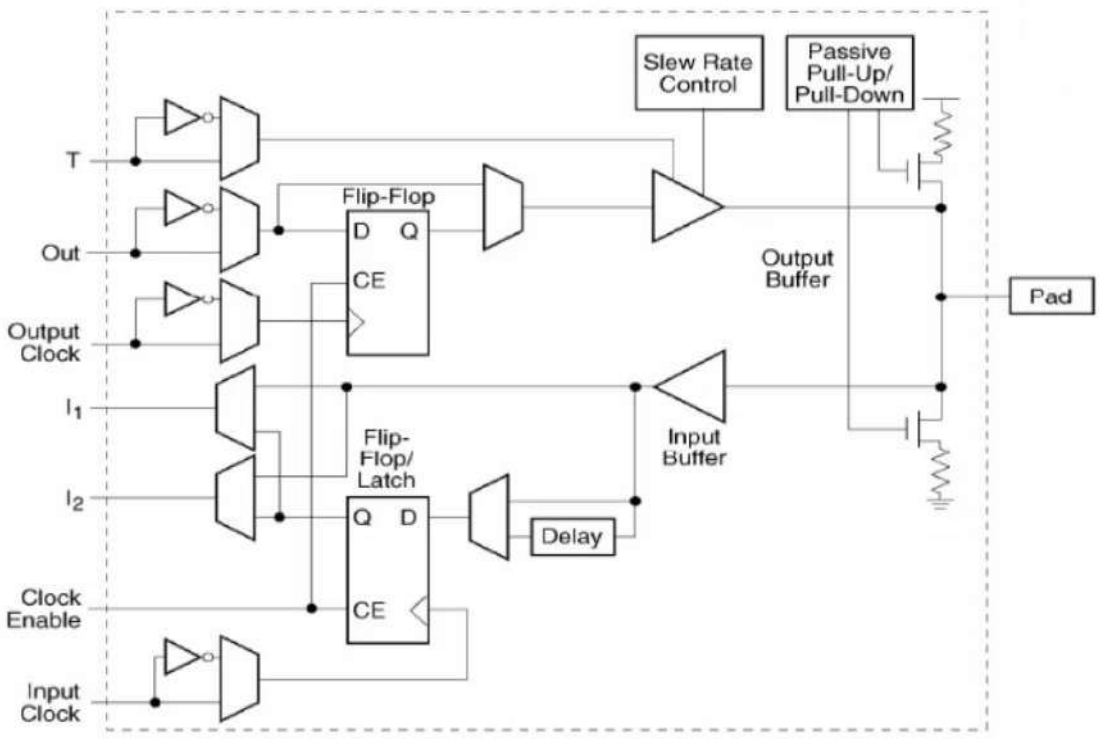


Figure II.4 : structure d'un IOB de Xilinx.

II.3.2.3. Blocs d'interconnexion

La topologie du réseau d'interconnexion diverge d'un FPGA à l'autre, bien que les topologies récentes utilisent une représentation hiérarchique comprenant plusieurs niveaux d'interconnexion. Il existe différents types de connexions possibles, illustrés sur la Figure II.5 :

- **Les interconnexions directes** sont des liaisons entre blocs logiques ou entre un bloc logique et une broche d'entrée/sortie.
- **Les lignes** parcourent toute la longueur et la largeur du FPGA. Ces lignes peuvent être de différent type selon leur dimension (exemple : single-length, double-length ou longlines pour un FPGA Xilinx XC4000). Les lignes courtes offrent plus de flexibilité et un routage rapide entre des blocs adjacents, mais traversent forcément toutes les matrices d'interconnexion. Lorsque les connexions sont éloignées, le passage par les matrices d'interconnexion ralentit la propagation des signaux dans le FPGA, limitant la fréquence maximale de fonctionnement de l'application. L'utilisation des lignes longues (traversant moins de matrices d'interconnexions) permet d'obtenir une propagation des signaux plus rapides entre deux blocs éloignés.
- **Les matrices d'interconnexion** permettent de relier un bloc à n'importe quel autre bloc du FPGA. Ces systèmes sont des aiguilleurs configurables situés à chaque intersection et raccordent les lignes entre elles selon diverses configurations.

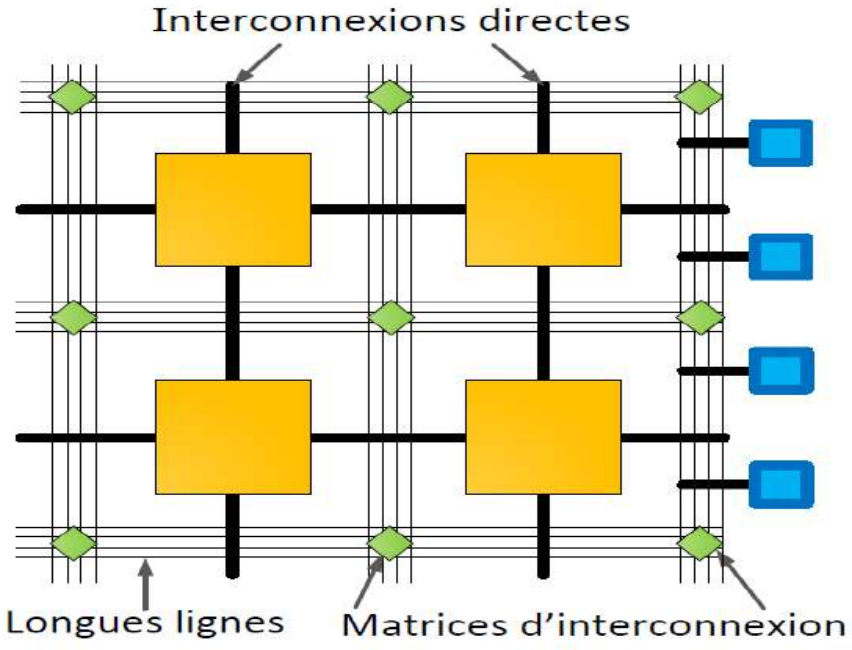


Figure II.5 : Interconnexion interne d'un circuit FPGA.

II.3.2.4. Les blocs RAM (BRAM)

Les blocs RAM sont des mémoires embarquées sur le circuit FPGA, utilisées pour stocker de grandes quantités de données à l'intérieur du circuit. La taille de chaque bloc RAM dépend de la catégorie du circuit FPGA et chaque bloc RAM peut être configuré en plusieurs sous blocks de différentes tailles.

Un BRAM est une partie discrète d'un FPGA, ce qui signifie qu'il n'y en a qu'un nombre limité disponible sur la puce. Chaque FPGA a une quantité différente, donc selon l'application, le concepteur de l'application à base de circuit FPGA doit savoir combien il aura besoin de BRAM pour sélectionner le circuit FPGA à utiliser. Les BRAMs sont utilisées pour l'implémentation des mémoires RAM à port unique (Single Port BRAM Configuration), des mémoires RAM à double port (Dual Port BRAM Configuration) et des registres FIFO (FIFO BRAM Configuration).

II.3.2.5. Les blocs DSP

Les blocs DSP sont des unités arithmétiques optimisées en performance et en consommation d'énergie pour effectuer des opérations de multiplications/accumulations (MAC). Différentes précisions sont possibles (9, 18, 27 bits pour les opérandes). Il est également possible de faire des calculs directement sur des nombres complexes en utilisant ce type de ressources. Le nombre de blocs DSP embarqués sur un circuit FPGA dépend de sa catégorie. La Figure II.6 montre l'architecture interne d'un bloc DSP48E de Xilinx.

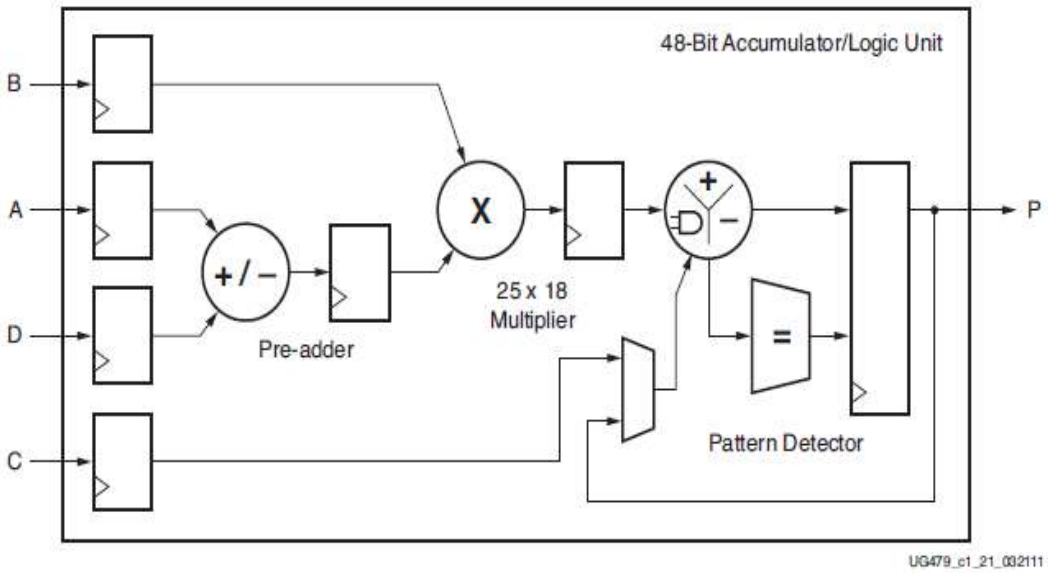


Figure II.6 : architecture simplifiée d'un bloc DSP48E.

II.3.2.6. Les processeurs embarqués

Beaucoup d'applications nécessitent l'utilisation d'un processeur pour faciliter l'exécution de certaines tâches. Pour répondre à ce besoin, les fabricants des circuits FPGA ont intégré des processeurs embarqués. L'utilisation d'un circuit FPGA qui contient un processeur embarqué (comme le Virtex 5 de Xilinx) 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 et microBlaze [26] .

II.3.2.7. Gestionnaire d'horloge numérique (DCM)

Un gestionnaire d'horloge numérique (Digital Clock Manager DCM) est un composant électronique disponible sur certains types de circuits FPGA, notamment ceux produits par Xilinx. Il est utile pour manipuler les signaux d'horloge à l'intérieur du FPGA et pour éviter un décalage d'horloge qui introduirait des erreurs dans le circuit. Le DCM est un dispositif important pour le traitement de l'horloge à l'intérieur d'un FPGA. Les gestionnaires d'horloges numériques ont les applications suivantes : [11, 27]

- Multiplier ou diviser une horloge entrante (qui peut provenir de l'extérieur du FPGA ou d'un synthétiseur de fréquence numérique.
- Assurez la stabilité du cycle de l'horloge.
- Ajout d'un déphasage avec l'utilisation supplémentaire d'une boucle à verrouillage de retard.
- Élimination du décalage d'horloge dans une conception FPGA.

II.4. Les types de configuration des circuits FPGA

Les FPGA peuvent être classés en fonction du type de leurs commutateurs d'interconnexion programmables : anti-fuse, SRAM et Flash. Chacune des trois technologies comporte des compromis. [24]

II.4.1. Les FPGA anti-fusion

Les FPGA anti-fusion sont non volatils et ont un retard minimal dû au routage, ce qui se traduit par une vitesse plus rapide et une consommation d'énergie réduite. L'inconvénient est évident car ils ont un processus de fabrication relativement plus compliqué et ne sont programmables qu'une seule fois.

II.4.2. Les FPGA basés sur SRAM

Les FPGA basés sur SRAM sont reprogrammables par l'utilisateur. Lors de l'implémentation d'une application sur FPGA basée sur SRAM, on utilise un routage pour établir une communication entre les composants utilisés dans cette conception d'un côté et pour fournir une connexion entre les blocs d'E/S (IOBs) et les ressources logiques utilisées de l'autre côté. La configuration du réseau de routage est assurée par l'utilisation de cellules SRAMs comme indiqué sur la figure II.6. Ce type de circuit FPGA est caractérisé par son niveau élevé de performances. Cependant, ces FPGA sont volatils et peuvent ne pas tenir la configuration en cas de problème d'alimentation. En outre, ils ont des retards de routage plus importants et nécessitent plus de puissance.

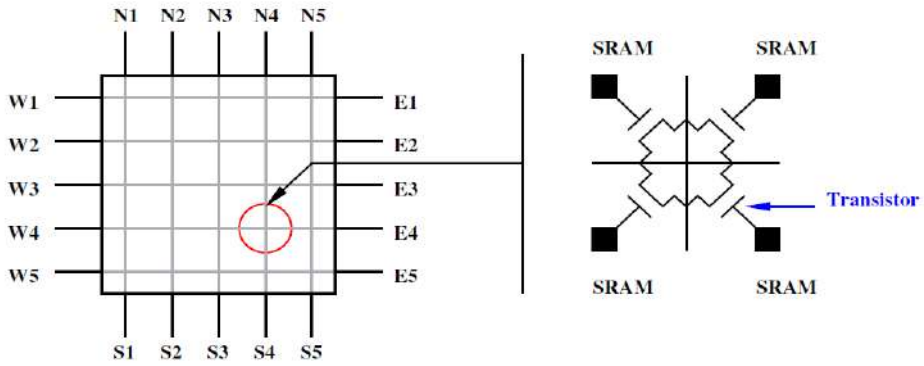


Figure II.7 : Structure générale d'une matrice de routage pour un FPGA à base de SRAM.

II.4.3. Les FPGA à base de Flash

Les FPGA à base de Flash sont non volatils et reprogrammables, et ont également une faible consommation d'énergie et un délai d'acheminement court. L'inconvénient majeur est que la reconfiguration dynamique n'est pas recommandée pour les FPGA basés sur flash en raison des résultats potentiellement destructeurs si des effets de rayonnement se produisent pendant le processus de reconfiguration

II.5. La famille FPGA Virtex-7

Les FPGA de la série 7 de Xilinx comprennent quatre familles qui sont toutes conçues afin de permettre une conception optimale en puissance, performances et coût. La famille Spartan-7 est la plus faible en ressource avec le prix le moins élevé dans le portefeuille de la série 7. La famille Artix-7 est optimisée pour les performances par watt et la bande passante par watt les plus élevées pour les applications à haut volume sensibles aux coûts. La famille Kintex-7 est une classe innovante de FPGA optimisés pour le meilleur rapport qualité-prix. La famille Virtex-7 est optimisée pour des performances et une capacité de système maximales. L'architecture interne de la famille Virtex 7 de Xilinx est illustrée dans la Figure II.7. [28]

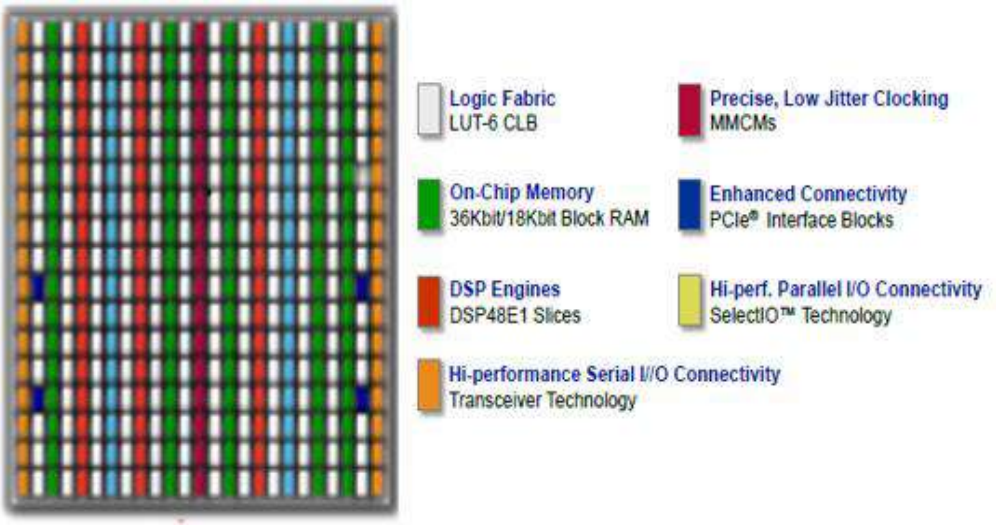


Figure II.8 : Architecture interne de la famille Virtex 7 de Xilinx.

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.9.

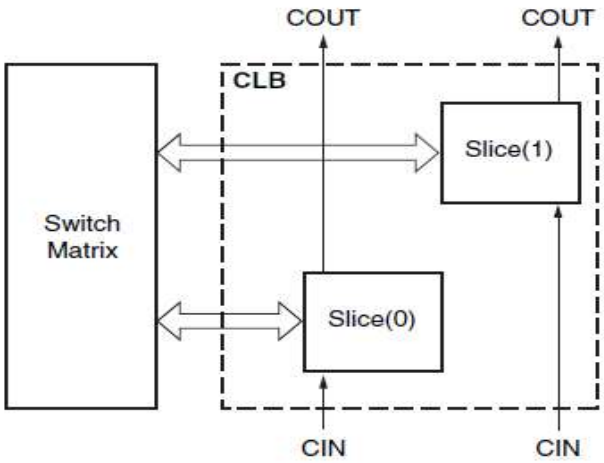


Figure II.9 : Arrangement des slice dans un CLB pour la série 7 de Xilinx.

Selon leur structure interne, les slices sont divisé en deux types :

- ✓ **Slices M** : chaque slice M de la série 7 de Xilinx contient :
  - Quatre générateurs de fonctions logiques LUT.
  - Huit éléments de mémorisation de type Flip Flop.
  - Des multiplexeurs.
  - Une logique de calcul de la retenu.
- ✓ **Slices L :** en plus des éléments contenus dans les Slices M, les slices L contiennent des mémoires distribuées et des registres à décalage.

Chaque CLB peut contenir deux Slices L ou un Slice L et un Slice M. Le tableau II.1 donne le nombre de ressources disponibles dans chaque type de slice

	LUTs	Flip-Flops	Calcul de retenu	Mémoire distribuée	Registres à décalage
Slice L	4	8	1	256 bits	128 bits
Slice M	4	8	1	0	0

Tableau II.1 : nombre de ressources disponibles dans chaque type de slice de la série 7 de Xilinx.

Le constructeur Xilinx a commercialisé plusieurs versions de la famille Virtex 7. Les différentes versions se distinguent par le nombre de ressources disponibles, le niveau de performance, la consommation de l'énergie et bien évidement le prix. Le tableau II.2 donne les ressources disponibles pour les différentes versions de virtex-7.

Versions de virtex-7	Cellules logiques.	CLBs		Slices de DSP	Les blocs BRAM		
		Les slices	RAM distribuée Max (kb)		18 kb	36 kb	Max (kb)
XC7V585T	582720	91050	6938	1260	1590	795	28620
XC7V2000T	1954560	305400	21550	2160	2584	1292	46512
XC7VX330T	326400	51000	4388	1120	1500	750	27000
XC7VX415T	412160	64400	6525	2160	1760	880	31680
XC7VX485T	485760	75900	8175	2800	2060	1030	37080
XC7VX550T	554240	86600	8725	2880	2360	1180	42480
XC7VX690T	693120	108300	10888	3600	2940	1470	52920
XC7VX980T	979200	153000	13838	3600	3000	1500	54000
XC7VX1140T	1139200	178000	17700	3360	3760	1880	67680
XC7VH580T	580480	90700	8850	1680	1880	940	33840
XC7VH870T	876160	136900	13275	2520	2820	1410	50760

Tableau II.2 : Ressources disponibles pour les différentes versions de Virtex-7.

II.6. Environnement de travail

II.6.1. Vivado

La suite d'outils de conception Vivado contient des services qui prennent en charge toutes les phases de conception des FPGA, à commencer par la saisie de la conception, la simulation, la synthèse, le placement et le routage, la génération de flux de bits (bitstream), le débogage et la vérification, ainsi que le développement de logiciels ciblés pour ces FPGA. Le concepteur peut interagir avec l'environnement Vivado de plusieurs manières. Cela inclut une interface graphique pour les utilisateurs interactifs, ainsi qu'une interface de ligne de commande. Vivado prend également en charge une interface de script avec un riche ensemble de commandes Tcl. Ces multiples modes d'interaction peuvent également être combinés de différentes manières pour répondre aux besoins exacts des utilisateurs. La Figure II.10 donne une vue globale de l'environnement de visualisation Vivado IDE. [20]

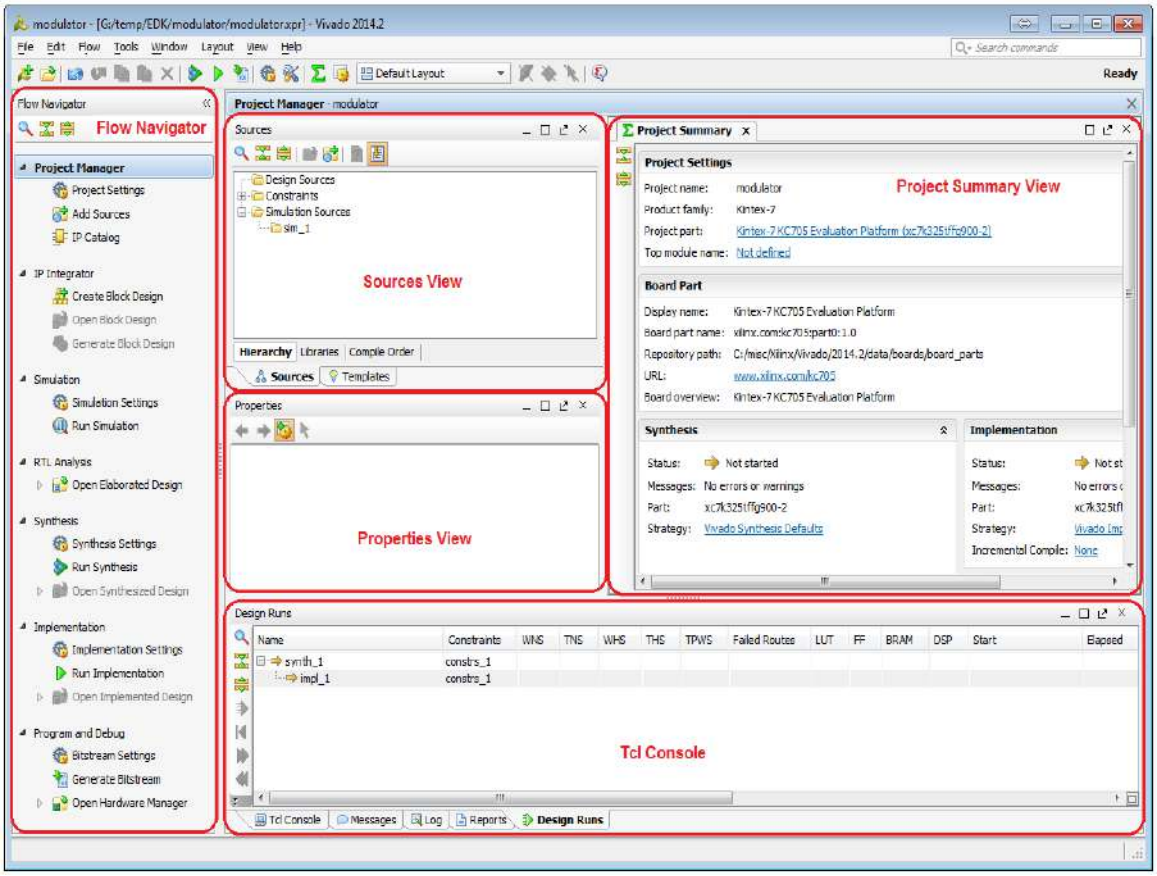


Figure II.10 : L'environnement de visualisation de l'IDE de Vivado.

II.6.2. System Générateur (XSG)

Bien que les langages de description des matériels récents, comme le VHDL et le VERILOG, simplifient le cycle de développement des systèmes numériques, l'implémentation et la réalisation d'une conception sur une plateforme matérielle réelle reste toujours un cauchemar face aux ingénieurs. En effet, l'utilisation des approches de conception classique pour l'implémentation des systèmes numériques sophistiqués est indésirable. Ce type d'approches rend le cycle de développement d'un projet long et complexe. D'où la nécessité d'une plateforme de prototypage rapide qui permet aux ingénieurs de mettre l'accent sur la manière de concevoir le système, au lieu de se concentrer sur l'écriture du code VHDL et le débogage du code généré. Pour ces raisons, XILINX a introduit la plateforme SYSTEM GENERATOR FOR DSP (XSG). [29]

System générateur (XSG) est un outil logiciel de haut niveau qui permet l'utilisation de l'environnement MATLAB/Simulink de créer et vérifier les conceptions matérielles des FPGA Xilinx rapidement et facilement. Il fournit une bibliothèque de blocs Simulink pour la modélisation précise des bits et des cycles pour les fonctions arithmétiques et logiques, les mémoires et les fonctions DSP. Il comprend également un générateur de code qui génère automatiquement du code HDL à partir du modèle créé. Le code HDL généré peut être synthétisé et implémenté dans les FPGA Xilinx. Les blocs XSG sont comme les blocs Simulink standard, sauf qu'ils ne peuvent fonctionner qu'en temps discret et en précisant le format de représentation des données (virgule fixe, virgule flottante, binaire…). [30]

II.7. Réseaux CNN sur FPGA

II.7.1. Exemples de travaux en relation

L'un des défit pour l'implémentation des techniques de Deep Learning sur les circuits de type FPGA est la taille de conception. En effet, une application à base de Deep Learning nécessite généralement un nombre important d'opérations arithmétiques (multiplication, addition…) et un espace mémoire de grande capacité pour sauvegarder les valeurs des pondérations et les résultats des calculs intermédiaires. L'implémentation de grands réseaux de neurones n'a pas toujours été possible. Cependant, alors que les FPGA modernes ont connus un progrès énorme pour augmenter le nombre de ressources et incorporer des unités de calcul très performantes aux côtés de la structure FPGA générique, des réseaux profonds ont commencé à être implémentés sur des systèmes FPGA uniques. [31]

Les premières implémentations FPGA de réseaux de neurones ont commencé à apparaître au début des années 1990. Cependant, les premières implémentations FPGA de réseaux CNN ont commencé à apparaître quelques années plus tard. Cloutier et all [32] ont été parmi les premiers à explorer ces efforts, mais ont été fortement limités par les contraintes de taille des FPGA à l'époque, conduisant à l'utilisation de l'arithmétique de faible précision. Depuis lors, la technologie FPGA a considérablement changé en termes de ressources et de performances. Cette évolution a encouragé les chercheurs pour exploiter les capacités des circuits FPGA pour faire des implémentations efficaces de la technique CNN.

Un groupe de l'Institut des semi-conducteurs de l'Académie chinoise des sciences à Pékin en Chine a tenté une petite implémentation en 2015. Leur implémentation a fonctionné sur une carte FPGA Altera Arria V fonctionnant à 50 MHz. Les images d'entrée étaient de taille 32x32, utilisaient un format de données à point fixe de 8 bits et utilisaient 3 couches de convolution avec activation, 2 couches de regroupement et 1 classificateur Softmax. Ce travail a implémenté des éléments de traitement personnalisés qui pourraient être reconfigurés en cas de besoin. [33]

Un effort conjoint entre l'Université de Tsinghua et l'université de Stanford en 2016 a abouti au système "Angel-Eye". Pour accélérer l'opération de convolution, ce système fonctionne sur la plate-forme Xilinx Zynq XC7Z045 et utilise un ensemble d'éléments de traitement. Le circuit FPGA fonctionne à une fréquence d'horloge de 150 MHz, utilise un format de données en virgule fixe 16 bits, consomme une puissance de 9,63 watts et atteint des performances de 187,80 GFLOPS lors de l'exécution du ConvNet VGG16. L'utilisation des ressources n'a pas été précisée dans leurs travaux publiés. Le projet qui a produit Angel-Eye a également créé un compilateur personnalisé qui a tenté de minimiser l'accès à la mémoire externe et ainsi de réduire la latence inhérente aux transactions de mémoire. [33]

Un autre travail d'un groupe de l'Université Purdue en 2016 a développé un accélérateur de réseau neuronal convolutif CNN qui utilisait des outils logiciels personnalisés. Ces outils logiciels étaient essentiellement des compilateurs qui exploraient des méthodes d'optimisation pour améliorer les performances du réseau de neurones. Ce travail a utilisé le FPGA Xilinx Kintex-7 XC7K325T, il a atteint des performances de 58 à 115 GFLOPS et a exécuté un ConvNet avec le même nombre de couches qu'AlexNet mais avec des canaux plus petits. L'utilisation des ressources pour leur implémentation FPGA n'a pas été spécifiée. [33]

II.7.2. Les techniques d'optimisation de CNN sur FPGA

Les techniques d'optimisation de réseau CNN sur FPGA peuvent être groupées en quatre catégories. La première catégorie englobe les techniques qui utilisent des algorithmes qui permettent d'optimiser les calculs. La deuxième catégorie concerne les méthodes d'amélioration de l'efficacité de l'opération de convolution sur un FPGA. La troisième catégorie réuni l'ensemble des techniques qui permettent de réduire la taille du modèle et donc le nombre de paramètres et d'opérations nécessaires. La Figure II.11 montre les différentes techniques d'optimisation. [16]

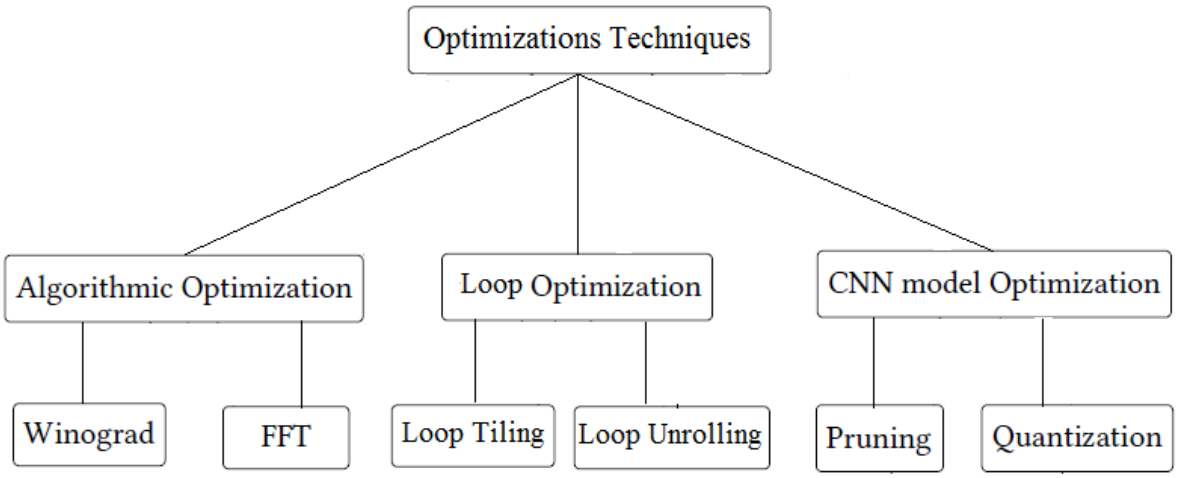


Figure II.11 : les différentes techniques d'optimisation.

II.7.2.1. Optimisations algorithmiques

Selon Shawahna et al, 90% du temps de calcul dans un CNN est consommé par l'opération de convolution. Par conséquent, la diminution du nombre d'opérations dans une convolution réduira la complexité de calcul de CNN.

Selon Liang et al, il existe deux algorithmes de convolution rapide qui exécutent efficacement la convolution 2D : l'algorithme de filtre minimal de Winograd et la Transformée de Fourier rapide (FFT).

L'idée derrière ces algorithmes de convolution rapide est de transformer le noyau et l'entrée en un domaine cible, où la convolution correspond à une multiplication par éléments dans ce domaine. En fin de compte, nous pouvons obtenir la sortie correspondante en appliquant la transformation inverse sur la sortie de produit. En conséquence, l'algorithme rapide réduit le nombre total d'opérations en calculant une multiplication par élément (plutôt qu'une convolution).

II.7.2.2. Technique d'optimisation de boucle

Les chercheurs ont exploré diverses techniques d'optimisation de boucle pour réduire la consommation de mémoire ainsi que le mouvement des données. De cette façon, ils ont maximisé l'utilisation des ressources afin d'atteindre des performances élevées. En général, il existe deux techniques principales d'optimisation de boucle :

- **Tuilage de boucle :** la mémoire sur puce du FPGA est limitée au stockage de toutes les données intermédiaires et d'entrée. Par conséquent, il est nécessaire d'utiliser des DRAM externes pour stocker les poids et une partie des valeurs de pixels intermédiaires des couches CNN. Le tuilage de boucle est proposé pour diviser les données en plusieurs blocs à intégrer dans des tampons sur puce. [34]
- **Déroulement de boucle :** il détermine le nombre de calculs parallèles. Quatre types de configuration de déroulement peuvent être effectués (en fonction de la boucle à dérouler). Les différentes configurations sont visibles sur la Figure II.12. Comme le FPGA a des ressources limitées, il n'est pas toujours possible de dérouler complètement une boucle. Les paramètres de déroulement sont (Pkx, Pky), Pif, (Pix, Piy) et Pof, qui représentent la partie des données des quatre boucles stockées dans des tampons sur puce.

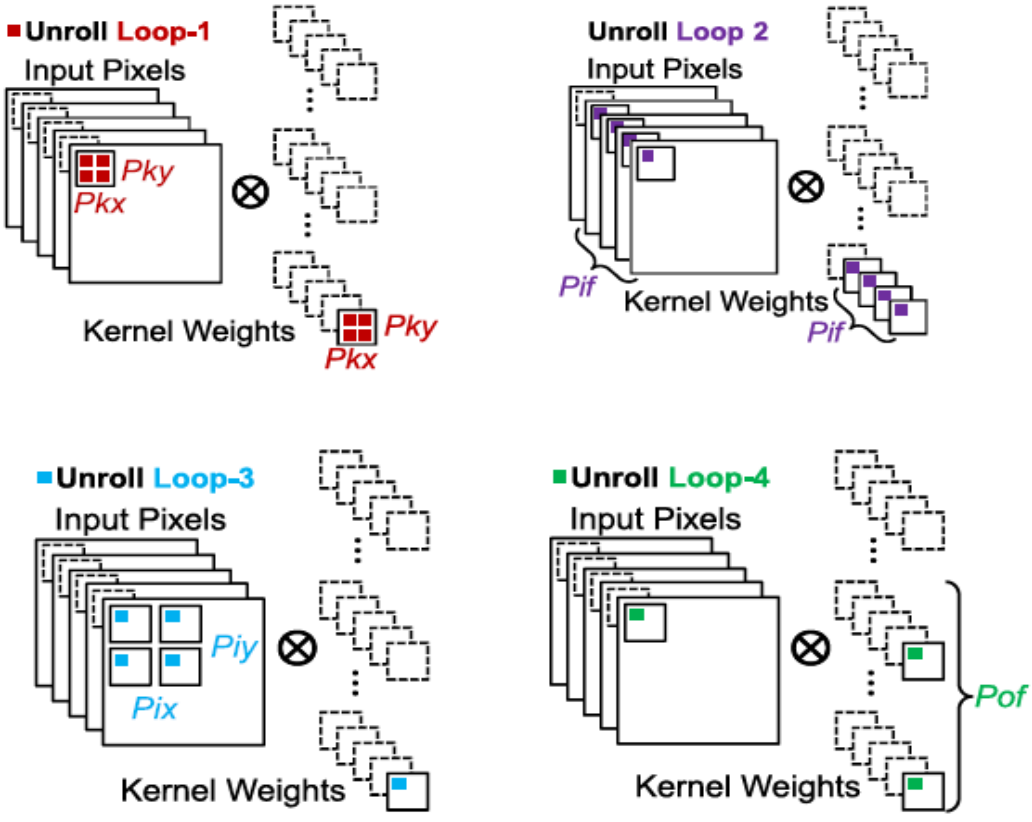


Figure II.12 : Les différentes configurations de déroulement de boucle.

II.7.2.3. Optimisation du modèle CNN

Les problèmes majeurs, lors de l'implémentation d'un CNN sur un FPGA, sont la taille du CNN et sa complexité de calcul. Des recherches ont été menées pour développer des techniques abordant ces deux problèmes en modifiant directement l'architecture CNN. Les techniques d'optimisation sont :

- **Élagage :** La méthode d'élagage supprime les connexions redondantes en apprenant la connectivité à l'aide d'une formation réseau standard. Ensuite, toutes les connexions dont le poids est inférieur à un seuil (valeur absolue) sont supprimées du réseau. Enfin, le réseau est recyclé pour apprendre les poids finaux des connexions restantes.
- **Quantification :** La quantification est une approche pour échanger la précision contre une diminution des besoins de stockage d'un réseau. En effet, on peut définir la quantification comme la réduction du nombre de bits représentant une donnée. De plus, au lieu d'utiliser un nombre à virgule flottante, nous pouvons utiliser un nombre à virgule fixe. Comme indiqué précédemment, les nombres à virgule fixe sont connus pour être plus efficaces sur du matériel tel que le FPGA car nous pouvons utiliser l'arithmétique entière. La quantification en nombres à virgule fixe peut alors réduire les besoins en mémoire et la latence de l'étape d'inférence.

II.8. Conclusion

Ce chapitre a été consacré à la présentation des circuits logiques programmables FPGA et leur utilisation pour l'implémentation des réseaux CNN. Nous avons commencé par la présentation des différents blocs constituants l'architecture de base de ce type de circuits. En plus des trois éléments classiques qui sont les CLB, les IOB et les interconnexions, l'évolution des FPGA a permis l'intégration de bloc permettant une amélioration importante de performance comme les blocs DSP, le processeurs embarqués et les BRAM . Nous avons vue aussi que le développement d'une application à base de FPGA est facilité par l'introduction de plusieurs logiciels de conception. Enfin nous avons présenté un aperçu des différentes techniques qui visent à réaliser une implémentation efficace des réseaux de type CNN sur les circuits FPGA. Ces techniques sont basées sur : des méthodes pour réduire la complexité de calcul, des méthodes pour améliorer l'efficacité de l'opération de convolution sur un FPGA, et des techniques pour réduire la taille du modèle et donc le nombre de paramètres et d'opérations nécessaires.

Chapitre III :

Implémentations, optimisation et résultats.

III.1. Introduction

Jusqu'à présent, nous avons examiné le contexte nécessaire à l'apprentissage en profondeur, nous avons détaillé les réseaux de Neurone Convolutif CNN. Ensuite, nous avons détaillé les circuits FPGA. Examinons maintenant les spécificités et les étapes de la mise en œuvre de ce projet.

Ce chapitre décrira comment implémenter des réseaux de Neurone Convolutif profonds dans un circuit FPGA d'une manière efficace. Nous allons commencer par la présentation de réseau CNN proposée pour l'application de reconnaissance automatique de digits manuscrits. Nous avons ciblé cette application car elle ne nécessite pas des réseaux de neurones très compliqués, on plus la base de donnée nécessaire pour le test et l'apprentissage du réseau pour cette application est disponible gratuitement sur internet. Ensuite, on passe à l'explication détaillée des architectures de conception du réseau CNN. Pour la conception des circuits nous avons utilisé un environnement mixte Matlab, System Générateur et Simulink. L'étape suivante consiste à optimiser les circuits pour l'adaptation sur FPGA. À la fin, nous avons passé à l'étape d'implémentation en utilisant l'outil Vivado de Xilinx. La carte FPGA choisi pour réaliser notre application est le Virtex7 XC7VX485T-2FFG1761C.

III.2. Le réseau CNN proposé

Le but de notre projet est l'implémentation sur FPGA d'un réseau CNN qui fait la reconnaissance automatique des digits manuscrits. La reconnaissance consiste à classer les images de chiffres manuscrits dans l'une des dix classes possibles allant de zéro à neuf.

Dans le cadre de notre application, nous avons utilisé la base de données MNIST, une base de données de chiffres manuscrits largement utilisée en apprentissage automatique. La base MNIST contient 70 000 images en niveaux de gris de 28x28 pixels. Elle est divisée en dix sous-ensembles, chacun contenant environ 7 000 images.

Pour notre application, nous recherchons un compromis entre la précision de la classification et la complexité d'implémentation du réseau. Pour cela, nous avons choisi une architecture simple et efficace. La structure détaillée du réseau utilisé est donnée dans la Figure III.1.

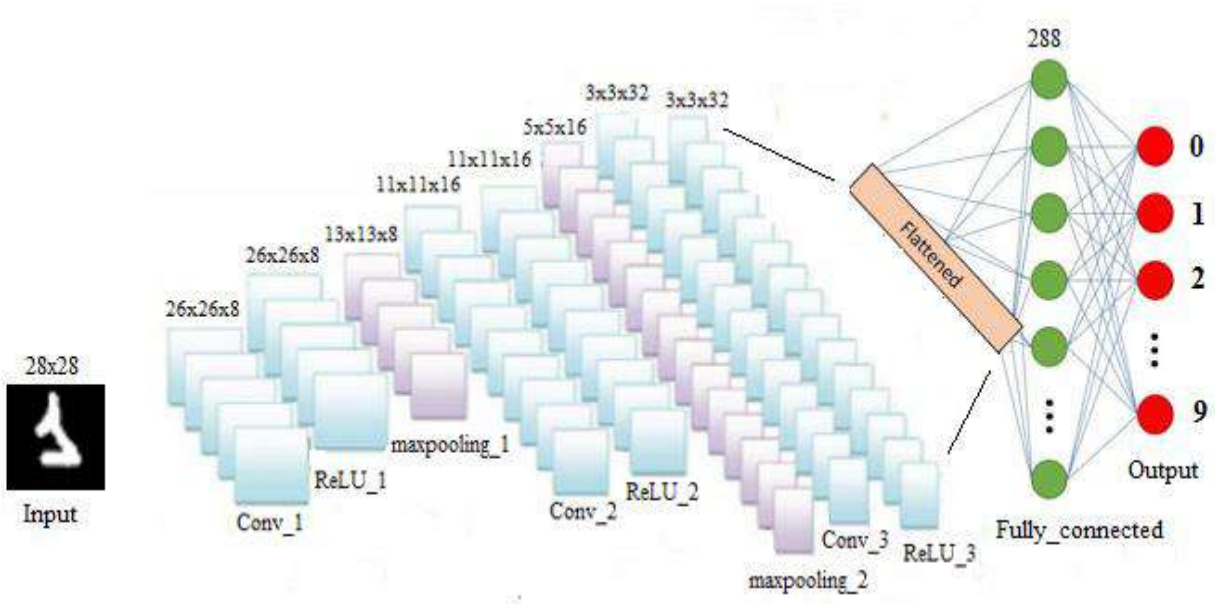


Figure III.1 : La structure du réseau proposé.

Le modèle est composé de 12 couches. L'image d'entrée (Input) est une image en niveau de gris codé sur 8 bits sous format png de taille 28×28. Cette image passe d'abord à la première couche de convolution. Cette couche est composée de 08 filtres de taille 3×3 et de pas 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 l'opération de 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 Max pooling, 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 conception, l'apprentissage et l'évaluation de la précision de classification du réseau sont effectués sous l'environnement Matlab et en utilisant les fonctions du Deep Learning toolbox. La Figure III.2 montre l'évolution des graphes de la précision de classification (Accuracy) et de la fonction perte (Loss) durant la phase d'apprentissage.

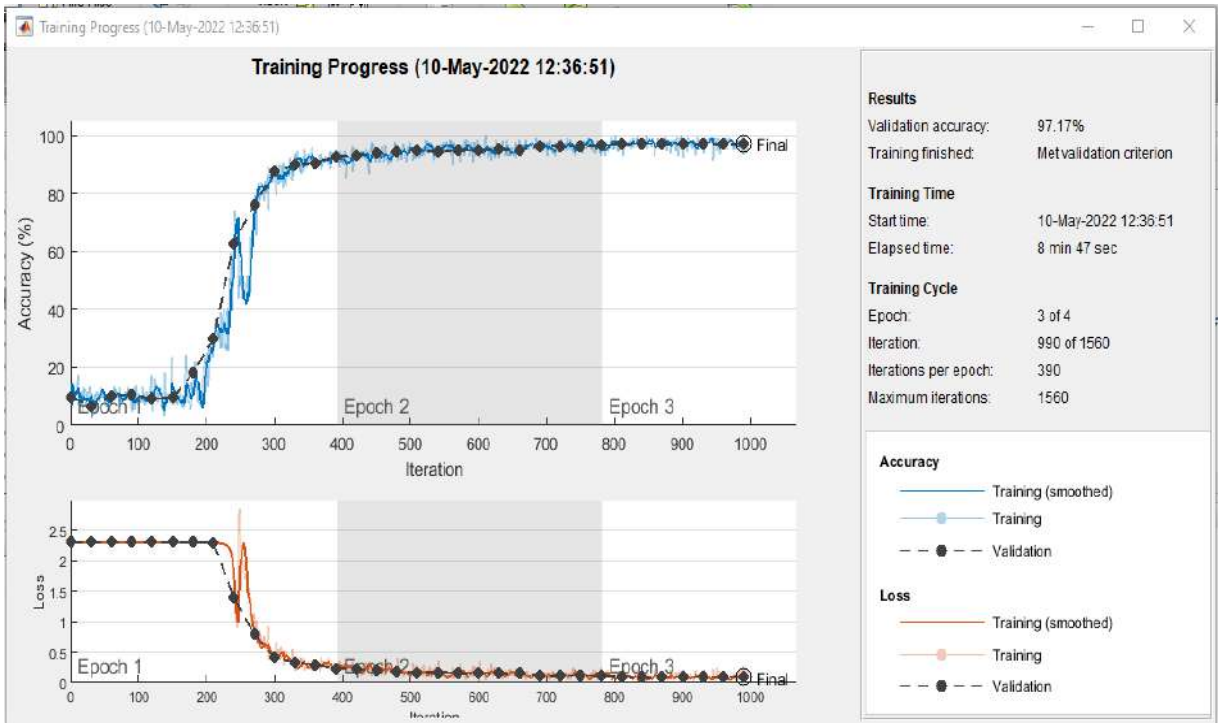


Figure III.2 : Fenêtre de progression de l'apprentissage.

A la fin de l'apprentissage on obtient une précision de 97.17% ce qui est largement acceptable vue la simplicité du réseau. Donc il est possible de classifier n'importe quelle image de digit.

Après la fin de la phase d'apprentissage, nous avons sauvegardé le réseau afin d'utiliser ces paramètres dans les circuits d'implémentation.

III.3. Conception des circuits de calcul

Cette étape est pratiquement le cœur de notre travail dans laquelle nous avons fait la conception des différents circuits qui vont effectuer les différentes opérations du réseau CNN. Ces circuits vont être ensuite convertis en RTL (registre transfert Level) et implémentés sur le circuit FPGA cible.

Pour la conception des circuits de calcul de la convolution, de ReLU et de Max pooling nous avons utilisé le logiciel System Générateur et pour faire la mise en forme des données après chaque étape de calcul nous avons utilisé le logiciel Matlab.

III.3.1. 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.3, le circuit est composé de huit parties identiques notées conv_1 à conv_8 dont chaque partie est chargée de calculer une seule sortie de convolution.

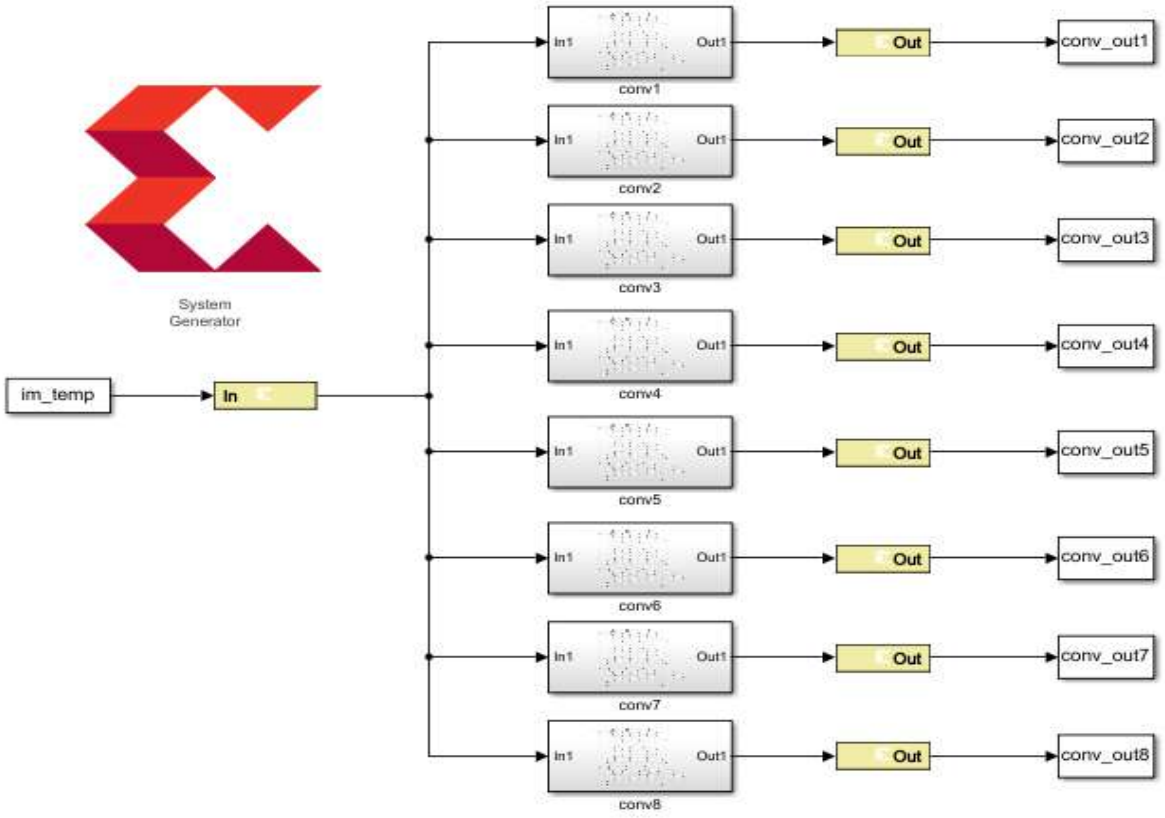


Figure III.3 : Schéma bloc du circuit de calcul de la convolution 1.

Chaque partie contient : 9 multiplieurs de nombres réels, 9 additionneurs de nombres réels, 8 éléments de retard et 10 éléments de mémorisation pour sauvegarder les neuf paramètres d'un filtre de dimension 3×3 et un seul biais.

Il est important de noter que chaque branche du circuit de calcul de la première convolution introduit un retard global au début de calcul et un retard intermédiaire. On outre, chaque branche donne le résultat de convolution sous forme d'un vecteur unique, ce vecteur est reformuler en matrice à l'aide d'un programme de remise en forme que nous avons développé sous l'environnement Matleb.

III.3.2. Circuit de calcul ReLU 1 et Max pooling 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 un élément de mémorisation.

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 ReLU 1. Il est formé de huit branches, 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.

Le schéma global du circuit de calcul de ReLU 1 et de Max pooling 1 est donné dans la Figure III.4.

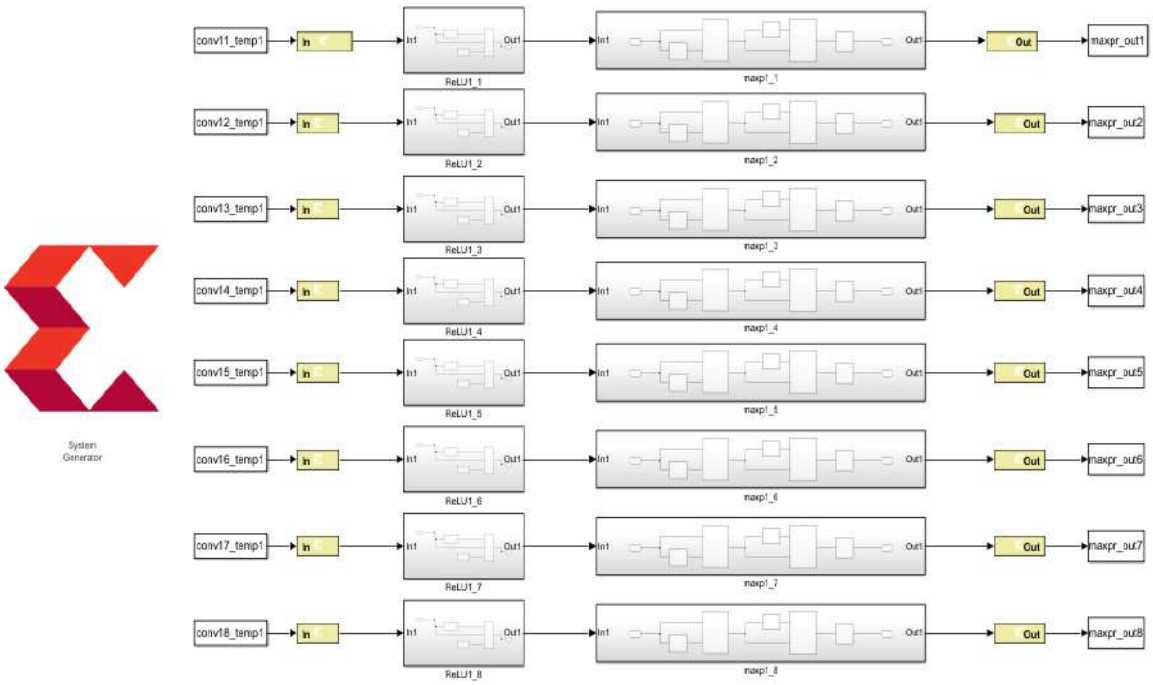


Figure III.4 : Structure des huit branches du circuit de calcul du ReLu 1 et Max pooling 1.

III.3.3. Circuit de calcul de la convolution 2

Le circuit de calcul pour la deuxième convolution est trop compliqué que la première couche de convolution comme le montre la Figure III.5. Le circuit est composé de 16 groupes de calcul identiques dont chaque groupe permet de calculer l'une des 16 maps des caractéristiques de la couche.

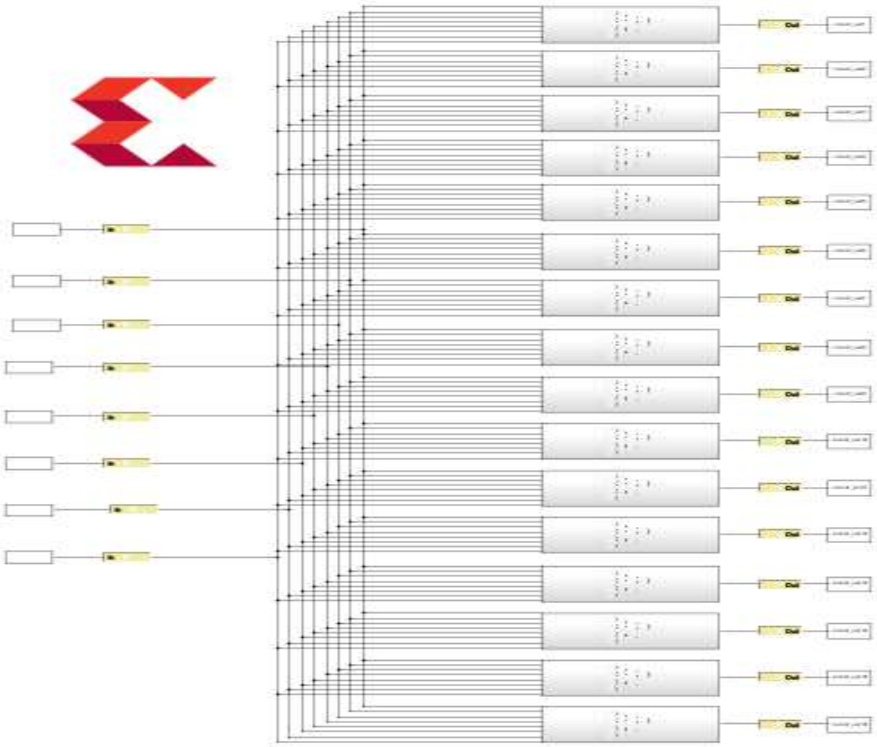


Figure III.5 : Schéma bloc du circuit de calcul de la convolution 2.

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 comme montré dans la Figure III.6.

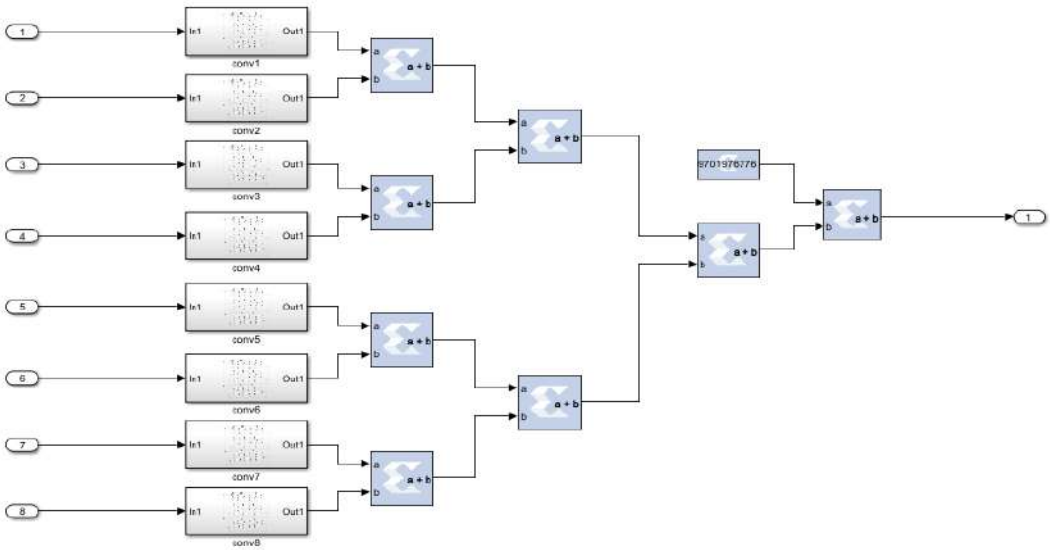


Figure III.6 : La structure détaillée d'un seul groupe du circuit de calcul de la convolution 2.

III.3.4. Les circuits de calcul de ReLU 2 et de max pooling 2

Les circuits sont identiques aux circuits de calcul de ReLu1 et de max pooling 1, sauf qu'ils sont composés de 16 branches parallèles au lieu de 8 branches. 0

III.3.5. Circuit de calcul de la convolution 3

Le circuit de calcul pour la troisième convolution est trop compliqué que la première et la deuxième couche de convolution. Il est composé de 32 groupes de calcul identiques dont chaque groupe contient 16 branches de calcul de convolution notées conv_1 à conv_16. Les sorties des 16 branches sont additionnées à l'aide de 15 additionneurs pour former une convolution unique. Cette dernière est additionnée à un biais pour former une map de caractéristiques.

Le tableau III.1 résume le nombre total d'éléments utilisés par les trois couches de convolution. Il est clair que l'implémentation parallèle de la deuxième couche de convolution et de la troisième couche nécessite un nombre important d'éléments de calcul et de mémorisation.

	Multiplieurs	**Additionneur**	**Eléments de retard**	**Eléments de mémorisation**
Conv 1	72	72	64	80
Conv 2	1152	1152	1024	1168
Conv 3	4608	4608	4096	4640

Tableau III.1 : Nombre total des éléments utilisés par chacun des circuits de calcul des trois couches de convolution.

III.3.6. Le circuit de calcul de ReLU 3

Le circuit de calcul de ReLU 3 est constitué de 32 branches identiques à ceux du circuit de ReLU 1 et ReLU 2.

III.3.7. Le circuit de calcul de la couche complètement connectées

Le circuit de la couche complétement connecté est composé de deux parties comme indiqué sur la Figure III.7. La première partie permet de calculer d'une façon parallèle les 10 scores de classification correspondants aux dix classes de digits. La deuxième partie calcul le score maximale et le numéro de la classe correspondante.

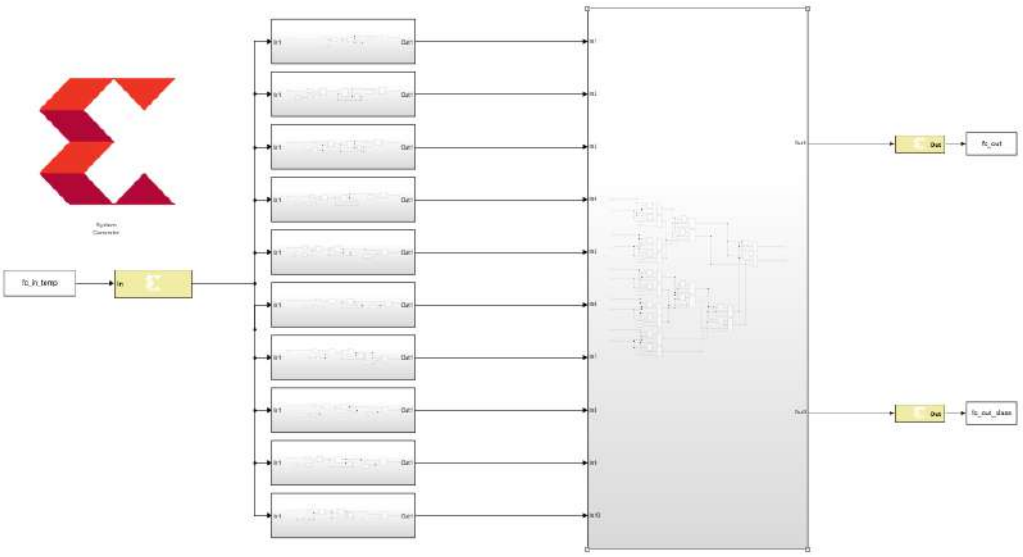


Figure III.7 : La structure du circuit de calcul de la couche complètement connectées.

La Figure III.8 donne le schéma détaillé de l'une des dix branches de la première partie du circuit.

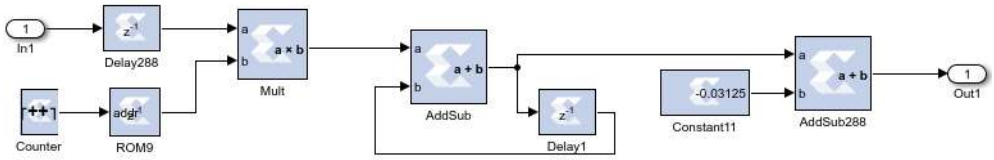


Figure III.8 : schéma détaillé de l'une des dix branches de calcul des scores de classification de la partie entièrement connectée.

La Figure III.9 montre le schéma proposé pour la deuxième partie du circuit de la partie entièrement connectée.

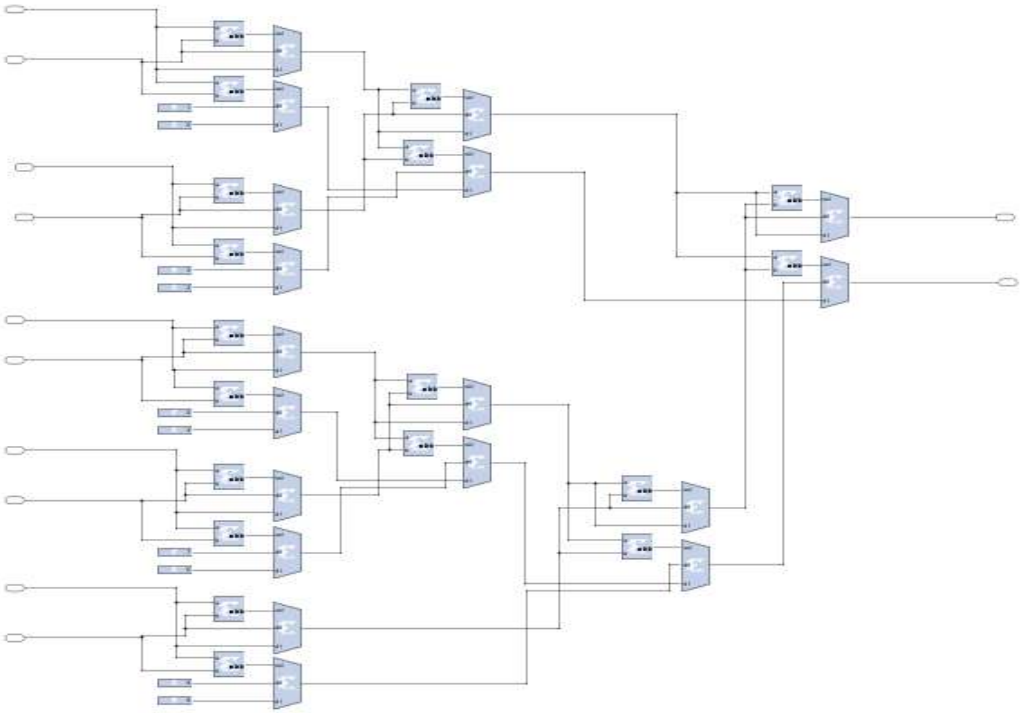


Figure III.9 : Schéma détaillé du circuit de calcul de la classe finale et de son score.

III.4. Optimisation des circuits de calcul

Les conceptions des circuits de calcul du modèle CNN proposées dans l'étape précédente sont meilleures en termes de temps d'exécution, car elles sont basées sur une exécution parallèle. Cependant, elles consomment beaucoup de ressources et sont difficiles ou même impossible de les implémenter sur une seule carte FPGA surtout si on utilise la représentation en virgule flottante des données. Par exemple, si nous prenons la convolution 2, nous constatons que les ressources du circuit FPGA Virtex 7 XC7VX485T-2FFG1761 ne sont pas suffisantes comme le montre le tableau III.2.

Type de ressource	Slice	LUT	Flip-Flop	DSP
Utilisées	75900	303600	30256	3456
Pourcentage	100%	100%	9.96%	123%

Tableau III.2 : ressources du circuit Virtex 7 XC7VX485T-2FFG1761 consommées par la convolution 2 en virgule flottante.

Pour cela, il est indispensable de procéder à l'optimisation des circuits proposés pour pouvoir les implémenter d'une façon efficace sur un circuit FPGA unique. Pour faire l'optimisation nous avons utilisé deux techniques. La première technique est basée sur la réutilisation des ressources alors que la deuxième technique est basée sur l'utilisation de la virgule fixe au lieu de la virgule flottante pour la représentation des données.

III.4.1. Technique de réutilisation de ressources

La technique de réutilisation des ressources est basée sur la conversion de quelques étapes de calcul parallèle en calcul série qui utilise les mêmes ressources. Par exemple, au lieu d'utiliser un circuit qui contient huit branches parallèles pour calculer huit sortie en même temps on utilise une seule branche pour calculer les huit sortie l'une après l'autre. Cette méthode est plus efficace du point de vue consommation de ressources mais elle nécessite plus de temps pour effectuer les calculs.

III.4.1.1. Circuit de calcul série de la convolution 1

Le circuit réalisé permet de calculer d'une façon série les huit maps de caractéristiques de la première couche de convolution. Donc à la sortie, il y a un seul vecteur qui contient les huit maps de caractéristiques. La Figure III.10 montre le circuit du calcul série de la première couche de convolution.

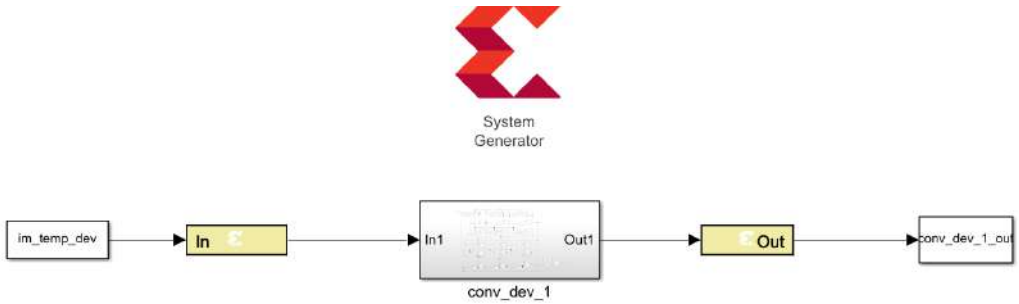


Figure III.10 : Schéma bloc du circuit de calcul série de la convolution 1.

Puisque on utilise le même circuit pour calculer les 8 maps de caractéristiques de sortie et sachant que chaque sortie utilise son propre filtre, il est nécessaire d'envisager une procédure qui permet l'utilisation correcte des filtres. Pour ce faire, nous avons proposé un circuit basé sur l'utilisation de mémoires et de compteur. Les mémoires servent à stocker les valeurs des filtres alors que le compteur sert à générer les adresses d'accès aux mémoires. La Figure III.11 montre le schéma complet du circuit de calcul de la convolution série.

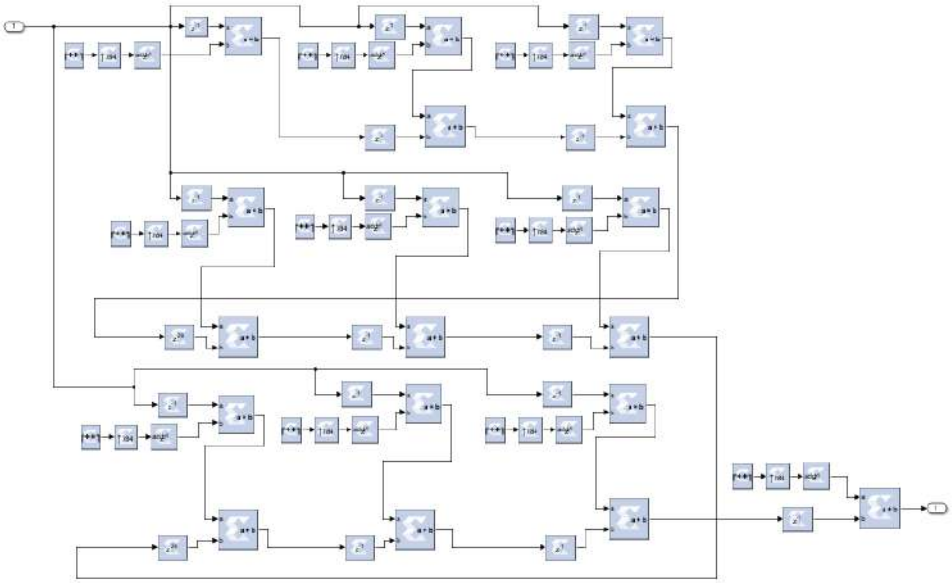


Figure III.11 : Schéma détaillé du circuit de calcul de la convolution série.

III.4.1.2. Circuit de calcul série de ReLU 1 et de Max pooling 1

Le circuit de calcul série de ReLU 1 et de Max pooling 1 ne contient qu'une seule branche au lieu de huit branches pour le circuit de calcul parallèle comme montré dans la Figure III.12.

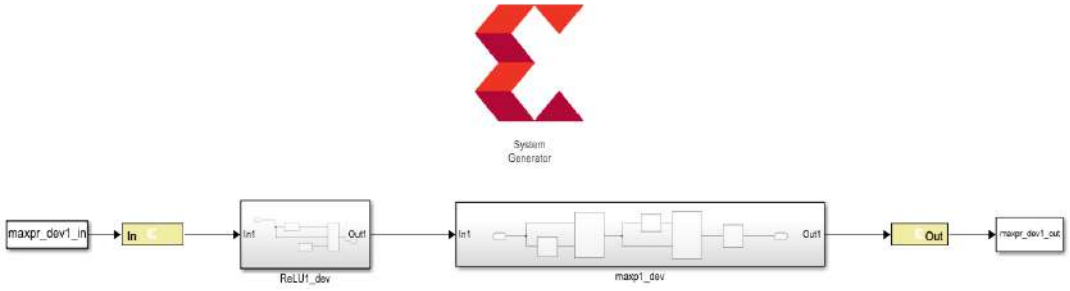


Figure III.12 : Structure du circuit de calcul série de ReLU 1 et Max pooling 1.

III.4.1.3. Circuit de calcul série de la convolution 2

Le circuit est composé de huit parties identiques notées conv_dev_1 à conv_dev_8 dont chaque partie est chargée de calculer une convolution partielle correspondant à un seul canal d'entrée. Les huit parties fonctionnent en parallèle, leurs sorties sont additionnées à l'aide de 7 additionneurs puis additionnées à un biais pour calculer une seule sortie convolution. Le circuit global répète l'opération de calcul pour produire en série les 16 sorties de convolution sur 16 cycles de calcul. Comme pour la première convolution, le circuit de calcul de la deuxième convolution nécessite un circuit adéquat de gestion des paramètres de convolution par l'utilisation des mémoires pour le stockage et d'un compteur pour la génération des adresses.

III.4.1.4. Circuit de calcul série de ReLU 2 et Max pooling 2

Le circuit de calcul de ReLU 2 et de Max pooling 2 est identique à celui utilisé pour ReLU 1 et Max pooling 1 sauf que le premier calcul 16 vecteurs de sortie alors que le deuxième calcul 8 vecteurs.

III.4.1.5. Circuit de calcul série de la convolution 3

Le circuit est composé de 16 parties identiques notées conv3_dev_1 à conv3_dev_16 dont chaque partie est chargée de calculer une convolution partielle. Les sorties des 16 branches sont additionnées à l'aide de 15 additionneurs pour former une seule convolution. Cette dernière est additionnée à un biais pour former une seule map de sortie. Donc, le circuit global nécessite 32 cycles de calcul pour produire en série les 32 maps de sortie.

Le tableau III.3 résume le nombre d'éléments de calcul de mémorisation utilisés par chaque circuit de calcul série correspondants aux trois couches de convolution. La dernière colonne donne le nombre de cases mémoire nécessaire pour mémoriser les valeurs des filtres et des biais de chaque couche. Ce nombre est exprimé sous forme de produit de deux nombres, le premier correspond au nombre mémoires utilisées et le deuxième correspond au nombre de cases de chaque mémoire.

Une comparaison entre le tableau III.1 et III.3 montre que l'utilisation de circuit de calcul série permet de réduire significativement le nombre d'éléments de calcul et de mémorisation en comparaison avec les circuits de calcul parallèle surtout pour les deux convolutions Conv 2 et Conv 3. Cette réduction permet d'implémenter le réseau CNN d'une façon efficace sur un circuit FPGA.

	Multiplieurs	Additionneur	Eléments de retard	Cases mémoire
Conv 1	9	9	18	18×8
Conv 2	72	72	137	73×16
Conv 3	144	144	273	145×32

Tableau III.3 : Nombre total des éléments utilisés par chacun des circuits de calcul série pour les trois couches de convolution.

III.4.1.6. Circuit de calcul série de ReLU 3

Le circuit de calcul série de ReLU 3 est identique circuit de calcul série de ReLU 1, mais le vecteur de sortie contient 32 maps des caractéristiques.

III.4.1.7. Circuit de calcul série de couche complètement connectées

Les circuits sont exactement comme les circuits de calcul de couche complètement connectées de la Figure III.7.

III.4.2. Représentation des données en virgule fixe

Les formats à virgule flottante et à virgule fixe sont deux représentations majeures des nombres réels en informatique. Les structures de ces deux formats pour la même longueur de 32 bits sont présentées à la Figure III.13. Les données sont représentées par leur signification et mises à l'échelle par l'exposant sous la forme d'une puissance de deux. Le terme "flottant" fait référence au fait que la virgule décimale du nombre peut se déplacer par rapport aux chiffres significatifs. Par conséquent, le format à virgule flottante peut représenter une large gamme de nombres.

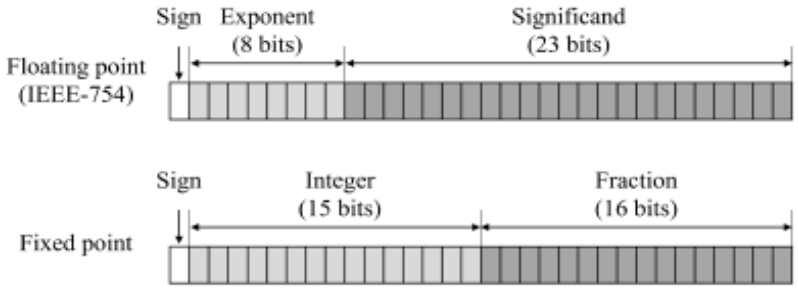


Figure III.13 : Formats de représentation en virgule fixe et en virgule flottante 32 bits.

Le format à virgule fixe se compose de bits entiers (y compris les bits de signe) et de bits fractionnaires. Les données à virgule fixe sont essentiellement des données binaires décalées d'un facteur statique donné. Les positions des bits sont fixes sans décalage supplémentaire des chiffres, contrairement aux données à virgule flottante. Bien que les données à virgule fixe aient une précision limitée et entraînent une perte progressive des données, des recherches récentes ont montré que l'utilisation de données à virgule fixe pour les CNN peut donner

approximativement les mêmes résultats que l'utilisation de données à virgule flottante lorsqu'on leur donne un nombre suffisant de bits. Pour les conceptions FPGA, l'utilisation de données à point fixe peut réduire l'utilisation des ressources, la puissance et la latence. [35]

Pour déterminer le nombre optimal de bits fractionnaires, nous pouvons tester des modèles expérimentaux avec différentes tailles de données. Le nombre de bits entiers dépend de l'architecture du réseau, en particulier de la taille des noyaux de convolution. Nous avons constaté que dans notre architecture CNN, le nombre minimum de bits de la partie entière requis pour empêcher le débordement lors de l'accumulation est de six bits (dont un bit de signe). Pour la partie fractionnaire, et afin de garder une précision acceptable de la classification nous avons utilisé 6 bits comme le montre la Figure III.14.

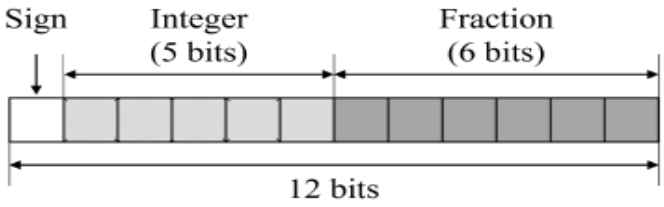


Figure III.14 : Format de représentation en virgule fixe choisi pour notre architecture.

III.5. Résultats d'implémentation sur circuit FPGA

Après optimisation et vérification du bon fonctionnement de chacun des circuits de conception développés, nous avons généré leurs model HDL « Hardware Discription Language ». La génération du HDL se fait automatiquement avec le bloc de System Générateur. Mais, Il est nécessaire de spécifier le type de la carte FPGA ciblé et leurs paramètres, comme montré dans la Figure III.15. Ensuite nous avons passés à l'environnent Vivado pour faire l'implémentation.

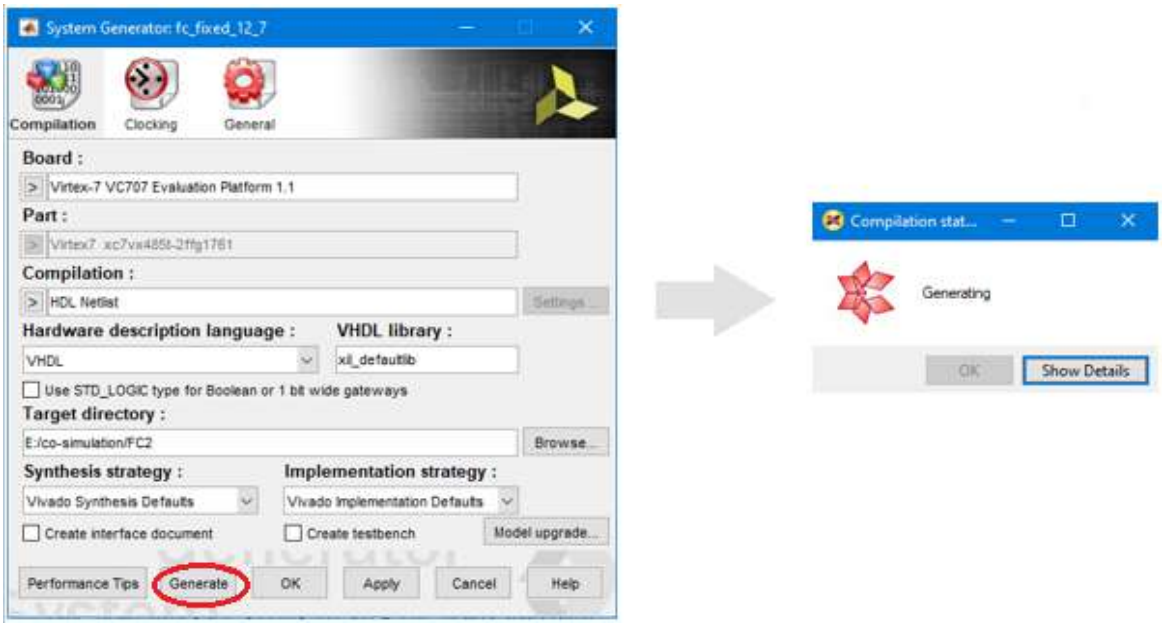


Figure III.15 : Fenêtre System Generator de génération du HDL de l'un des circuits développé.

Après l'implémentation dans l'environnent Vivado des circuits, nous avons obtenu tous les schémas des circuits synthétisés sur le circuit FPGA cible de tout le réseau CNN. Par exemple, la Figure III.16 montre le circuit de la convolution 1 obtenu après la fin de l'étape d'implémentation sur le circuit FPGA Virtex 7 XC7VX485T-2FFG1761.

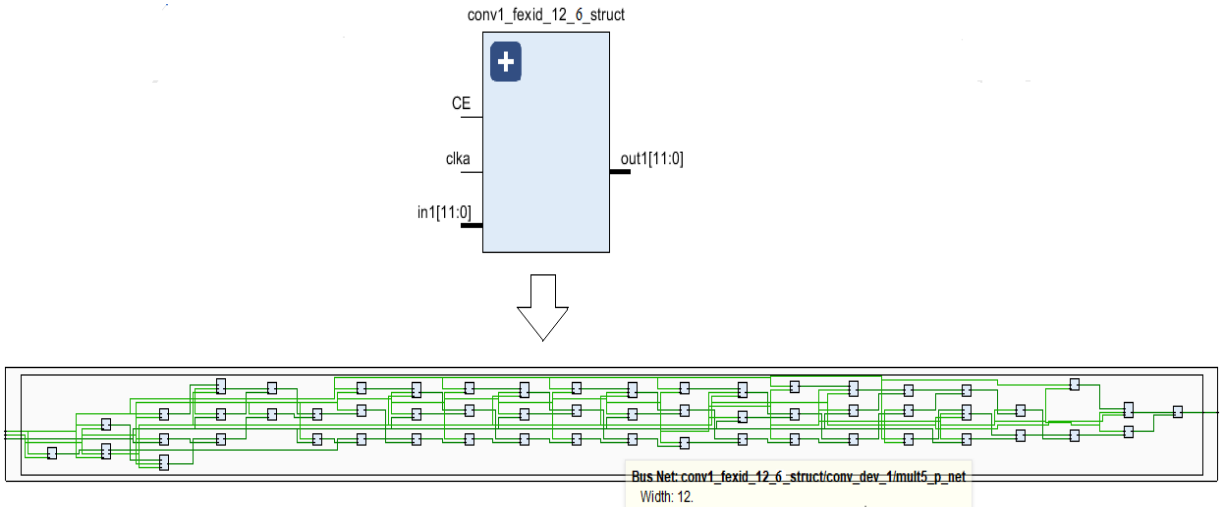


Figure III.16 : Le schéma du circuit de convolution 1 sur le circuit Virtex 7 XC7VX485T-2FFG1761.

Le logiciel Vivado donne aussi l'estimation de la consommation de puissance de chaque circuit. Par exemple, la Figure III.17 détaille la consommation de la convolution 1.

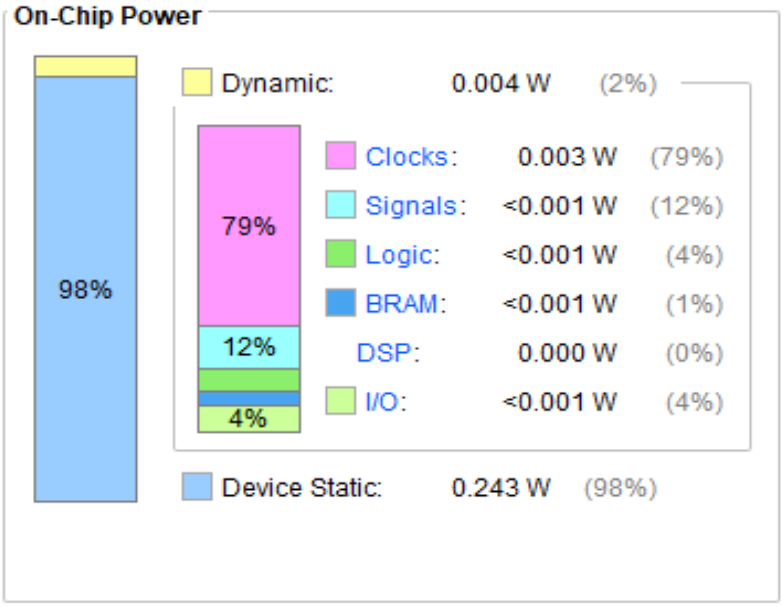


Figure III.17 : Estimé de la puissance consommée par le circuit de la convolution 1 l'implémenté sur Virtex 7 XC7VX485T-2FFG1761.

Nous avons fait l'implémentation de tous les circuits de calcul pour la totalité du réseau CNN. Le tableau III.4 donne les ressources FPGA et la puissance dynamique consommée par chacun des circuits de l'architecture du réseau CNN proposé.

			Ressources						
			Slice	**LUT**	**Flip-Flop**	**DSP**	**IOB**	**BRAM**	**Energie dynamique (Watt)**
convolution 1	**virgule flottante**	**Nombre Utilisée**	1217	4227	687	27	129	5	0.070 (22%)
		Pourcentages (%)	1.60	1.39	0.11	0.96	18.43	0.49	
	virgule fixe	**Nombre Utilisée**	76	161	263	9	25	5	0.004 (2 %)
		Pourcentages (%)	0.10	0.05	0.04	0.32	3.57	0.49	
ReLU 1 et Maxpoling 1	**virgule flottante**	**Nombre Utilisée**	69	198	132	0	/	0	0.013 (5%)
		Pourcentages (%)	0.09	0.07	0.02	0	/	0	
	virgule fixe	**Nombre Utilisée**	26	51	55	0	/	0	0.005 (2%)
		Pourcentages (%)	0.03	0.02	0.01	0	/	0	
convolution 2	**virgule flottante**	**Nombre Utilisée**	9599	33187	4691	216	/	36.50	0.448 (64%)
		Pourcentages (%)	12.65	10.93	0.77	7.71	/	3.54	
	virgule fixe	**Nombre Utilisée**	535	1076	1951	72	/	36.50	0.018 (7%)
		Pourcentages (%)	0.70	0.35	0.32	2.57	/	3.54	
ReLU 2 et Maxpoling 2	**virgule flottante**	**Nombre Utilisée**	65	183	132	0	/	0	0.013 (5%)
		Pourcentages (%)	0.09	0.06	0.02	0	/	0	
	virgule fixe	**Nombre Utilisée**	23	43	52	0	/	0	0.005 (2%)
		Pourcentages (%)	0.03	0.01	0.01	0	/	0	
convolution 3	**virgule flottante**	**Nombre Utilisée**	19139	66331	9473	432	/	72.50	1.061 (81%)
		Pourcentages (%)	25.22	21.85	1.56	15.43	/	7.04	
	virgule fixe	**Nombre Utilisée**	1211	2133	4013	144	/	72.50	0.260 (51%)
		Pourcentages (%)	1.60	0.70	0.66	5.14	/	7.04	
ReLU 3	**virgule flottante**	**Nombre Utilisée**	4	16	0	0	/	0	0.009 (4%)
		Pourcentages (%)	0.01	0.01	0	0	/	0	
	virgule fixe	**Nombre Utilisée**	2	6	0	0	/	0	0. 004 (2%)
		Pourcentages (%)	0.01	0.01	0	0	/	0	
couche complètement connectées	**virgule flottante**	**Nombre Utilisée**	2435	8504	188	30	97	5	0.342 (58%)
		Pourcentages (%)	3.21	2.80	0.06	1.07	13.86	0.49	
	virgule fixe	**Nombre Utilisée**	174	371	130	10	37	5	0.064 (21%)
		Pourcentages (%)	0.23	0.12	0.04	0.39	5.29	0.49	

Tableau III.4 : Les ressources et la puissance dynamique consommées par les différents circuits de calcul implémenté sur Virtex 7 XC7VX485T-2FFG1761.

A partir du tableau on peut tirer les conclusions suivantes :

- L'utilisation des circuits de calcul série, que ce soit en virgule fixe ou en virgule flottant, réduit l'utilisation des ressources hardware de la carte FPGA par rapport aux circuits de calcul parallèle.
- Les couches de convolution nécessites plus de ressources FPGA et plus puissance par rapport au couches ReLU, Max pooling et FC.
- La troisième couche de convolution est la partie qui consomme le maximum de ressources et de puissance par rapport à toutes les couches du réseau CNN.
- Les ressources disponibles sur le circuit FPGA Virtex 7 XC7VX485T-2FFG1761 sont largement suffisantes pour implémenter la totalité du réseau en mode de calcul série ce qui est impossible si on utilise le mode complètement parallèle.
- L'utilisation du format de représentation en virgule fixe améliore l'efficacité d'implémentation d'une façon très significative en comparaison avec le format en virgule flottante. En effet, et comme la montre la Figure III.18, l'implémentation en virgule réduit d'une façon très significative le nombre de ressources consommées surtout pour les Slices et les LUTs.

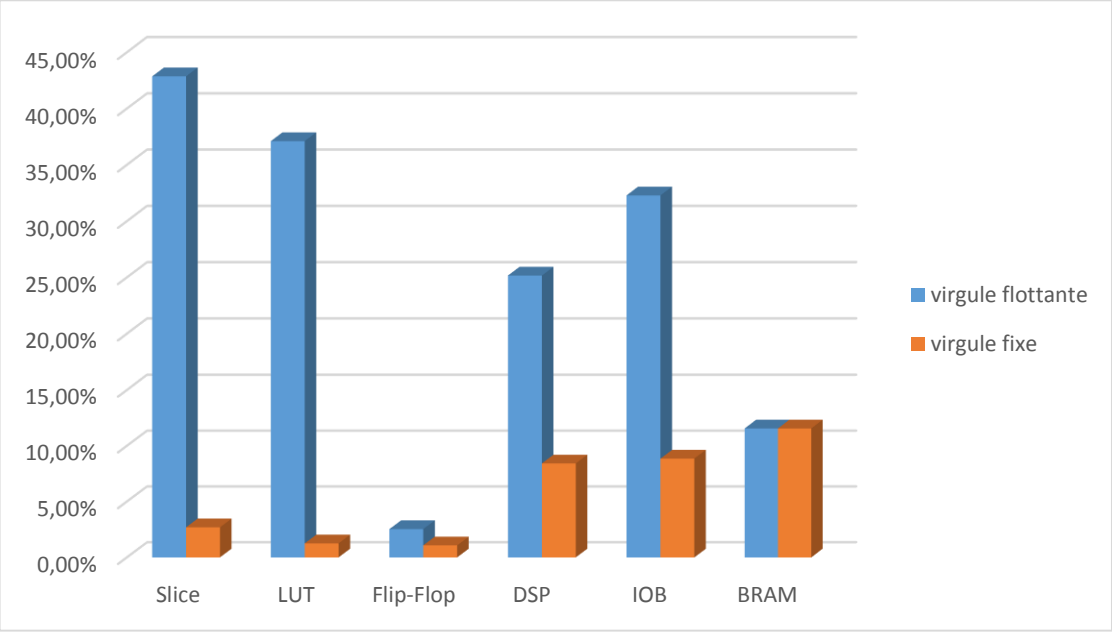


Figure III.18 : La consommation des ressources du réseau implémenté sur Virtex 7 XC7VX485T-2FFG1761.

III.6. Vérification des résultats par Co-simulation sur la carte FPGA

Après vérification des résultats par simulation, nous avons utilisé la technique de Co-simulation matériel et logiciel (Hardware Software Co-simulation) afin de vérifier les résultats de calcul en utilisant la carte d'évaluation VC707 qui est conçu autour d'un circuit FPGA de type Virtex 7 XC7VX485T-2FFG1761. La Co-simulation permet de faire l'implémentation sur une plateforme FPGA d'une architecture qui est déjà conçu sous System Generator. La Co-simulation commence par l'étape de génération du bitstream et de l'interface de communication entre la machine hôte comme illustré dans la Figure III.19. .

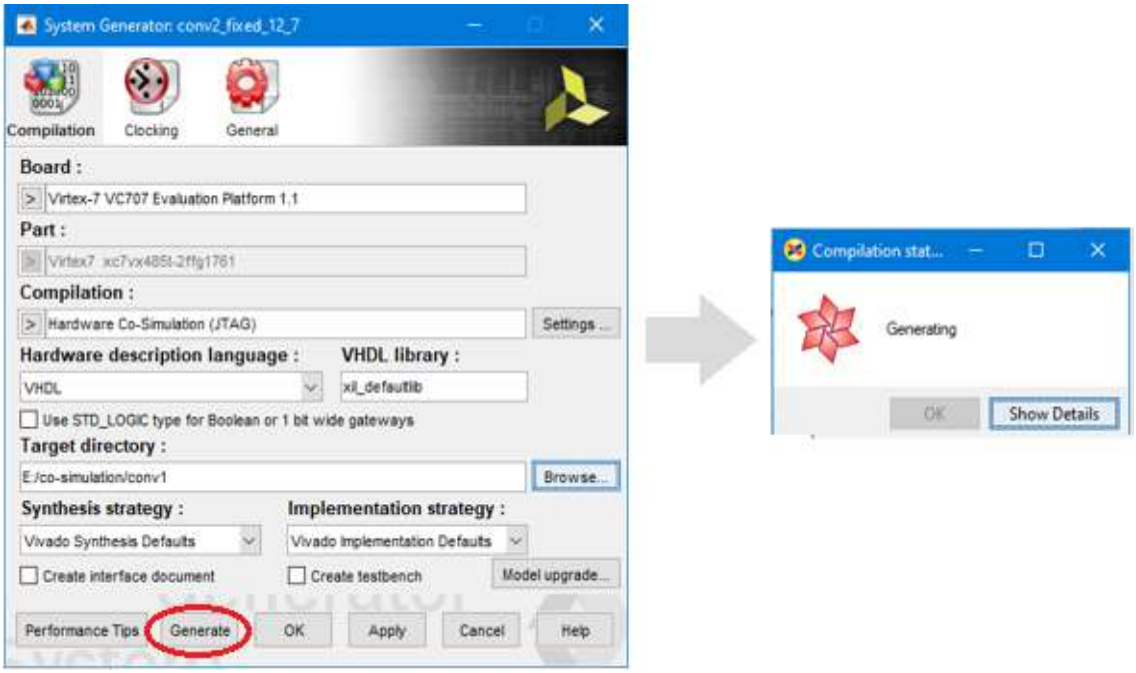


Figure III.19 : Génération du modèle de gestion de la Co-Simulation.

Après génération du modèle de gestion de Co-simulation on doit lui brancher les signaux d'entrée et le moyen de visualisation de résultat. La Figure III.20 donne un exemple de modèle de gestion de la Co-simulation pour la convolution 1.

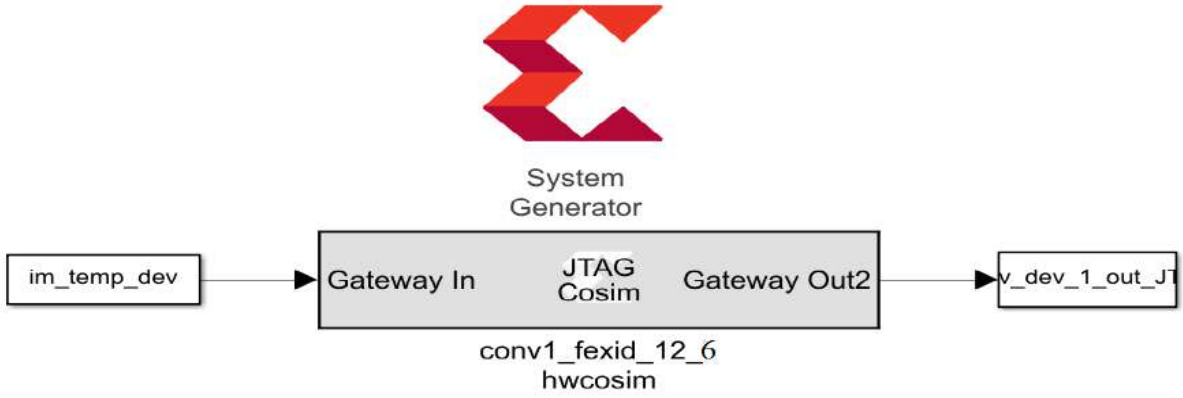


Figure III.20 : Schéma du modèle de gestion de la Co-simulation du circuit de convolution 1.

Pour la vérification pratique des résultats de calcul sur FPGA, il faut d'abord relier la carte FPGA à l'entrée USB du PC à travers le bus JTAG de la carte que la montre la Figure III.21. Une fois la carte est détectée, on lance la Co-simulation. Le bitstream est chargé dans le circuit FPGA cible, les entrées sont communiquées à la carte qui effectue les calculs et renvoi les résultats vers l'espace de travail de Matlab. Les résultats sont utilisés pour vérifier le bon fonctionnement du réseau sur la carte FPGA.



Figure III.21 : Image de la carte VC707 en exécution de la Co-simulation.

III.7. Conclusion

Ce chapitre a été consacré à la présentation des différentes étapes de réalisation de notre projet qui consiste à faire une implémentation efficace de CNN sur circuit FPGA. Pour cela, nous avons commencé par la sélection d'une architecture simple du réseau CNN pour l'application de connaissance automatique des digits manuscrits. Ensuite, nous avons présenté les conceptions des circuits de calcul des différentes couches du réseau CNN en utilisant le système générateur, Matlab 2018 et Simulink. Après ça, nous avons optimisé les conceptions des circuits par deux techniques qui sont la réutilisation de ressources et l'utilisation de la virgule fixe pour la représentation des données. Le but de cette étape est de réduire l'utilisation des ressources le plus possible dans la carte FPGA. Ensuite, nous avons passé à l'étape d'implémentations des circuits de calcul en utilisant le logiciel Vivado 2018. Les résultats d'implémentation montrent un gain très important en ressources et en puissance consommée lorsqu'en utilise le calcul série au lieu du calcul parallèle et la représentation en virgule fixe à la place de la représentation en virgule flottante. À la fin, nous avons fait une vérification sur matériel en utilisant la technique de Co-simulation sur une carte FPGA VC 707 qui contient un circuit FPGA de type Virtex 7 XC7VX485T-2FFG1761.

Conclusion générale

Au cours de la dernière décennie, les progrès dans le domaine de CNN ont conduit à des réseaux plus précis pour la classification des images et la détection des objets dans les images et les séquences vidéo. Cela s'est fait au prix d'une augmentation du nombre de paramètres et de la complexité de calcul. Pour effectuer une inférence efficace du modèle CNN, la plate-forme sur laquelle le modèle est exécuté doit disposer de ressources de calcul et de mémorisation suffisantes. Pour cette raison, les GPU sont la plate-forme dominante pour implémentée CNN. Cependant, les GPU ne sont pas adaptés aux applications embarquées et mobiles en raison de leur forte consommation d'énergie. Une meilleure solution serait d'implémenter les réseaux CNN sur des plateformes reconfigurables de type FPGA qui offre une meilleure efficacité énergétique et les meilleures performances.

Dans ce travail nous avons réalisé implémentation d'un réseau de Neurone Convolutif CNN sur circuit FPGA d'une manière efficace.

Premièrement, Nous avons essayé de faire une étude théorique sur les trois parties principale en relation directe de projet qui sons Deep Learning, réseaux de neurones profonds (CNN) et Les circuits logiques programmables FPGA. Deuxièmement, nous avons fait la conception, la simulation, l'optimisation et l'implémentation des différentes couches d'un réseau CNN destiné à l'application de reconnaissance automatique de digits manuscrits.

Nous avons ciblé deux méthodes d'optimisation pour résoudre le problème des ressources limitées, ces méthodes sont la réutilisation de ressources et l'utilisation de la représentation des données en virgule fixe où nous avons utilisés 12 bit (6 bit pour la partie entière et 6 bit pour la partie fractionnelle) dans tous les couches du réseau CNN ciblé.

Les résultats obtenus sont très satisfaisants, car nous avons réussi à réduire l'utilisation des ressources d'une façon très significative. Les résultats d'implémentation montre que le réseau ne consomme que 2.7% de slice, 1.26% de LUT, 1.08% de flip-flop, 8.42% de DSP, 9% des IOB et 11.59 de BRAM du circuit FPGA Virtex 7 XC7VX485T-2FFG1761. Les circuits proposés permettent aussi de réduire la puissance dynamique consommée par le circuit.

La réalisation de ce projet nous a permis d'acquérir des connaissances très précieuses et un savoir-faire pratique sur le domaine des réseaux CNN et sur le domaine de l'implémentation sur FPGA.

Plusieurs améliorations peuvent être considérées comme perspectives de notre travail, entre autres on peut citer :

- Proposition d'une architecture d'implémentation sur FPGA de la partie de mise en forme des données.
- Implémentation du réseau sur une plateforme hybride Zynq contenant des processeurs ARM et un circuit FPGA.
- Implémentation de réseaux CNN plus profond sur circuit FPGA et Zynq.

Bibliographie

1. Abd El-Maksoud, A.J., et al. FPGA Design of High-Speed Convolutional Neural Network Hardware Accelerator, in 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2021.
2. Kelleher, J.D., Deep learning, MIT press, 2019.
3. Abdelouahab, K., et al., Accelerating CNN inference on FPGAs: A survey. arXiv preprint arXiv:1806.01683, 2018.
4. Deng, L. and D. Yu, Deep Learning: Methods And Applications, Foundations and Trends in Signal Processing 2014.
5. Ketkar, N. and J. Moolayil, Deep learning with Python: learn best practices of deep learning models with PyTorch, Springer, 2021.
6. Fernandez Brillet, L., Réseaux de neurones CNN pour la vision embarquée, thèse de doctorat, Université Grenoble Alpes, 2020.
7. Mokri M. Z., Classification des images avec les réseaux de neurones convolutionnels, , mémoire de Master, université de Tlemcen, 2017.
8. Etienne, C., Apprentissage profond appliqué à la reconnaissance des émotions dans la voix, thèse de doctorat, Université Paris-Saclay, 2019.
9. Salim, T., Algorithme Hybride de Débruitage Image/Vidéo à base de Réseaux de Neurones Convolutionnels (CNN), mémoire de Master, université Borj Bou Erréridj, 2021.
10. Bouhedadja, M.A., Utilisation de l'apprentissage profond pour la classification des mauvaises herbes dans le bleuet nain, Mémoire de maitrise, Université du Québec à Trois-Rivières, 2021.
11. Wang, Y., The Impact of Single Event Effect Reliability of Convolution Neural Network Architectures and Hardening Approaches Implemented on SRAM FPGA, mémoire de Master, University of Saskatchewan, 2021.
12. Yamashita, R., et al., Convolutional neural networks: an overview and application in radiology, Insights into imaging, **9**(4): p. 611-629, 2018.
13. Gourmet, V., Real-time hand pose estimation on a smart-phone using Deep Learning, Digitala Vetenskapliga Arkivet, 2019.
14. Alzubaidi, L., et al., Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of big Data, **8**(1): p. 1-74, 2021.

15. YAHIAOUI, A., Système de Discrimination Visages/Faux Visages par Réseaux de Neurone Convolutif (CNN), mémoire de Master université de Guelma, 2021.
16. Gheysen, G., J.-D. Legat, and C. De Vleeschouwer, Accelerating Convolutional Neural Networks for FPGA using depthwise separable convolution and pruning, mémoire de Master, Ecole polytechnique de Louvain, 2020.
17. Ben Lazreg, M., Recherche de l'information dans les réseaux de Neurone Convolutif pré-entraînés,mémoire de mairise, École de technologie supérieure, Québec 2020.
18. Khan, A., et al., A survey of the recent architectures of deep convolutional neural networks, Artificial intelligence review, **53**(8): p. 5455-5516, 2020.
19. Zhang, Z., H. Rong, and Y. Wang, Introduction Of Special Issue on FPGA-Based Computing [From The Guest Editors]. IEEE Circuits and Systems Magazine, **21**(2) p. 3-3, 2021.
20. Churiwala, S. and I. Hyderabad, Designing with Xilinx® FPGAs, in Circuits &Systems, Springer, 2017.
21. Bourrasset, C.d., Le modèle flot de données appliqué à la synthèse haut-niveau pour le traitement d'images sur caméra intelligente à base de FPGA. Application aux systèmes d'apprentissage supervisés, thèse de doctorat, Université Blaise Pascal-Clermont-Ferrand II, 2016.
22. BOUKHENOUS, M.S., Implémentation sur FPGA d'un système intelligent pour la surveillance de l'état de santé des patients souffrants d'arythmie cardiaque, thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediène, 2021.
23. Stratoudakis, T., Introduction to LabVIEW FPGA for RF, Radar, and Electronic Warfare Applications, Artech House, 2021.
24. Liu, S., et al., Robotic computing on fpgas, Synthesis Lectures on Computer Architecture, **16**(1): p. 1-218, 2021.
25. Hajji, B., A. Mellit, and L. Bouselham, A Practical Guide for Simulation and FPGA Implementation of Digital Design, Springer.
26. Saptono, D., Conception d'un outil de prototypage rapide sur le FPGA pour des applications de traitement d'images, thèse de doctorat, universite de Bourgogne 2011.
27. Dubey, R., Introduction to embedded system design using field programmable gate arrays, Springer Science & Business Media, 2008.
28. Свид, I., et al., Review of Seventh Series FPGA Xilinx, NURE, MC&FPGA, 2019.
29. Ezzeddine, M., Modélisation et réalisation de la couche physique du système de communication numérique sans fil, WiMax, sur du matériel reconfigurable, mémoire de maitrise, université de Montréal 2009.

30. Bahoura, M. and H. Ezzaidi. FPGA-implementation of a sequential adaptive noise canceller using Xilinx system generator. in 2009 International Conference on Microelectronics-ICM. 2009. IEEE.
31. Lacey, G., G.W. Taylor, and S. Areibi, Deep learning on fpgas: Past, present, and future. arXiv preprint arXiv:1602.04283, 2016.
32. Cloutier, J., et al. Vip: An fpga-based processor for image processing and neural networks. in Proceedings of fifth international conference on microelectronics for neural networks,IEEE, 1996.
33. Espinosa, M., Implementation of Convolutional Neural Networks in FPGA for Image Classification, Mémoire de Master, Californian State University, 2019.
34. Shahshahani, M., P. Goswami, and D. Bhatia, Memory optimization techniques for fpga based CNN implementations. in 2018 IEEE 13th Dallas Circuits and Systems Conference (DCAS), 2018.
35. Cho, M. and Y. Kim, FPGA-Based Convolutional Neural Network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit, Electronics,. **10**(22): p. 2859, 2021.