République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université Yahia Fares de Médéa
Faculté de Technologie



Thèse

Présentée pour l'obtention du **diplôme** de **DOCTORAT** en Science

En : Electronique

Spécialité : Electronique des Systèmes Embarqués

Par : FEKIR Mohamed

Sujet

IMPLÉMENTATION EFFICACE DE LA TECHNIQUE DES RÉSEAUX DE NEURONE CONVOLUTIFS SUR UN CIRCUIT FPGA.

Soutenue publiquement, le 12/Juin/2024, devant le jury composé de :

Dr.	BENGHARBIA Bilel	U. de Médéa	Président
Pr.	BENZAOUI Amir	U. de Skikda	Examinateur
Dr.	HAMZAOUI Ihssen	U. de Khemis Miliana	Examinateur
Pr.	MOUALDIA Abdelhafidh	U. de Médéa	Directeur de thèse
Dr.	DALI Mohamed	U. de Médéa	Invité

Cette thèse est dédiée à la mémoire de mon père, dont la présence me manque chaque jour. Son amour inconditionnel et son soutien constant ont constitué une source d'inspiration perpétuelle tout au long de ma vie. Je suis profondément reconnaissant d'avoir eu la chance d'avoir un père aussi aimant et attentionné, et c'est avec une grande affection et gratitude que je lui rends hommage à travers cette thèse. Son influence positive demeure présente dans mon travail et dans chaque étape de mon parcours académique. Qu'**Allah** l'accueille dans son vaste paradis.

Cette thèse est également un hommage à ma mère, qui a été une source inépuisable d'amour et de soutien dans ma vie. J'exprime ma gratitude pour ses enseignements, ses sacrifices et son amour inconditionnel. Mes vœux sincères pour qu'**Allah** lui accorde une bonne santé, la sérénité et le bonheur tout au long de sa vie.

À ma merveilleuse épouse, mon pilier de force et mon soutien constant. Tu es le soleil qui illumine ma vie, et chaque instant à tes côtés est un trésor précieux.

À mes précieux enfants, qui apportent joie et bonheur dans ma vie à chaque instant. Vous êtes la lumière de mon existence, et je suis béni de vous avoir comme mes plus grands trésors.

À mes sœurs et mon frère, des compagnons de vie inestimables, avec qui j'ai partagé des souvenirs inoubliables, des rires, et des moments de complicité. Votre présence a enrichi ma vie de manière indescriptible.

À mes chers amis et frères, Salim et Mahfoudh, pour leur soutien inébranlable tout au long de ce parcours.

Remerciements

Je tiens à exprimer ma profonde gratitude au Professeur **MOUALDIA Abdelhafidh**, directeur de cette thèse, pour son soutien continu, ses conseils éclairés et son engagement inébranlable tout au long de ce travail de recherche. Son expertise et sa bienveillance ont été des piliers essentiels dans la réalisation de ce projet.

Je souhaite exprimer mes sincères remerciements à **BENGHARBIA Bilel**, maître de conférences à l'Université de Médéa et président du jury, ainsi qu'à **HAMZAOUI Ihssen**, maître de conférences à l'Université de Khemis Miliana, et **BENZAOUI Amir**, Professeur à l'Université de Skikda, pour leur contribution en tant qu'examinateurs de cette thèse. Je suis reconnaissant de l'honneur qu'ils ont accordé à ce travail en acceptant de jouer un rôle crucial dans son évaluation. Leurs commentaires éclairés et leur expertise ont grandement enrichi cette recherche, contribuant ainsi à son excellence.

Je tiens à renouveler mes remerciements et ma gratitude envers Monsieur **BENGHARBIA Bilel** pour le soutien qu'il m'a apporté en mettant à ma disposition les logiciels nécessaires au développement. Leur utilité a été inestimable durant la phase d'implémentation.

Je voudrais également exprimer ma sincère gratitude envers Monsieur **DALI Mohamed** pour son assistance précieuse et son soutien prodigués au début de ce travail de thèse. Sa contribution a joué un rôle significatif dans l'orientation initiale de cette recherche et a grandement enrichi sa qualité et sa portée.

Je réaffirme mes remerciements et ma gratitude à toutes les personnes qui ont contribué de près ou de loin à l'élaboration de ce travail. Ce projet est le fruit de leur collaboration et de leur soutien précieux.

ملخص

تتناول هذه الأطروحة التنفيذ الفعال للشبكات العصبية التلافيفية CNNsعلى مصفوفات البوابات المنطقية القابلة للبرمجة FPGAs. في مجال التكنولوجيا الذي يتقدم بسرعة، أدى ظهور تطبيقات الذكاء الاصطناعي والتعلم العميق إلى زيادة الاهتمام بالحلول المادية المتخصصة. تهدف أبحاثنا إلى استكشاف وتحسين تنفيذ استنتاجات الشبكات العصبية التلافيفية على مصفوفات البوابات المنطقية القابلة للبرمجة FPGAs.

لتحقيق هذا الهدف، قمنا بتحقيق نهجين رئيسيين:

- **التجميع عالي المستوى** (HLS): باستخدام أداة Xilinx Vivado، قمنا بتحويل الكود عالي المستوى (C/C++)إلى كود مادي. هذه العملية تسهل التنفيذ الأسرع والأكثر كفاءة على مصفوفات البوابات المنطقية القابلة للبرمجة FPGAs. يمكّن (HLS)من وصف العمليات من حيث السلوك والخوارزميات، ثم تحويلها إلى منطق مادي. بالإضافة إلى ذلك، يوفر توجيهات تحسين معمارية لتحسين أداء التصاميم المادية.
- Verilog : كما قمنا بفحص استخدام لغة وصف الأجهزة Verilog للحصول على تحكم دقيق في بنية FPGA. هذه الطريقة تسمح بتطوير تصاميم مخصصة، مما يكون مفيداً بشكل خاص للمشاريع التي تتطلب تحكماًا دقيقاًا في FPGA.

في النهج الأول، قمنا بتنفيذ LeNet5 CNN بشكل شامل، مغطين كل من مرحلة التدريب ومرحلة التنبؤ. لقد استخدمنا بيئات التطوير Python و Vivado HLS ، واستفدنا من مجموعة البيانات MNIST للتدريب ومجموعة تطوير Vertex7 للمحاكاة.

حقق نموذج LeNet5، المدرب على مجموعة بيانات MNIST، دقة بلغت ٧٩٪. تم دمج المعلمات والتحيزات التي تم الحصول عليها خلال التدريب في مرحلة الاستنتاج باستخدام Vivado HLS. كشفت تقييمات الأداء للتصاميم المختلفة عن زمن انتقال قدره ٢٩٩.٣ ميكروثانية. تجمع أبحاثنا بين التحسينات مثل خط أنابيب الوظائف، التوازي الحلقات، وتقسيم المصفوفات، باستخدام كل من HLS و Verilog لتحقيق تنفيذ فعال للشبكات العصبية التلافيفية على FPGAs.
وأخيراً، قارنا نتائجنا بالنهج الأخرى في أحدث التقنيات، مما يؤكد فعالية وملاءمة منهجيتنا في مجال تنفيذ CNN على FPGAs .

الكلمات المفتاحية: الشبكات العصبية التلافيفية (CNN)، مصفوفة البوابة الميدانية القابلة للبرمجة (FPGA)، تنفيذ الأجهزة، التحسينة

Résumé

Cette thèse se focalise sur l'optimisation de l'implémentation des réseaux de neurones convolutifs (CNN) sur les FPGA (Field-Programmable Gate Arrays). Avec l'avènement rapide des applications basées sur l'intelligence artificielle et le deep learning, il y a un intérêt croissant pour les solutions matérielles spécialisées. L'objectif de notre recherche est d'explorer et d'améliorer l'efficacité de l'implémentation de l'inférence des CNN sur les FPGA.

Pour ce faire, Nous avons exploré deux approches essentielles :

— **High-Level Synthesis (HLS) :** Utilisant l'outil Xilinx Vivado, nous avons traduit du code de haut niveau (C/C++) en code matériel, facilitant ainsi une exécution plus rapide et efficace sur FPGA. Le HLS permet de décrire des opérations en termes de comportement et d'algorithmes, pour ensuite les convertir en logique matérielle, et propose des directives d'optimisation architecturale pour améliorer la performance des conceptions matérielles.

— **Verilog :** Nous avons également exploré le langage de description matérielle Verilog pour obtenir un contrôle précis sur l'architecture FPGA. Cette approche favorise la création de conceptions personnalisées, particulièrement avantageuses pour les projets nécessitant un contrôle détaillé sur le FPGA.

Dans notre première approche, nous avons complètement implémenté le modèle CNN LeNet5, couvrant à la fois l'apprentissage et l'inférence. À l'aide de Python et de Vivado HLS, nous avons formé le modèle sur les données MNIST et effectué des simulations sur le kit de développement Vertex7. Le modèle LeNet5, formé sur MNIST, a atteint une précision de 97%. Les paramètres et les biais appris ont été intégrés dans la phase d'inférence en utilisant Vivado HLS, avec des évaluations de performance montrant une latence de 299,3 s. Notre recherche combine des optimisations comme le pipelining des fonctions, le parallélisme des boucles et le partitionnement des matrices, en utilisant à la fois HLS et Verilog pour une implémentation efficace des CNN sur FPGA. En conclusion, nous avons comparé nos résultats avec d'autres approches utilisant les dernières technologies, confirmant ainsi l'efficacité et la pertinence de notre méthodologie dans le domaine de l'implémentation des CNN sur les FPGA.

Mots-Clés : Convolutional neural networks (CNNs), Field Programmable Gate Array (FPGA)), hardware implé mentation, optimisation.

Abstract

This thesis focuses on the efficient implementation of Convolutional Neural Networks (CNNs) on Field-Programmable Gate Arrays (FPGAs). In a constantly evolving technological landscape, the emergence of AI-based and deep learning applications has sparked growing interest in dedicated hardware solutions. Through our research endeavors, we aim to explore and optimize the implementation of CNN inference on FPGAs.

To achieve this goal, To achieve this, we explored two essential approaches:

1. **High-Level Synthesis (HLS) :** Utilizing the Xilinx Vivado tool, we converted high-level code (C/C++) into hardware code. This process facilitates faster and more efficient execution on FPGAs. HLS enables the description of operations in terms of behavior and algorithms, which are then translated into hardware logic. Additionally, it provides architectural optimization directives to enhance hardware design performance.
2. **Verilog :** We also examined the use of the hardware description language Verilog to gain precise control over the FPGA architecture. This method allows for the development of customized designs, which is particularly beneficial for projects that require detailed control over the FPGA.

In our initial approach, we fully implemented the CNN model LeNet5, addressing both training and inference phases. Using Python and Vivado HLS, we trained the model on the MNIST dataset and conducted simulations on the Vertex7 development kit.
The LeNet5 model achieved 97% accuracy on the MNIST dataset. Parameters and biases from training were integrated into the inference phase using Vivado HLS. Performance evaluations of various designs revealed a latency of 299.3 s. Our research combines optimizations such as function pipelining, loop parallelism, and matrix partitioning, utilizing HLS and Verilog for efficient CNN implementation on FPGAs.
Finally, we compared our results with other approaches employing the latest technologies, confirming the effectiveness and suitability of our methodology in the field of CNN implementation on FPGAs.

Keywords: Convolutional neural networks (CNNs), Field Programmable Gate Array (FPGA)), hardware implementation, optimization.

Table des matières

Table des figures **i**

Liste des tableaux **v**

Liste des Algorithmes **vi**

Nomenclature **vii**

1 Introduction **1**

1.1 Contexte de recherche 1
1.2 Problématique 2
1.3 Motivation 4
1.4 Contributions 5
1.5 Données et outils d'implémentation 6
1.6 Organisation du manuscrit 8

2 Réseaux de Neurones Convolutifs (CNNs) **10**

2.1 Introduction 10
2.2 Contexte historique des algorithmes AI 10
2.3 Réseaux de neurones artificiels 12
2.4 Réseaux de neurones convolutifs 12
2.5 Modèles CNNs 13
2.6 Analyse en Profondeur des Couches Constitutives des CNNs 18
2.6.1 Couche de convolution 18
2.6.2 Fonctions d'activation 19
2.6.3 couche *"Poolimg"* ou mise en commun 22
2.6.4 Couche entièrement connectée *"Fully connected"* 22

2.7 Opérations de la couche de convolution 23
2.8 Conclusion 30

3 Revue de la Littérature scientifique et Travaux Connexes 31
3.1 Introduction 31
3.2 Domaines d'application des CNNs 32
3.3 Revue des travaux relatifs à l'implémentation et à l'accélération des CNNs 33
3.3.1 Analyse des Approches des CNNs sur les Plateformes Matérielles : Revue de la Littérature 35
3.3.2 Approches d'implémentation des CNNs sur FPGAs 36
3.4 Conclusion 47

4 Implémentation Utilisant la Synthèse de Haut Niveau : Approche et Méthodologie 48
4.1 Introduction 48
4.2 Synthèse de l'Approche Méthodologique Adoptée 49
4.3 Phase d'apprentissage 50
4.3.1 Prétraitement du jeu de données 51
4.3.2 Compilation du modèle 53
4.3.3 Validation et sauvegarde des poids et des biais du modèle 54
4.4 Analyse des Résultats et Discussion de la Phase d'Apprentissage 55
4.5 Implémentation Matérielle de l'Inférence et Stratégies d'optimisation . . . 57
4.5.1 Implémentation Fonctionnelle des Modules et Exploration de la Quantification 59
4.6 Résultats et interprétation 84
4.7 Étude comparative et Validation 85
4.8 Conclusion 86

5 Implémentation utilisant le registre de transfert logique : approche et méthodologie. 88
5.1 Introduction 88
5.2 Méthodologie d'implémentation 89
5.2.1 Résultats et interprétations 90
5.3 Conclusion 100

Bibliographie **104**

A Données Annexes **A**

B Schémas bloc et caractéristiques. **C**

Table des figures

2.1 Neurone biologique 11
2.2 neurone formel 11
2.3 Architecture à base de perceptrons des ANNs 12
2.4 Évolution chronologique des modèles CNNs 17
2.5 Architecture LeNet5 18
2.6 Opération de convolution 2D 19
2.7 Fonction d'Activation ReLU 20
2.8 Fonction d'Activation Sigmoid 21
2.9 Fonction d'Activation Tanh 21
2.10 Opération de mise en commun 22
2.11 Opération du calcul de convolution multi-kernel et multi-entrees 24
2.12 Un stride de 1 appliqué sur une image d'entrée 25
2.13 Paddind de 2 sur une image de 28x28 25
2.14 Partitionnement des données et des poids 28
2.15 Pipelining des fonctions 29
2.16 Parallélisation des opérations dans une boucle 29

3.1 Organigrame des différentes catégories d'approches 34
3.2 Organigramme d'implémentation Avec le HLS 37
3.3 Opérations d'élagage ou Pruning : (a) Élagage des connexions, (b) Élagage des neurones. 40
3.4 Opérations de transformation de Winograd 41
3.5 Modélisation en mode poids stationnaires 45
3.6 Modélisation en mode sorties stationnaires 46

4.1 Aperçu complet de l'approche adoptée 49

4.2 Chargement et répartition de l'ensemble de données 51
4.3 redimensionnement de l'ensemble de données en 32x32 en utilisant des valeurs de rembourrage de 2 52
4.4 Transformation de l'ensemble de données en un vecteur unidimensionnel 1-D 52
4.5 Normalisation des valeurs de pixel des ensembles d'entraînement et de test 52
4.6 Sommaire du modèle CNN LeNet5 construit 53
4.7 Ajustement des hyperparamètres de compilation 54
4.8 Structure et paramètres de la Matrice de confusion 54
4.9 Courbes de performances sur 20 époques 56
4.10 Courbes de performances sur 10 époques 56
4.11 Matrice de confusion 57
4.12 Histogramme $F_{F}core$ 57
4.13 Schéma bloc représentant l'interface du module de convolution 59
4.14 Format de la représentation en virgule fixe 60
4.15 Interface du module de convolution implémentée. 62
4.16 Cosimulation du module de convolution à base du format (24,4,20). 63
4.17 Profil des ressources du module de convolution à base du format (24,4,20). 63
4.18 Profil de performance du module de convolutionà base du format (24,4,20) 64
4.19 Cosimulation du module de convolution à base du format (23,3,20) 64
4.20 Profil des ressources du module de convolution à base du format (23,3,20). 65
4.21 Profil de performance du module de convolution à base du format (23,3,20) 65
4.22 Cosimulation du module Convolution à base du format (22,2,20). 65
4.23 Profil de ressources à base du format (22,2,20). 66
4.24 Cosimulation du module Convolution à base du format (21,2,19). 66
4.25 Profil de ressources à base du format (21,2,19). 67
4.26 Cosimulation du module Convolution à base du format (20,2,18). 67
4.27 Profil de ressources à base du format (20,2,18). 67
4.28 Cosimulation du module Convolution à base du format (19,2,17). 68
4.29 Profil de ressources à base du format (19,2,17). 68
4.30 Cosimulation du module Convolution à base du format (18,2,16). 69
4.31 Profil de ressources à base du format (18,2,16). 69
4.32 Cosimulation du module Convolution à base du format (17,2,15). 69
4.33 Profil de ressources à base du format (17,2,15). 70
4.34 Cosimulation du module Convolution à base du format (16,2,14). 70

4.35 Profil de ressources à base du format (16,2,14). 70
4.36 Cosimulation du module Convolution à base du format (15,2,13). 71
4.37 Profil de ressources à base du format (15,2,13). 71
4.38 Cosimulation du module de Convolution à base du format (14,2,12). 72
4.39 Profil de ressources à base du format (14,2,12). 72
4.40 Évolution de la précision du module de convolution en fonction de la taille du format en virgule fixe. 73
4.41 Évolution du nombre de flip-flops du module de convolution en fonction de la taille du format en virgule fixe. 74
4.42 Évolution du nombre de Look Up Table du module de convolution en fonction de la taille du format en virgule fixe. 74
4.43 Résultat de la Cosimulation du module de mise en commun 75
4.44 Interface du module mise en commun implémenté 76
4.45 Profil de Performances du module mise en commun 76
4.46 Profil de ressources du module mise en commun 77
4.47 Interface du module entièrement connectés implémenté. 77
4.48 Profil de performance(latence) du module entièrement connectées. 78
4.49 Profil de ressources du module entièrement connectées. 78
4.50 Architecture matérielle à base de flux de données 79
4.51 Métriques de performance de l'architecture séquentielle 80
4.52 Métriques de performance de l'architecture en flux de données 80
4.53 Estimations d'utilisation des bascules (Flip Flops) et des LUTs (Look Up Tables). 84
4.54 Estimations d'utilisation des BRAMS (Bloc RAMS) et des DSP48E 85
4.55 Analyse de l'évolution de la latence selon les options architecturales adoptées 85

5.1 Architecture de matrice systolique basée sur les types de poids et de sortie stationnaires 90
5.2 Hiérarchie des modules et les ressources internes qui composent la convolution 3x3. 91
5.3 Sommaire des ressources utilisées 91
5.4 Logigramme de l'implémentation du module de convolution dans un contexte de traitement systolique (Partie 1). 93

5.5 Logigramme de l'implémentation du module de convolution dans un contexte de traitement systolique (Partie 2). 94
5.6 Logigramme de l'implémentation du module de convolution dans un contexte de traitement systolique (Partie 3). 95
5.7 Schéma de câblage des différents blocs 96
5.8 Flux des caractéristiques d'entrée et poids vers les PEs. 97
5.9 Câblage et composants internes du bus de transmission des caractéristiques d'entrée vers les PEs. 98
5.10 Connectivité et Configuration des Composants Constituant les PEs 98
5.11 Rapport détaillé de la puissance consommée par le module de convolution. 99
5.12 Occupation matérielle du circuit sur la plateforme FPGA 99

A.1 Exemples de chiffres issus de l'ensemble de données MNIST importé . . . B

B.1 Schéma bloc de la carte FPGA VC707 C
B.2 Emplacements des composants de la carte VC707 . . . . . D

Liste des tableaux

2.1 Nombre de paramètres, taille de filtre et nombres de couches convolutives de ces modèles CNNs 17

4.1 Optimisation Séquentielle : Un Cheminement à Travers les Améliorations Successives. 83

4.2 Étude comparative avec des travaux antérieures 86

Liste des Algorithmes

2.1 Algorithme d'une operation de convolution Tridimentionnelle 27

4.1 Directives d'optimisation appliquées sur le processus de convolution 82

Nomenclature

Acronymes / Abréviations

ASIC	Circuit Intégré Spécifique à une Application (Application Specific Integrated Circuit)
BCI	Brain-Computer Interface (Brain-Computer Interface)
BRAM	Bloc Mémoire à accès aléatoire (Block Random Access Memory)
DRAM	Mémoire à accès aléatoire Dynamique (Dynamic Random Access Memory)
DSP	Processeur de signaux numériques (Digital Signal Processor)
EWMM	Multiplication Matricielle Élément par Élément(Element Wise Matrix Multiplication)
FPGA	Réseaux de portes Programmables en Site (Field Programmable Gate Arrays)
GPU	Unité de Traitement Graphique (Graphical Processing Unit)
HLS	Haut Niveau de synthèse (High Level Synthesis)
IoT	Internet des Objets (Internet of Thinks)
MNIST	Base de données modifiée de l'Institut National de Normalisation et de Technologie (Modified National Institute of Standards and Technology database)
OpenCl	Language de Calcul Ouvert (Open Computer Language)
OpenCV	Bibliothèque de Vision par Ordinateur en Source Ouverte (Open Computer Vision)

RTL	Niveau de Transfert Registre (Register Transfer Level)
SGD	Descente de gradient stochastique (Stochastic Gradient Descent)
SSVEP	Steady-State Visual Evoked Potentials (Steady-State Visual Evoked Potentials)
SVD	décomposition des valeurs singulières (Singular Value Decomposition)
TPU	Unité de Traitement Tensoriel (Tensor Processing Unit)
CPU	unité centrale de traitement (Central Processing Unit)
FFT	Fast Fourier Transform (Fast Fourier Transform)
GPU	Unité de Traitement Graphique (Graphical Processing Unit)
PE	Processing Element (Processing Element)
Verilog HDL	Langage de Description Matérielle à verification logique (Verification Logic Hardware Description Language)
VHDL	Langage de Description Matérielle pour les Circuits Intégrés Très Haute Vitesse (Very high speed integrated circuits Hardware Description Language)
AI	intelligence Artificielle (Artificial intelligence)
ANN	Réseau de Neurone Artificiel (ArtificiaL Neuronal Network)
CNN	Réseau de Neurone Convolutif (Convolutional Neuronal Network)
ReLu	Unité Linéaire Rectifiée (Rectified Linear Unit)
tanh	Tangente hyperbolique (Hyperbolic tangent)

Autres Symboles

A^T	Matrice de transformation Winograd
B^T	Matrice de transformation Winograd
FN	Faux Négatives
FP	Faux Positives
FP	Faux Positives

FP	Faux Positives
G	Matrice de transformation Winograd
Σ	Matrice SVD
S	Pas de glissement
$\cup$	Matrice SVD
$\vee$	Matrice SVD
VN	Vraies Négatives
VP	Vraies Positves
VP	Vraies Positves
VP	Vraies Positves

Chapitre 1

Introduction

1.1 Contexte de recherche

L'intelligence artificielle (AI) *(Artificial intelligence)* émerge comme un domaine de recherche extrêmement actif et dynamique, couvrant une vaste gamme de domaines d'application. Cette discipline en constante évolution s'étend de la médecine à la finance, en passant par la logistique et l'industrie. L'omniprésence de l'IA a radicalement transformé nos interactions avec la technologie et le monde qui nous entoure, apportant avec elle des avantages indéniables en termes d'efficacité et d'innovation. Cependant, cette avancée technologique suscite à la fois fascination et inquiétude quant à son impact sur notre quotidien et notre société dans son ensemble.

En effet, les premiers travaux de recherche en AI [1–4] ont jeté les bases des réseaux de neurones artificiels (ANN) *(ArtificiaL Neuronal Network)*, marquant le début d'une ère passionnante dans le domaine de l'apprentissage automatique et de l'AI. Inspirés par la structure des neurones biologiques, les ANNs ont été les pionniers dans la tentative de créer des modèles informatiques capables de simuler des processus d'apprentissage, de généralisation et de prise de décision similaires à ceux du cerveau humain.

Bien que les ANNs aient démontré leur potentiel pour résoudre un large éventail de problèmes, leur évolution a conduit à une forme plus spécialisée de réseau de neurones, à savoir les réseaux de neurones convolutifs (CNN) *(Convolutional Neuronal Network)*. Les CNNs résultent de plusieurs années de recherche visant à améliorer la capacité des modèles à traiter de manière efficace des données complexes, notamment des images. Inspirés par la façon dont le cerveau humain traite l'information visuelle à travers des processus hiérarchiques et des filtres de caractéristiques, les CNNs introduisent des couches de convolution

spéciales. Ces couches permettent l'extraction automatique de caractéristiques pertinentes des images, révolutionnant ainsi la façon dont les ordinateurs peuvent comprendre et analyser visuellement leur environnement. En raison de leur architecture en couches et de leur capacité à apprendre des caractéristiques, les CNNs ont été largement adoptés en vision par ordinateur pour des tâches telles que la reconnaissance d'objets, la détection d'images et la classification d'images. Dans le cadre de nos travaux de recherche, cette thèse se concentre principalement sur les CNNs et leur mise en œuvre sur une plateforme matérielle en vue d'accélérer les opérations liées à ces réseaux neuronaux.

1.2 Problématique

L'évolution notable des CNNs dans la résolution de problèmes complexes en vision par ordinateur représente une avancée incontestable. Cependant, cette progression remarquable s'accompagne de défis significatifs, mettant en lumière la nécessité de relever des obstacles majeurs, notamment dans le domaine délicat de leur implémentation matérielle. En raison de leur architecture complexe, les CNNs manifestent une forte demande en ressources computationnelles, les contraignant fréquemment à recourir à des unités de traitement puissantes et à un matériel haut de gamme. La complexité inhérente de leur structure rend impératif l'utilisation de technologies matérielles performantes afin d'assurer un fonctionnement optimal et de tirer pleinement parti de leur potentiel.

Dans cette perspective, diverses solutions de plates-formes matérielles ont été explorées, incluant les unités de traitement graphique (GPU) *(Graphical Processing Unit)*, les circuits intégrés spécifiques à une application (ASIC) *(Application-Specific Integrated Circuit)*, les unités de traitement tensorielle (TPU) *(Tensor Processing Unit)* et les reseaux de portes programmables en site (FPGA) *(Field Programmable Gate Arrays)*. Cette exploration vise à proposer des alternatives plus efficientes et économiques pour le traitement des CNNs.

Un autre défi, non négligeable, réside dans l'implémentation des CNNs, avec un accent particulier sur l'optimisation des performances, incluant la latence *(Latency)*, le débit *(throughput)*, et la consommation d'énergie. L'efficacité opérationnelle des CNNs est étroitement liée à la manière dont ils sont matériellement intégrés. Ces considérations soulèvent des questions importantes concernant la réduction des temps de traitement, l'augmentation de la vitesse de transfert des données, et la minimisation de la consommation énergétique, autant d'aspects clés pour assurer le déploiement optimal et efficient de

ces réseaux neuronaux dans des applications réelles. Ainsi, l'optimisation de la latence, du débit et de l'efficience énergétique représente un défi majeur dans l'implémentation des CNNs sur des FPGAs. La latence, définie comme le temps nécessaire pour traiter une entrée et générer une sortie, est capitale dans de nombreuses applications en temps réel, telles que la reconnaissance d'images en temps réel ou les systèmes autonomes. L'utilisation judicieuse des ressources disponibles sur le FPGA, telles que les blocs DSP (Digital Signal Processing) et les blocs logiques, ainsi que l'optimisation des chemins de données et la réutilisation des données sont des facteurs essentiels pour minimiser la latence. Le débit, en tant que paramètre essentiel, met en lumière la capacité intrinsèque du système à traiter simultanément un volume substantiel d'entrées. Lorsqu'il s'agit d'implémenter des CNNs sur un FPGA, l'optimisation de ce débit revêt une importance stratégique. Pour atteindre des performances maximales, il est impératif de tirer parti de manière optimale des ressources de calcul parallèle offertes par le FPGA. Cela implique la conception réfléchie d'architectures hautement parallèles, exploitant efficacement les unités de traitement disponibles pour traiter plusieurs opérations simultanément.

Toutefois, pour concevoir une architecture hautement parallèle sur FPGA, plusieurs considérations doivent être prises en compte. Premièrement, il est indispensable de posséder une compréhension approfondie des spécificités du réseau neuronal à implémenter. Cette connaissance permet d'identifier les caractéristiques clés du réseau, telles que sa topologie et ses exigences en termes de performances, qui influenceront directement la conception de l'architecture parallèle.

Deuxièmement, une gestion efficace des transferts de données entre les différents modules du réseau est cruciale. Cette gestion vise à éviter les goulots d'étranglement potentiels et à garantir une communication fluide et rapide entre les différentes composantes du système. Une analyse minutieuse des besoins en bande passante et des délais de latence dans les transferts de données est nécessaire pour concevoir des mécanismes de communication efficaces et optimisés.

En combinant une connaissance approfondie du réseau neuronal avec une gestion optimale des flux de données, il est possible de concevoir une architecture parfaitement adaptée, exploitant au mieux les capacités de calcul parallèle du FPGA tout en favorisant une interconnexion fluide entre ses différents éléments. Cette approche permet non seulement d'améliorer les performances de l'architecture parallèle, mais également de garantir une utilisation efficace des ressources matérielles disponibles sur le FPGA.

l'efficacité énergétique est également un enjeu majeur auquel se heurte la mise en

oeuvre de ces réseaux CNNs, en particulier avec la montée en puissance des applications embarquées et de l'Internet des objets (IoT) *(Internet of Thinks)*. Ces dispositifs sont alimentés par batterie ayant une autonomie limitée. dans ce contexte, il est essentiel de conduire une conception méticuleuse du circuit, en exploitant de manière avisée les ressources matérielles disponibles, dans le but de minimiser au maximum la consommation d'énergie. Cette approche englobe diverses pratiques, telles que l'optimisation des ressources matérielles, la réduction des accès mémoire, la réutilisation judicieuse des données, et d'autres stratégies visant à maximiser l'efficacité du système tout en minimisant son empreinte énergétique.

En résumé, accomplir ces divers objectifs exige une optimisation minutieuse à la fois de l'architecture matérielle et des algorithmes, assurant ainsi des performances en temps réel sans sacrifier la précision des résultats. Pour relever ces défis, il est impératif d'explorer des approches méthodologiques d'implémentation matérielle hautement efficaces, spécifiquement adaptées à chaque application, afin d'atteindre les niveaux de performance requis.

1.3 Motivation

L'implémentation des CNNs occupe une place centrale dans l'évolution de l'intelligence artificielle et de l'apprentissage profond. Ce processus s'articule autour de deux phases successives : la première étant la phase d'apprentissage, également appelée transfert d'apprentissage, suivie de la phase d'inférence. La phase de transfert d'apprentissage des CNNs consiste à utiliser un ensemble de données d'entraînement afin d'ajuster minutieusement les poids et les paramètres du réseau. Cela permet au modèle d'acquérir la capacité de reconnaître et de généraliser des motifs complexes présents dans les données, formant ainsi une base solide pour les tâches ultérieures d'inférence. Cette dernière consiste à utiliser ce modèle pour effectuer des prédictions sur de nouvelles données, en exploitant les connaissances acquises lors de la phase d'apprentissage.

Au cours de notre travail de thèse, nous accordons une attention particulière à la phase d'inférence et à son rôle essentiel dans l'application pratique des modèles de convolution neuronale CNNs. Notre objectif principal est d'apporter des contributions amélioratives aux performances du modèle lors de cette phase, en réduisant à la fois la latence et l'utilisation des ressources matérielles. Pour ce faire, nous examinons différentes approches et stratégies visant à optimiser l'efficacité de l'inférence, afin de rendre notre modèle plus

adapté aux applications du monde réel.

Dans cette optique et dans le cadre de nos travaux de recherche, notre objectif principal est de jouer un rôle significatif dans l'évolution des méthodologies et des techniques utilisées dans l'implémentation matérielle des CNNs sur une plateforme FPGA. Notre focalisation est particulièrement axée sur la création de méthodes fiables, optimales et facilement applicables à une gamme diversifiée de scénarios.

1.4 Contributions

Cette thèse se consacre à une exploration approfondie du défi que représente l'implémentation matérielle des réseaux de neurones convolutifs (CNNs) sur des plateformes FPGA. Bien que diverses études aient déjà abordé ce domaine de recherche [5–8], notre travail se distingue en tant qu'étude exploratoire visant à analyser de manière approfondie les différentes exigences de performance inhérentes aux implémentations matérielles sur FPGA. Dans cette perspective, notre objectif à travers cette recherche est de fournir un aperçu détaillé des besoins architecturaux spécifiques à chaque aspect des performances précédemment mentionnées. Cette étude s'inscrit dans la continuité des recherches existantes, cherchant à apporter une contribution substantielle en examinant les défis particuliers associés à l'intégration efficace des CNNs sur des FPGAs. Nous ambitionnons d'identifier les paramètres architecturaux cruciaux, de comprendre les implications de leur manipulation sur les performances générales du système, et de proposer des solutions optimisées pour surmonter les obstacles rencontrés.

Les apports majeurs de cette thèse peuvent être succinctement récapitulés de la manière suivante :

1. En premier lieu, nous élaborons un cadre théorique visant à approfondir notre compréhension de l'architecture matérielle des réseaux de neurones convolutifs sur les plates-formes FPGA. Cette démarche implique une décomposition minutieuse des concepts fondamentaux ainsi que l'identification des principaux éléments architecturaux. En procédant de la sorte, nous érigeons une base solide qui servira de fondement à nos travaux ultérieurs. À travers cette exploration théorique approfondie, nous aspirons à éclairer et à orienter nos futures étapes de recherche, assurant ainsi une approche solide et informée dans le développement de solutions matérielles efficaces pour les réseaux de neurones convolutifs.

2. Deuxièmement, nous entreprenons une analyse détaillée des différents composants d'un réseau de neurones convolutifs et élaborons une stratégie systématique visant à optimiser leur implémentation matérielle sur les FPGA. Cela inclut l'optimisation des opérations de convolution, de la gestion de la mémoire, et de la parallélisation pour maximiser les performances.
3. En troisième lieu, nous mettons en œuvre un prototype de réseau neuronal convolutif sur une plateforme FPGA et le comparons à des architectures alternatives telles que les CPU et les GPU. Cette comparaison permet d'évaluer l'efficacité de notre approche en termes de vitesse de traitement, de consommation d'énergie et d'autres métriques pertinentes.
4. En avant dernier lieu, notre étude s'attarde sur les deux environnements de développement prédominants pour les implémentations sur FPGA : le haut niveau de synthèse et le bas niveau de synthèse.
5. Enfin, nous discutons en détail des résultats obtenus et démontrons le potentiel des FPGA en tant qu'accélérateurs matériels pour les réseaux neuronaux convolutifs. Nous mettons en évidence les avantages et les limites de cette approche, ainsi que les perspectives futures pour l'utilisation des FPGA dans le domaine de l'apprentissage automatique.

Cette thèse vise donc à contribuer à une meilleure compréhension de l'implémentation matérielle des réseaux de neurones convolutifs sur FPGA, en fournissant des recommandations architecturales, des analyses de performance comparatives et des perspectives prometteuses pour cette technologie.

1.5 Données et outils d'implémentation

Pour simuler l'ensemble des méthodologies et approches élaborées au cours de nos travaux de recherche dans le cadre de cette thèse, nous avons eu recours à des ensembles de données de test et divers environnements de développement.

1. **Donnees de test :** nous avons opté pour l'utilisation des ensembles de données MNIST, reconnus pour leur pertinence et leur utilité dans l'évaluation des performances des modèles.

   Ces ensembles de données MNIST, composés d'images numériques de chiffres manuscrits, ont été sélectionnés en raison de leur large adoption dans la communauté

de la recherche en intelligence artificielle et en apprentissage automatique. Ils offrent un terrain d'expérimentation standardisé, permettant ainsi une comparaison objective des résultats obtenus avec nos différentes méthodologies.

2. **Environnements de développement :** En ce qui concerne l'environnement de développement, notre approche a été stratégiquement guidée par les différentes phases du processus, à savoir l'apprentissage et l'inférence, chacune requérant des caractéristiques spécifiques de développement.

   Dans la phase d'apprentissage, nous avons choisi des environnements tels que Colab. Colab a été privilégié en raison de son infrastructure cloud robuste et évolutive, permettant une utilisation efficace des ressources nécessaires pour l'entraînement de modèles complexes. Parallèlement, nous avons utilisé des outils de développement de pointe tels que TensorFlow, Keras et NumPy, qui offrent des fonctionnalités de haut niveau pour simplifier la conception et la formation des modèles, garantissant ainsi une mise en œuvre efficace de nos algorithmes d'apprentissage.

   Dans la phase d'inférence, nous avons orienté notre choix vers l'environnement Vivado de Xilinx, en utilisant ses deux versions de développement, à savoir High-Level Synthesis (HLS) et Register-Transfer Level (RTL). Cette transition s'explique par la nécessité de maximiser l'efficacité de l'inférence en exploitant les fonctionnalités spécifiques de l'environnement Vivado, assurant ainsi une cohérence optimale entre les phases d'apprentissage et d'inférence de notre projet. Cette approche globale a été cruciale pour garantir la réussite de notre implémentation du processus complet de développement des modèles.

Pour la partie expérimentale de notre recherche, nous avons opté pour l'utilisation du Kit d'évaluation Virtex™ 7 FPGA VC707. Cette plateforme se distingue par son caractère complet, sa grande flexibilité et sa capacité à gérer des débits élevés. Elle repose sur la puce Virtex 7 XC7VX485T-2FFG1761C, intégrant des composants matériels de base, des outils de conception, des propriétés intellectuelles (IP) ainsi que des conceptions de référence pré-vérifiées. Ces éléments sont particulièrement adaptés aux systèmes exigeant une connectivité série à haute performance et une interface mémoire avancée.

Le Kit VC707 inclut des conceptions de référence pré-vérifiées, ainsi que des connecteurs FPGA Mezzanine Connectors (FMC) conformes aux normes de l'industrie. Ces caractéristiques offrent la possibilité de mettre à l'échelle et de personnaliser le kit avec des cartes filles, renforçant ainsi sa polyvalence et son adaptabilité à différents besoins

expérimentaux.

Pour de détails plus approfondis concernant cette plateforme expérimentale, veuillez voir l'appendice B.1

1.6 Organisation du manuscrit

La présente thèse se consacre à l'étude approfondie de l'optimisation et de la mise en œuvre des CNNs en exploitant les capacités d'une plateforme matérielle FPGA. Son organisation est structurée en deux parties distinctes, chacune composée de deux chapitres. La première partie, de nature théorique, englobe les chapitres deux et trois, tandis que la seconde partie, expérimentale, est constituée des chapitres quatre et cinq.

Après avoir établi le contexte, exposé la problématique et détaillé les motivations de notre recherche dans un premier chapitre introductif, le deuxième chapitre de notre étude s'engage dans une analyse approfondie des Réseaux de Neurones Convolutifs (CNN). Cette section essentielle de notre travail s'attache à offrir une vue d'ensemble exhaustive et minutieuse de l'évolution, des principes fondamentaux et des multiples applications des CNN. Dans cette optique, nous commençons notre exploration en examinant les fondements des CNN, retraçant leur développement depuis leur émergence jusqu'à leur position actuelle en tant que pilier central de la vision par ordinateur et du traitement d'images.

Par la suite, notre investigation se concentre sur une analyse approfondie des principes sous-jacents des CNN, mettant particulièrement l'accent sur leur architecture en couches et les opérations de convolution. Ce faisant, nous nous efforçons de fournir une compréhension détaillée des mécanismes internes de ces réseaux, en mettant en lumière leur fonctionnement et leur capacité à extraire des caractéristiques pertinentes et complexes des données d'entrée.

Dans le **chapitre 3**, nous procéderons à une revue exhaustive des approches et méthodologies pertinentes à notre domaine de recherche. Nous entreprendrons une cartographie détaillée des différents axes, mettant en évidence les tendances actuelles, les lacunes dans la littérature existante, ainsi que les opportunités pour des contributions significatives. En outre, nous examinerons les travaux de recherche récents, les progrès notables et les approches émergentes afin de fournir une analyse approfondie de chaque branche.

Dans le quatrième chapitre de notre thèse, nous présenterons une approche d'implémentation exhaustive englobant les deux phases essentielles du processus d'implémentation des CNNs : l'apprentissage et l'inférence. Cette approche est conçue dans un souci

d'inclusivité, prenant en considération l'intégralité du cycle de déploiement des CNN, depuis leur entraînement initial jusqu'à leur utilisation opérationnelle.

Dans le cadre de cette approche, nous explorerons les différentes directives d'optimisation ainsi que les diverses métriques d'analyse et de synthèse disponibles sur la plateforme de développement Vivado HLS. Ces directives d'optimisation englobent une gamme étendue de techniques et de stratégies visant à améliorer les performances et l'efficacité des implémentations matérielles des CNN. Quant aux métriques d'analyse et de synthèse, elles fournissent des outils précieux pour évaluer la qualité et l'efficacité des solutions mises en œuvre, permettant ainsi une rétroaction continue et un raffinement itératif de l'approche adoptée.

Le cinquième chapitre explore une approche novatrice visant à optimiser les performances des CNNs. Cette approche se concentre sur l'augmentation du parallélisme des opérations de convolution, en mettant en avant l'utilisation des matrices systoliques. Ces matrices jouent un rôle crucial dans l'accélération des calculs des CNN en tirant parti de leur architecture hautement parallèle. Cette approche offre ainsi la possibilité d'améliorer significativement les performances des CNN en réduisant les temps de traitement, tout en maintenant une efficacité énergétique optimale.

Nous conclurons cette thèse par une section finale visant à mettre en lumière les conclusions tirées des différentes contributions apportées dans ce travail de recherche. Nous examinerons également les possibilités futures et les pistes d'amélioration qui pourraient découler de cette étude.

Chapitre 2

Réseaux de Neurones Convolutifs (CNNs)

2.1 Introduction

Dans ce chapitre, nous entreprendrons une analyse approfondie du contexte historique, théorique et pratique des CNNs. Nous débuterons en revisitant les idées fondamentales qui ont conduit à la conception des CNN, en soulignant les contributions significatives de chercheurs visionnaires. Par la suite, nous explorerons les principes théoriques fondamentaux sous-tendant les CNNs, mettant en lumière des concepts essentiels tels que la convolution, les fonctions d'activation, de mise en commun ou regroupement (Pooling), et les couches entièrement connectées (Fully Connected). Ensuite, nous aborderons l'état de l'art de ces réseaux en examinant l'évolution des modèles CNN au cours des dernières années. De plus, nous plongerons dans les avancées technologiques et les développements algorithmiques qui ont catalysé l'essor de l'implémentation des CNNs, en examinant les progrès significatifs dans les domaines du calcul parallèle, et de l'optimisation des performances.

2.2 Contexte historique des algorithmes AI

Au cours des deux dernières décennies, les algorithmes de traitement d'images et de vision par ordinateur ont connu une renaissance remarquable grâce aux premiers travaux de recherche aboutits en AI. L'AI est un concept algorithmique inspiré par la structure biologique du cortex visuel humain, décrivant la capacité d'une machine à apprendre

de manière autonome. Bien que l'AI soit un domaine relativement récent, son concept remonte aux années 1940 lorsque Warren McCulloch et Walter Pitts [9] ont publié leur premier modèle mathématique du ***"neurone formel"***, directement inspiré du fonctionnement des neurones biologiques. Les Figures 2.1 et 2.2 illustrent respectivement le neurone biologique et le neurone formel. Cette convergence entre les sciences biologiques et l'informatique a jeté les bases de l'apprentissage automatique et de l'intelligence artificielle modernes, marquant le début d'une révolution dans la vision par ordinateur, l'analyse d'images et la compréhension des modèles visuels complexes.

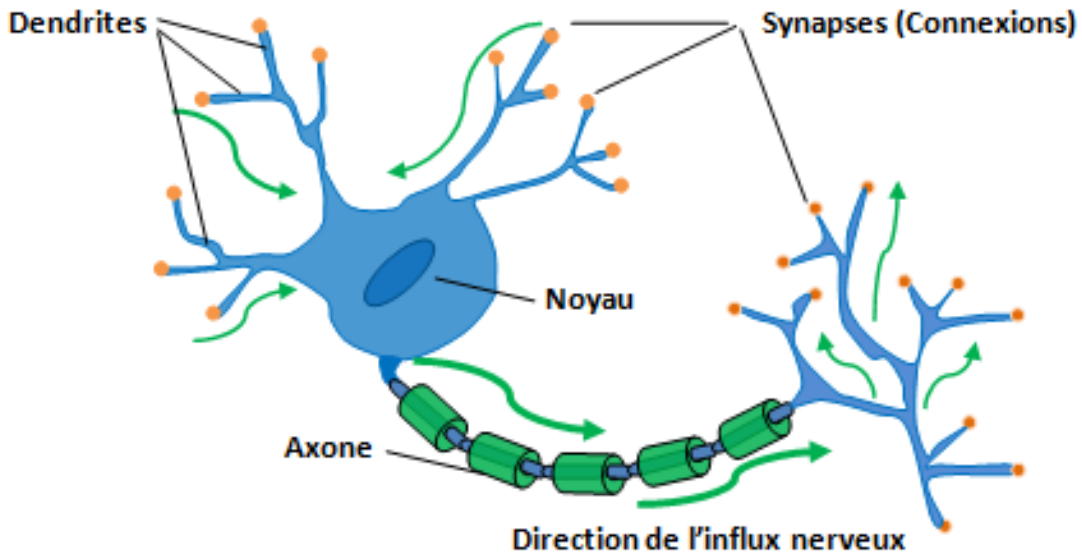


Figure 2.1: Neurone biologique

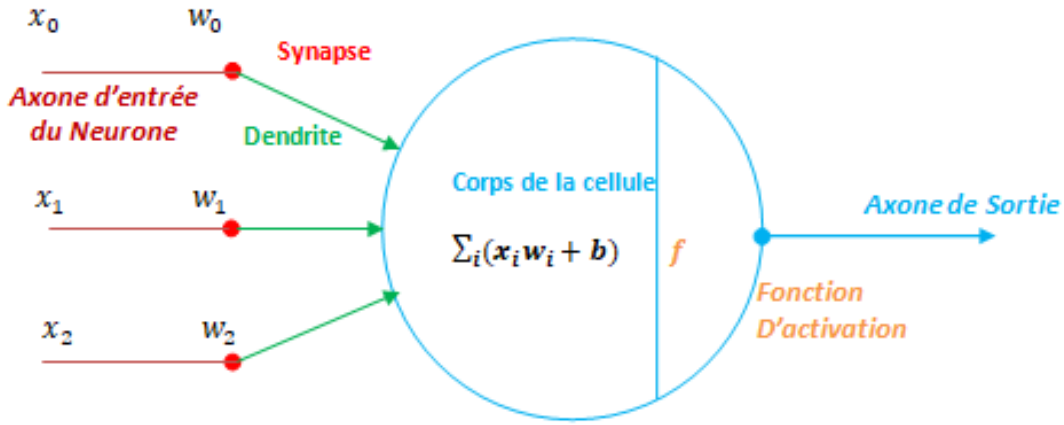


Figure 2.2: Neurone Formel.

2.3 Réseaux de neurones artificiels

Les réseaux de neurones artificiels (ANNs) représentent une avancée significative dans le domaine de l'intelligence artificielle et de l'apprentissage automatique. Leur émergence fait suite au développement du neurone formel puis celui du perceptron, qui a été introduit par Frank Rosenblatt (1957) [10]. Le perceptron a posé les bases de la modélisation mathématique des neurones biologiques et a marqué le début de l'exploration des réseaux de neurones. La figure 2.3 illustre l'architecture en perceptron des ANNs.

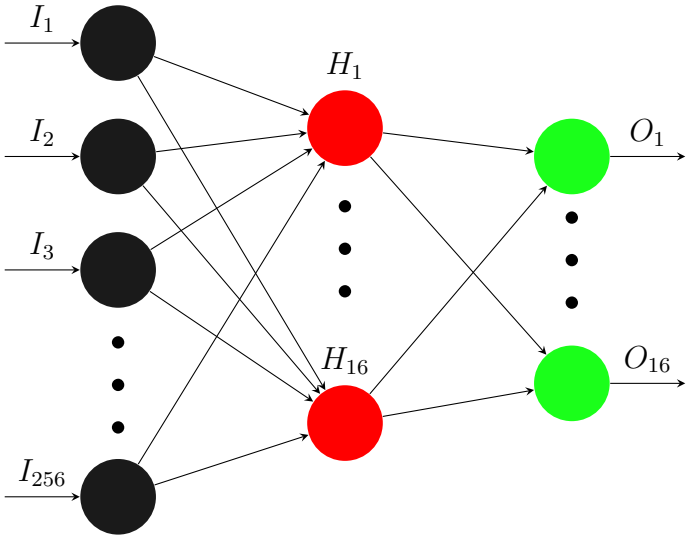


Figure 2.3: Architecture à base de perceptrons des ANNs .

2.4 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs (CNN) sont étroitement liés aux fondements de l'intelligence Artificielle. Ils trouvent leur origine dans les travaux de Y. LeCun et al. [11] à la fin des années 1980 et au début des années 1990. Ils ont développé des modèles de réseaux neuronaux convolutionnels pour la reconnaissance de caractères manuscrits, en particulier le modèle LeNet-5. Ces premiers CNNs étaient conçus pour imiter la façon dont le cerveau humain traite visuellement l'information, en utilisant des couches de convolutions pour extraire des caractéristiques importantes des images. Depuis lors, les CNNs ont connu une évolution spectaculaire et sont devenus la pierre angulaire de nombreuses applications en vision par ordinateur [12, 13], notamment la reconnaissance d'objets, la détection d'images [12, 14] et de nombreux autres domaines d'application [15–17].

Les modèles CNNs sont composés de plusieurs couches fondamentales, notamment les

couches de convolution, les couches de mise en commun (Pooling) et les couches entièrement connectées (Fully Connected). Chacune de ces couches joue un rôle capital dans le traitement des données. En somme, les réseaux de neurones convolutifs sont structurés de manière à effectuer une analyse progressive des données visuelles, de l'extraction des caractéristiques à la classification finale. Cette architecture en couches est fondamentale pour leur capacité à traiter efficacement des informations visuelles complexes dans une variété d'applications en vision par ordinateur.

2.5 Modèles CNNs

De nombreux modèles CNNs ont été développés pour une variété de tâches en vision par ordinateur, notamment la classification d'images, la détection d'objets, la segmentation, et bien d'autres. Ces modèles se distinguent les uns des autres par leur profondeur, leur architecture et leur complexité. Voici une sélection de certains des modèles CNNs les plus célèbres, chacun ayant apporté sont lot de contributions à l'évolution de la vision par ordinateur et de l'apprentissage profond . Le choix du modèle dépendra de la tâche spécifique et des contraintes de ressources. :

LeNet5 : Mis au point par Yann LeCun et et al. en 1998 [11], le modèle LeNet5 constitue une avancée significative dans le domaine des réseaux CNNs. Sa création a marqué une étape importante dans le développement de la reconnaissance de caractères manuscrits, ouvrant la voie à diverses applications en vision par ordinateur. En intégrant des couches convolutives et de sous-échantillonnage, le LeNet5 a démontré une efficacité notable dans la détection de motifs complexes, posant ainsi les bases pour de nombreuses architectures CNN contemporaines

AlexNet : Mis au point par Alex Krizhevsky et ses collaborateurs en 2012 [18], AlexNet s'est distingué en remportant le prestigieux concours ImageNet la même année. Cette victoire historique a marqué un tournant majeur dans l'histoire des réseaux de neurones convolutionnels (CNN), mettant en lumière leur capacité exceptionnelle à traiter efficacement des ensembles de données massifs comme ImageNet, qui comprend des millions d'images. En démontrant une performance remarquable dans la classification d'images à grande échelle, AlexNet a contribué de manière significative à populariser l'apprentissage profond et à stimuler l'exploration de nouvelles architectures et techniques dans ce

domaine en pleine croissance.

VGG16 et VGG19 : Les chercheurs du Visual Geometry Group (VGG) de l'Université d'Oxford ont donné naissance à deux géants de l'apprentissage profond : VGG16 et VGG19 [19]. Ces architectures ont marqué un tournant dans le paysage de la vision par ordinateur en raison de leur profondeur remarquable et de l'utilisation de convolutions uniformes de taille réduite (3x3). Cette approche démontre la puissance de la simplicité et de l'homogénéité dans la construction de réseaux neuronaux, en permettant une meilleure représentation des caractéristiques complexes des données visuelles. De plus, la profondeur accrue de ces modèles leur confère une capacité de généralisation améliorée, les rendant particulièrement efficaces pour des tâches telles que la classification d'images et la détection d'objets dans des scènes complexes. L'impact de VGG16 et VGG19 s'étend bien au-delà de la simple reconnaissance d'images, influençant profondément le développement de nombreuses autres architectures CNN et inspirant de nouvelles approches dans le domaine de l'apprentissage profond.

GoogLeNet (Inception) : Fruit de l'effort collectif de l'équipe de recherche de Google [20], le modèle GoogLeNet a marqué une avancée significative dans le domaine de l'apprentissage profond. En introduisant l'architecture Inception, il a repoussé les frontières de la capacité des réseaux neuronaux à comprendre et à extraire des motifs complexes dans les données visuelles. Cette architecture novatrice a révolutionné la manière dont les CNN sont conçus, en exploitant des modules de convolution parallèles pour capturer des caractéristiques à différentes échelles spatiales, permettant ainsi une représentation plus riche des images. En outre, GoogLeNet a été salué pour sa capacité à maintenir un équilibre délicat entre la profondeur du réseau et sa complexité computationnelle, ce qui en fait une architecture efficace même sur des ressources limitées. Son impact s'étend bien au-delà de la reconnaissance d'images, stimulant des avancées dans des domaines tels que la détection d'objets, la segmentation sémantique et même la génération d'images. En résumé, GoogLeNet a ouvert la voie à une nouvelle ère d'architectures de réseaux neuronaux, façonnant ainsi le paysage de l'apprentissage automatique moderne.

ResNet : Kaiming He et al. ont introduit les réseaux résiduels, couramment appelés ResNets, en 2016 [21]. Ces architectures révolutionnaires ont apporté une solution élégante au défi de la disparition des gradients dans les réseaux de neurones profonds. En permettant la formation de réseaux CNN significativement plus profonds, les ResNets ont ouvert de nouvelles perspectives dans le domaine de l'apprentissage en profondeur. Leur

conception novatrice repose sur l'idée d'introduire des connexions résiduelles, permettant ainsi de préserver les informations à travers les couches du réseau. Cette approche a non seulement amélioré la convergence de l'entraînement, mais a également conduit à des modèles plus performants, capables de capturer des caractéristiques complexes et abstraites des données. En repoussant les limites de la profondeur des réseaux neuronaux, les ResNets ont joué un rôle crucial dans de nombreuses applications telles que la vision par ordinateur, la reconnaissance vocale et le traitement du langage naturel.

MobileNet : Introduit par Howard et al. en 2017 [22], MobileNet représente une avancée notable dans le développement des réseaux neuronaux adaptés aux appareils mobiles. Ce réseau innovant répond à une demande croissante d'efficacité computationnelle dans le domaine de l'intelligence artificielle embarquée. En exploitant des techniques telles que les convolutions profondes et les couches de profondeur séparable, MobileNet parvient à réduire considérablement la charge de calcul tout en conservant des performances de classification compétitives. Cette approche ouvre la voie au déploiement d'applications d'apprentissage automatique sophistiquées sur des dispositifs aux ressources limitées, offrant ainsi de nouvelles perspectives dans des domaines comme la vision par ordinateur embarquée, la reconnaissance d'objets en temps réel et les interfaces utilisateur intelligentes sur les appareils mobiles. De plus, la souplesse et la légèreté de MobileNet en font une option privilégiée pour les applications nécessitant une faible consommation d'énergie et une latence minimale, ce qui renforce son rôle central dans la démocratisation de l'intelligence artificielle sur les plateformes mobiles.

EfficientNet : Les modèles EfficientNet, fruit du travail de Mingxing Tan et Quoc V. Le, représentent une avancée significative dans le domaine de l'apprentissage automatique [23]. Leur conception repose sur un équilibre subtil entre précision et efficacité des ressources, ce qui en fait des outils essentiels pour une variété d'applications. En combinant des techniques innovantes telles que l'optimisation des architectures de réseau et le dimensionnement équilibré des modèles, les EfficientNets parviennent à maximiser les performances tout en minimisant les coûts de calcul. Cette approche révolutionnaire a des implications profondes dans de nombreux domaines, de la vision par ordinateur à la reconnaissance vocale, où les contraintes de ressources sont souvent un défi majeur. En offrant une solution efficace et précise, les modèles EfficientNet ouvrent de nouvelles perspectives pour le déploiement de l'intelligence artificielle dans des environnements avec des ressources limitées.

DenseNet : Les DenseNets, ou Réseaux Convolutifs Densément Connectés, introduits par Huang et al. en 2017 [24], représentent une innovation significative dans le domaine des réseaux neuronaux convolutifs (CNN). Contrairement aux architectures traditionnelles où chaque couche est connectée uniquement à la suivante, les DenseNets établissent des connexions denses entre toutes les couches, favorisant ainsi un échange d'informations riche et continu. Cette conception unique offre plusieurs avantages : elle améliore la circulation du gradient lors de l'entraînement, réduit le nombre de paramètres à optimiser, et encourage le réutilisation des caractéristiques à différentes échelles. De plus, les DenseNets ont démontré une résilience accrue contre le surajustement, grâce à leur capacité à exploiter efficacement les informations provenant de toutes les couches du réseau. Ces propriétés en font une architecture populaire dans une variété de tâches de vision par ordinateur, de la segmentation d'images à la classification d'objets, contribuant ainsi de manière significative à l'avancement de l'apprentissage automatique.

Xception : Imaginé et réalisé par François Chollet [25], Xception émerge comme une innovation notable dans le paysage des architectures de réseaux neuronaux convolutifs. Inspiré par l'architecture Inception, ce modèle repousse les limites de la performance en adoptant une approche novatrice : les convolutions séparables en profondeur. Cette stratégie permet à Xception d'explorer de manière plus efficace les relations spatiales et les caractéristiques des données, conduisant à une représentation plus riche et à une meilleure généralisation. En séparant explicitement la capture des caractéristiques spatiales et des relations inter-canal, Xception minimise la redondance des calculs et améliore l'efficacité de l'apprentissage. Cette conception intelligente permet à Xception d'atteindre des performances remarquables dans une variété de tâches, allant de la classification d'images à la détection d'objets, tout en offrant une efficacité computationnelle supérieure par rapport à d'autres architectures. Ainsi, Xception incarne l'ingéniosité et l'ingénierie sophistiquée qui caractérisent les progrès constants dans le domaine de l'apprentissage automatique.

SqueezeNet : Initié pour répondre au besoin croissant de modèles compacts sans compromettre les performances, SqueezeNet [26] représente une avancée significative dans le domaine de l'apprentissage profond. Son architecture novatrice réduit de manière drastique l'empreinte mémoire requise, offrant ainsi une solution efficace pour les environnements où les ressources sont limitées, tels que les appareils mobiles ou les systèmes embarqués. En exploitant intelligemment des couches de convolution avec des filtres de taille réduite et en introduisant des modules de compression, SqueezeNet parvient à main-

tenir des performances comparables à celles des modèles plus volumineux. Cette approche révolutionnaire ouvre la voie à de nouvelles possibilités d'intégration d'algorithmes d'apprentissage profond dans des dispositifs où les contraintes de mémoire et de puissance sont cruciales. En outre, la conception élégante de SqueezeNet illustre l'importance croissante de l'efficacité des modèles dans le développement d'applications d'intelligence artificielle à l'échelle mondiale.

l'évolution chronologique de tous ces modèles CNNs est illustrée dans la figure 2.4.

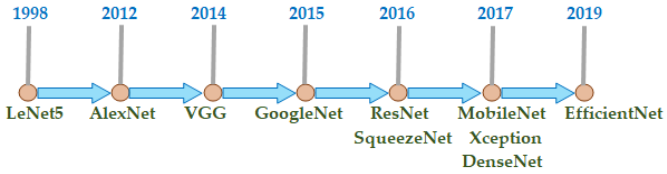


Figure 2.4: Évolution chronologique des modèles CNN.

Tableau 2.1: Nombre de paramètres, taille de filtre et nombres de couches convolutives de ces modèles CNNs

Caractéristiques des Modèles CNNs					
Modèle CNN	Année	Couche convolution	Taille Filtre	Dimensions Entrée	Total Paramètres K=10^3, $M = 10^6$
LeNet5	1998	5	5x5	1x28x28	60K
AlexNet	2012	7	11x11 5x5 3x3	3x224x224	62M
VGG11/16/19	2014	11, 16, 19	3x3	3x224x224	138M
GoogleNet	2015	16	1x1 3x3 5x5	3x224x224	205M
ResNet	2016	50	7x7	3x224x224	25.6M
SqueezeNet	2016	10	3x3	3x227x227	7.69M
MobileNet	2017	13	3x3	3x224x224	4.2M
Xception	2017	__	3x3	3x229x229	23.6M
DenseNet	2017	__	3x3	3x224x224	20.2 M
EfficientNet	2019	__	3x3, 5x5	3x224x224	5.3M

2.6 Analyse en Profondeur des Couches Constitutives des CNNs

Les CNNs sont des architectures profondes composées de plusieurs couches interconnectées. L'agencement et la profondeur de ces couches varient en fonction de la complexité de la tâche à accomplir. la Figure 2.5 illustre un exemple d'architecture en couches du model CNN LeNet5.

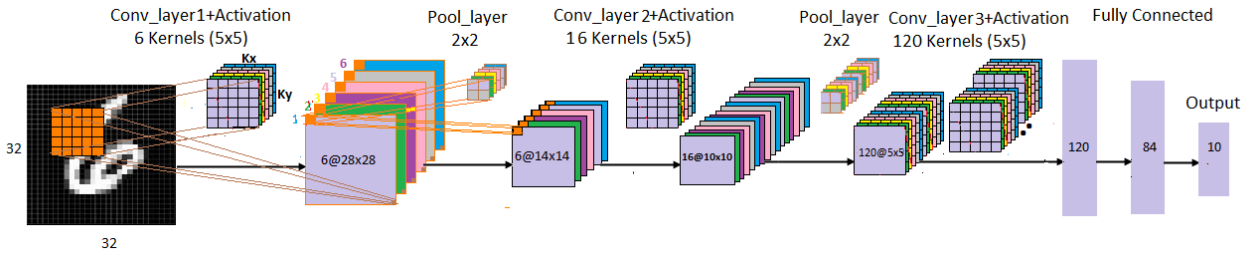


Figure 2.5: Architecture du Model CNN LeNet5

2.6.1 Couche de convolution

Les couches de convolution revêtent une importance capitale dans le processus d'extraction des caractéristiques significatives à partir des images en utilisant des filtres spécifiques adaptés aux données d'entrée. Ces filtres agissent comme des détecteurs de motifs, permettant ainsi de saisir des informations cruciales telles que les contours, les textures et les formes présentes dans les images. Cette étape fondamentale permet aux CNNs de développer une compréhension hiérarchique des données, en identifiant tout d'abord des caractéristiques de base comme les contours, puis en les combinant pour former des caractéristiques plus complexes, telles que des objets ou des motifs reconnaissables.

Illustrant cette opération, la Figure 2.6 expose de manière concrète le fonctionnement d'une opération de convolution en deux dimensions (2D), fournissant ainsi une représentation visuelle de ce processus essentiel dans le traitement des données par les CNNs.

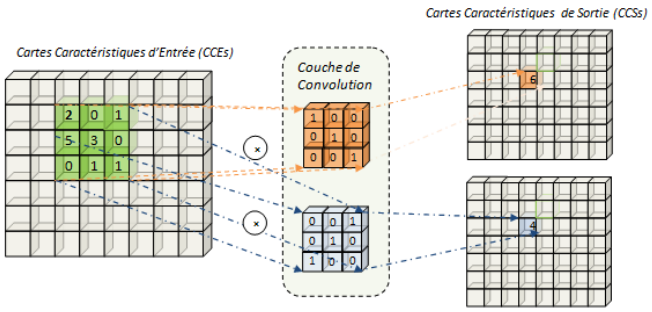


Figure 2.6: Opération de convolution 2D utilisant couche de deux Kernels de taille 3x3.

La convolution est une opération fondamentale dans le traitement des images par les Convolutional Neural Networks (CNNs). Elle implique l'application d'un filtre sur une image pour détecter ses caractéristiques principales. Pour ce faire, le filtre est déplacé sur l'image, et à chaque position, une multiplication élémentaire est effectuée entre le filtre et les parties correspondantes de l'image. Ensuite, toutes ces valeurs sont additionnées pour obtenir une unique valeur de sortie. Cette valeur est ensuite soumise à une fonction d'activation pour produire la sortie finale de la couche convolutive.

Ce processus permet à la couche convolutive de détecter les caractéristiques et les motifs essentiels des images, tels que les bords et les courbes. Ce qui est remarquable, c'est que ce mécanisme permet aux CNNs d'extraire ces caractéristiques sans nécessiter une ingénierie manuelle préalable des caractéristiques. Cette automatisation est une avancée puissante et économique en termes de temps, car elle permet au réseau de s'adapter à diverses tâches de traitement d'images sans intervention humaine directe.

Après cette étape de convolution, la sortie est généralement introduite dans la couche suivante du réseau, souvent une couche de mise en commun. Cette couche permet de réduire la dimension de la sortie tout en préservant les caractéristiques essentielles, facilitant ainsi le processus d'apprentissage et la généralisation du modèle aux données nouvelles.

2.6.2 Fonctions d'activation

Les couches de convolution, de mise en commun et les couches entièrement connectées sont des éléments essentiels des CNNs, jouant un rôle central dans l'extraction des caractéristiques des données en entrée et dans la production de représentations significatives. De manière similaire, les fonctions d'activation sont fondamentales, introduisant de la non-linéarité dans les sorties des neurones pour permettre au réseau d'apprendre des relations complexes. Elles régulent également les valeurs de sortie des neurones, limitant

leur plage pour stabiliser le processus d'apprentissage et prévenir la saturation, ce qui souligne l'importance du choix approprié de la fonction d'activation dans la conception des CNNs. Ainsi, parmi les fonctions d'activation existantes, on retrouve :

1. La fonction d'activation Unité Linéaire Rectifiée (ReLu)(Rectified Linear Unit) est la plus utilisée actuelement. Elle est définie par l'Equqtion 2.1,alorsque la figure 2.7 illustre son graphique..

$$f(x) = max(x, 0) \tag{2.1}$$

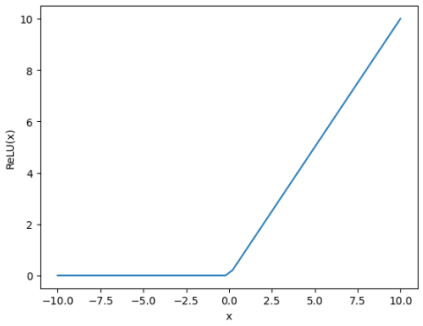


Figure 2.7: Fonction d'Activation ReLU

   En d'autres termes, pour toute valeur d'entrée x, la fonction ReLU renvoie x si x est positif, et 0 0 sinon. Graphiquement, cela se traduit par une sortie linéaire pour les valeurs positives de x et une sortie nulle pour les valeurs négatives de x. La fonction ReLU est appréciée pour sa simplicité et son efficacité, contribuant à l'accélération de l'apprentissage dans de nombreux scénarios.

2. La fonction d'activation Sigmoid est exprimée par l'Equation 2.2, où x est une variable réelle en entrée. Cette fonction est particulièrement utilisée dans le contexte de la classification binaire, car elle permet de modéliser la probabilité d'appartenance à une classe donnée. De plus, l'atout majeur de la fonction sigmoid réside dans sa capacité à être différenciable, ce qui facilite l'ajustement des poids du réseau lors de la phase de rétropropagation. Cependant, il est à noter que la fonction sigmoid peut présenter un problème de saturation lorsque les valeurs d'entrée deviennent extrêmement grandes ou petites, provoquant des gradients proches de zéro et ralentissant ainsi le processus d'apprentissage du réseau. Son grahique est illustré dans la Figure 2.8.

$$f(x) = \frac{1}{\exp(x) + \exp(-x)} \tag{2.2}$$

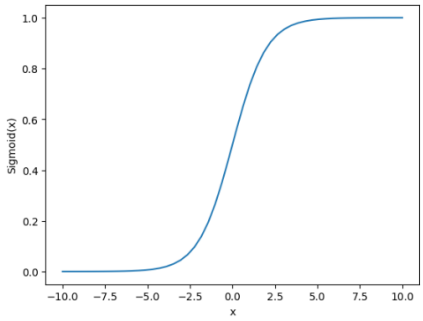


Figure 2.8: Fonction d'Activation Sigmoid

3. La fonction d'activation tangente hyperbolique (tanh) est similaire à la fonction sigmoïde mais avec une plage de valeurs de sortie allant de -1 à 1, ce qui la rend centrée autour de zéro et est exprimée par lEquation 2.3 et son grahique est illustré dans la Figure 2.9. Elle est souvent préférée à la fonction sigmoïde car elle produit des valeurs de sortie plus grandes, ce qui peut améliorer la stabilité et les performances du modèle.

$$f(x) = tanh(x) = \frac{\exp^{(+x)} - \exp^{(-x)}}{\exp^{(+x)} + \exp^{(-x)}} \tag{2.3}$$

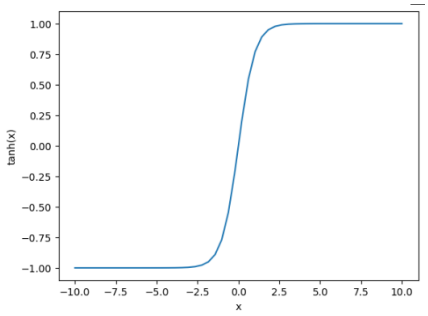


Figure 2.9: Fonction d'Activation Tanh

4. La fonction Softmax : Elle est généralement utilisée dans le domaine de l'apprentissage automatique, en particulier dans les tâches de classification et de probabilité. Elle est couramment utilisée pour transformer un vecteur de scores ou d'activations en une distribution de probabilités. La fonction Softmax prend en entrée un vecteur de nombres réels $X_i \in \mathbb{R}^{\kappa}$ et les convertit en probabilités voir Equation 2.4, de telle sorte que la somme de ces probabilités soit égale à un '1'.

$$X_i = \begin{bmatrix} x_1 \\ x_2 \\ . \\ \\ x_n \end{bmatrix} \qquad S(x_i) = \frac{\exp^{(x_i)}}{\sum_{i=1}^{\infty} \exp^{(x_i)}} \tag{2.4}$$

2.6.3 couche *"Pooling"* ou mise en commun

Les couches de pooling interviennent après les couches de convolution. Leur rôle est de réduire la dimensionnalité ou sous échantillonnage des caractéristiques extraites. Cela se fait en réduisant la taille des zones traitées, généralement en prenant la valeur maximale **"Maxpooling"** ou en effectuant une moyenne **"Average pooling"** des valeurs dans chaque zone , voir Figure 2.10. Cette réduction de dimension permet de conserver les informations essentielles tout en réduisant la complexité du modèle, ce qui est bénéfique pour la gestion de la mémoire et la vitesse de calcul.

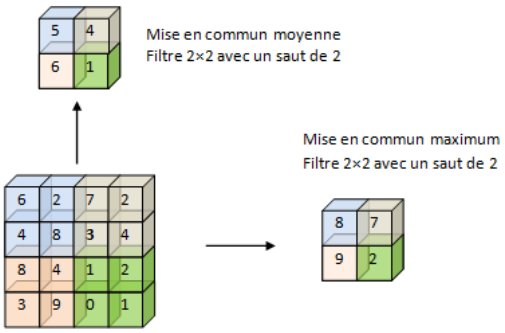


Figure 2.10: Opération de mise en commun moyenne et maximum de taille 2x2 et un stride de 2.

2.6.4 Couche entièrement connectée *"Fully connected"*

La couche entièrement connectée, parfois désignée sous le terme de couche dense, revêt une importance capitale au sein des réseaux de neurones artificiels, où elle assume une fonction cruciale dans la transformation des caractéristiques extraites des couches antérieures en des prédictions ou des classifications finales. Dans cette configuration, chaque neurone de la couche entièrement connectée est interconnecté à tous les neurones de la couche précédente, établissant ainsi une connexion dense. Cette architecture permet au

modèle d'appréhender des relations complexes entre les caractéristiques extraites, en capturant des motifs et des interactions plus abstraites au sein des données.

En procédant à une combinaison linéaire des entrées, pondérées par des poids spécifiques attribués à chaque connexion, suivie généralement par l'application d'une fonction d'activation non linéaire telle que ReLU (Rectified Linear Unit) ou une fonction sigmoïde, la couche entièrement connectée introduit de la non-linéarité et de la complexité dans le modèle. Cela permet au réseau d'apprendre des représentations hautement discriminatives des données, en vue de la tâche de classification ou de régression.

2.7 Opérations de la couche de convolution

Les opérations de la couche de convolution dans les CNN jouent un rôle capital dans l'extraction des caractéristiques des données d'entrée, contribuant ainsi à la capacité du modèle à reconnaître et à interpréter des motifs significatifs. Cependant, malgré leur importance, ces opérations présentent des défis majeurs en termes de temps de calcul et de consommation d'énergie. En effet, il est bien établi que ces opérations sont souvent responsables de la majeure partie de la charge de calcul et de la consommation énergétique dans un CNN. Cette réalité soulève des préoccupations quant à l'efficacité et à la durabilité des implémentations de CNN, en particulier dans des environnements où les contraintes de calcul et d'énergie sont prédominantes.

Des études antérieures, telles que [18, 19, 21] , ont mis en évidence l'impact significatif de ces opérations sur les performances globales des CNN. Elles ont montré que ces opérations représentent généralement environ 90% du temps de calcul et de l'énergie totale nécessaires pour exécuter une inférence ou un apprentissage d'un modèle CNN. Ces conclusions soulignent l'importance critique de l'optimisation des opérations de convolution pour améliorer l'efficacité des CNN, tant du point de vue des performances que de la consommation énergétique.

La couche de convolution constitue un élément central des CNNs, utilisant des filtres apprenants pour identifier les caractéristiques essentielles présentes dans les données d'entrée, telles que les images. Cette opération de convolution implique une multiplication linéaire tridimensionnelle entre un noyau ou filtre (kernel) et une fenêtre de l'image d'entrée. Le noyau est appliqué à différentes positions de l'image, avec une fenêtre glissante a chaque fois d'un pas de stride dans les dimensions de largeur et de hauteur, générant ainsi des cartes de caractéristiques tridimensionnelles, également appelées cartes carac-

téristiques ou en Anglais **"Feature Maps"** (FMs), qui seront utilisées par les couches suivantes du réseau.

Dans ce processus, les paramètres Nif et Nof désignent respectivement le nombre des cartes caractéristiques d'entrée "Input Feature Maps" (IFMs) et de cartes caractéristiques de sortie "Output Feature Maps" (OFMs). Les dimensions des IFMs sont représentées par Nix et Niy, tandis que celles des OFMs sont représentées par Nox et Noy. En outre, Nk représente le nombre de filtres de taille XY, jouant un rôle crucial dans la détermination des caractéristiques extraites par la couche de convolution.

Cette phase de convolution revêt une importance particulière dans la reconnaissance et l'identification de motifs dans les données, et elle est souvent suivie de différentes couches, telles que des couches de pooling ou des couches entièrement connectées, permettant ainsi d'affiner les caractéristiques extraites pour des tâches spécifiques, comme la classification d'images ou la détection d'objets. La Figure 2.11 illustre le processus de convolution dans un réseau neuronal.

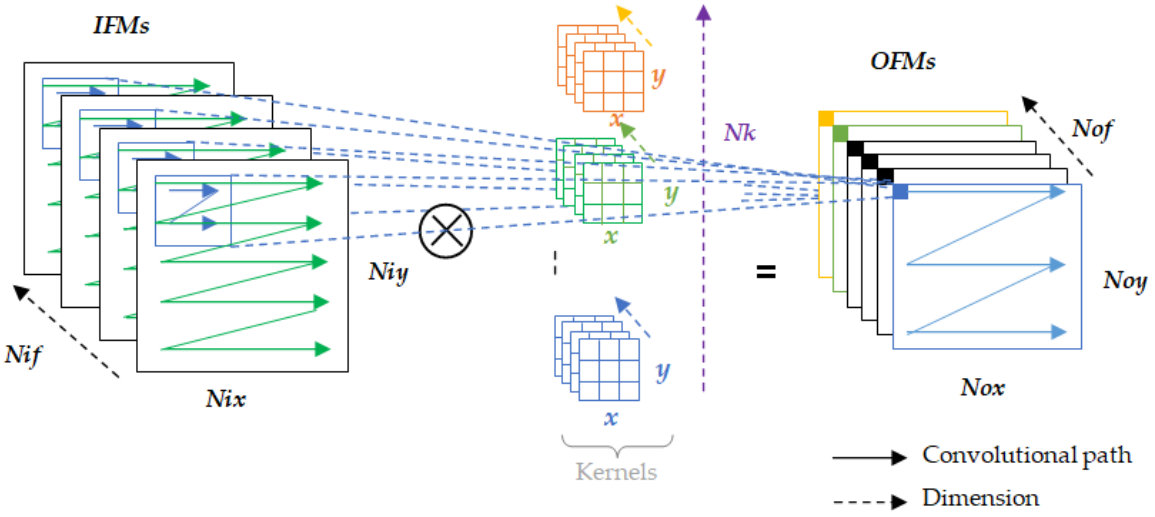


Figure 2.11: Opération du calcul de convolution multi-kernel et multi-entrees.

Le pas de glissement ou **"Stride"** se réfère à la distance entre deux positions successives lors de l'application d'un filtre de convolution sur une image d'entrée. Un Stride plus important accélère la réduction de la taille spatiale, favorisant l'extraction de caractéristiques générales, tandis qu'un Stride plus petit conserve les détails locaux. Le choix du Stride dépend de la tâche, de la complexité des données et des préférences en termes de compromis entre précision et efficacité computationnelle. la Figure 2.12 illustre un Stride

de 1 appliqué lors de l'opération de convolution.

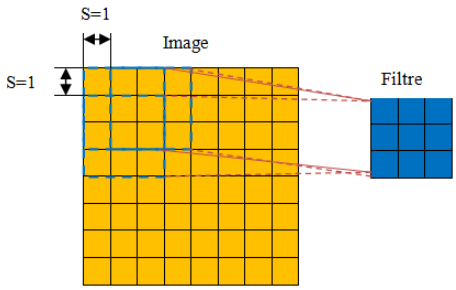


Figure 2.12: Un stride de 1 appliqué sur une image d'entrée.

Ainsi, les dimensions des caractéristiques de sortie dans le contexte convolution dépendent donc des dimensions des caractéristiques d'entrée ainsi que du Stride utilisé et sont données par les expressions 2.5 et 2.6

$$Nix = (Nox - 1)S + Nkx \tag{2.5}$$

$$Niy = (Noy - 1)S + Nky \tag{2.6}$$

Il convient de noter que, dans certaines situations, la technique de remplissage ou (*"Padding"*) est employée, impliquant l'ajout de pixels supplémentaires autour de l'image avant d'appliquer la convolution. Cette méthode a pour objectif de préserver les informations situées dans les coins et sur les bords de l'image, de prévenir une réduction excessive de la taille spatiale, et de garantir une application uniforme des filtres sur toute l'étendue de l'image. L'opération de "Padding" est effectuée de deux manières, soit en ajoutant des zéros (padding zéro) soit en répliquant les valeurs des pixels de bord (padding répétition). La Figure 2.11 illustre un "padding" de 2 sur les bords d'une image de taille 28x28.

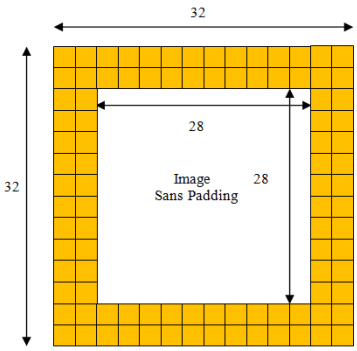


Figure 2.13: Paddind de 2 sur une image de 28x28.

L'opération de convolution entre les filtres de la couche de convolution et la carte des caractéristiques d'entrée est soigneusement orchestrée à travers une structure organisée en quatre boucles imbriquées, offrant ainsi une organisation méthodique. Cette organisation commence avec la première boucle, qui initie l'opération en combinant les valeurs des filtres avec celles des fenêtres correspondantes dans les cartes de caractéristiques d'entrée. Cet échange marque le début du processus d'interaction entre les filtres et les caractéristiques d'entrée.

La deuxième boucle prend ensuite le relais, gérant la traversée de la dimension de profondeur, notée Nif (Number of Input Feature), des "Input Feature Maps" (IFMs). Cette étape assure que chaque filtre est appliqué à toutes les profondeurs de l'entrée, permettant ainsi une analyse exhaustive des différentes caractéristiques présentes dans les IFMs (Input Feature Maps).

La troisième boucle s'attelle à l'exploration systématique des dimensions spatiales Nix et Niy des "Feature Maps" (FMs). Ces dimensions définissent la disposition des caractéristiques spatiales dans la carte des caractéristiques, et cette exploration assure que chaque région de la carte est traitée de manière exhaustive.

Enfin, la quatrième boucle conclut le processus en parcourant de manière exhaustive la dimension Nof (Number of Output Feature) correspondant aux (Output Feature Maps) (OFMs). Cette phase finalise la convolution en générant les différentes cartes de caractéristiques de sortie, où chaque filtre produit sa propre carte de sortie. Ce processus séquentiel et structuré permet une transformation efficace des caractéristiques d'entrée en représentations significatives dans les OFMs.

L'ensemble du processus de convolution est formellement exprimé dans l'Équation 2.7 et structuré dans l'Algorithme 2.1. Cette organisation en boucles imbriquées permet une mise en œuvre efficace de la convolution, assurant une couverture complète des interactions entre filtres, caractéristiques d'entrée et cartes de sortie.

$$Sortie_l(no,x,y) = \sum_{n_i=1}^{Nif}\sum_{j=1}^{Kx}\sum_{i=1}^{Ky}(Sortie_{L-1}(ni, x{*}S{+}kx, y{*}S{+}ky)){*}Poids(ni,no,kx,ky){+}Biais(no) \tag{2.7}$$

Où, S est le pas de glissement , x $\in \{1,2,...,Nox\}, y \in \{1,2,...,Noy\}, no \in \{1,2,...,Nof\}, L \in \{1,2,...,CONVs\}, et \#\text{CONVs}$ est le nombre de couches de convolution.

Algorithme 2.1 : Algorithme d'une operation de convolution Tridimentionnelle

Data : $NiF \leftarrow \#Entres$
Data : $Poids \leftarrow \#Kernels$
Data : $Bias \leftarrow \#Bias$
Data : $Stride \leftarrow S$
NoF : Nombre Images Sorties
Nix : Dimenssion X Entrées
Niy : Dimenssion Y Entrées
Nox : Dimenssion X Sorties
Noy : Dimenssion Y Sorties
Kx : Dimenssion X Kernels
Ky : Dimenssion Y Kernels
Resultat : $Sortie_L \leftarrow (Sortie_{L-1} + Biais)$
while $no \leq NoF$ **do**
  $y = 1$ **while** $y \leq Noy$ **do**
    $x = 1$;
    while $x \leq Nox$ **do**
      $ni = 1$;
      while $ni \leq NiF$ **do**
        $j = 1$;
        while $j \leq Ky$ **do**
          $i = 1$;
          while $i \leq Kx$ **do**
            $Sortie_L(no, x, y) \leftarrow Sortie_{L-1}(ni, x + i, y + j)$
            $\times Poids_{l-1}(ni, no, i, j)$;
            $i \leftarrow i + 1$;
          end
          $j \leftarrow j + 1$;
        end
        $ni \leftarrow ni + 1$;
      end
      $\text{Sortie}_L(no, x, y) = Sortie_{L-1}(no, x, y) + \text{Bias(no)}$
      $x \leftarrow x + S$;
    end
    $y \leftarrow y + S$;
  end
  $no \leftarrow no + 1$;
end

Ainsi, l'implémentation matérielle des opérations associées à la couche de convolution nécessite une attention particulière portée à l'optimisation minutieuse des processus d'accès, de structuration et de manipulation des données. Il est donc impératif de concevoir une architecture matérielle qui vise à maximiser les performances tout en réduisant au minimum la complexité inhérente.

En ce qui concerne la réduction de l'impact des accès à la mémoire, deux stratégies d'optimisation sont généralement envisagées : la première consiste à minimiser les accès à la mémoire externe DDR3 ainsi qu'aux mémoires tampons intégrées, tandis que la deuxième consiste à partitionner les matrices des données et des poids sur plusieurs blocs mémoires afin de permettre des accès multiples, ce qui améliore la latence et le débit.

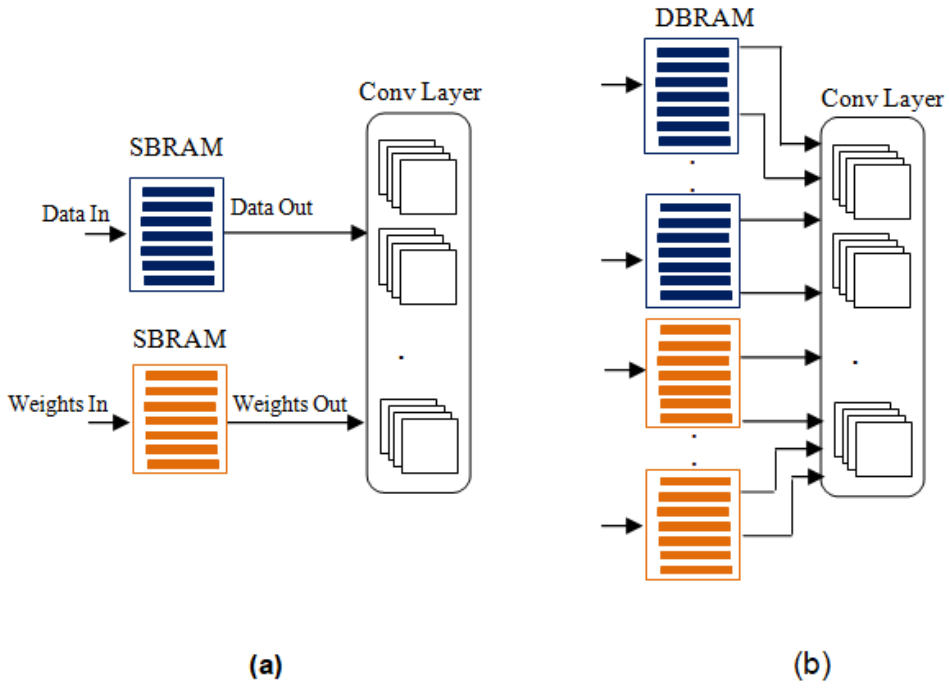


Figure 2.14: (a) Avant le partitionnement, (b) Après le partitionnement.

Quant au traitement des données, il est courant de recourir à deux stratégies d'optimisation distinctes. Premièrement, le fractionnement des fonctions en étapes distinctes, connu sous le nom de pipelining, favorise le chevauchement des opérations, ce qui a pour effet d'améliorer les performances globales du système. La Figure 2.16 illustre un exemple de pipelining de traitement de trois fonctions, montrant que le processus de lecture, de traitement et d'écriture se fait en quatre cycles d'horloge au lieu de neuf sans le pipelining.

.

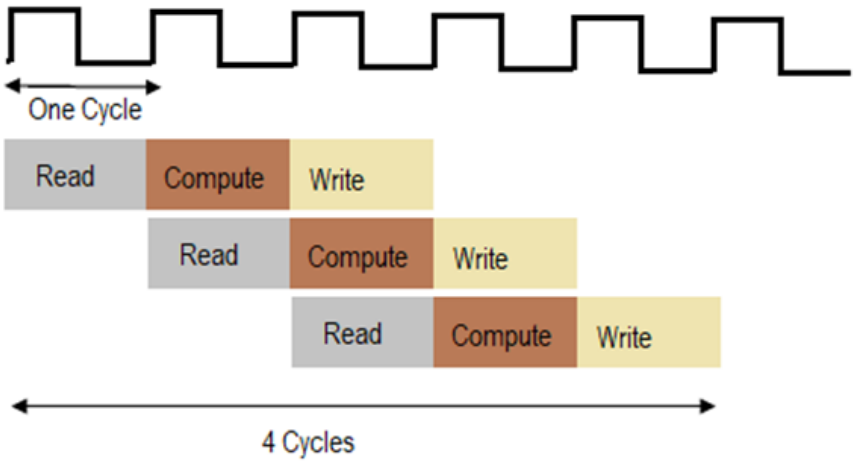


Figure 2.15: Pipelining des fonctions.

En outre, conjointement avec le pipelining, la parallélisation joue un rôle crucial dans l'accélération du traitement des opérations. Cette technique implique l'exécution simultanée de plusieurs opérations, généralement réalisée en déroulant **"Unrolling"** des boucles. Cette optimisation tire pleinement parti des ressources matérielles disponibles, ce qui entraîne une accélération notable du traitement des données, bien que cela nécessite une consommation supplémentaire de ressources et d'énergie.

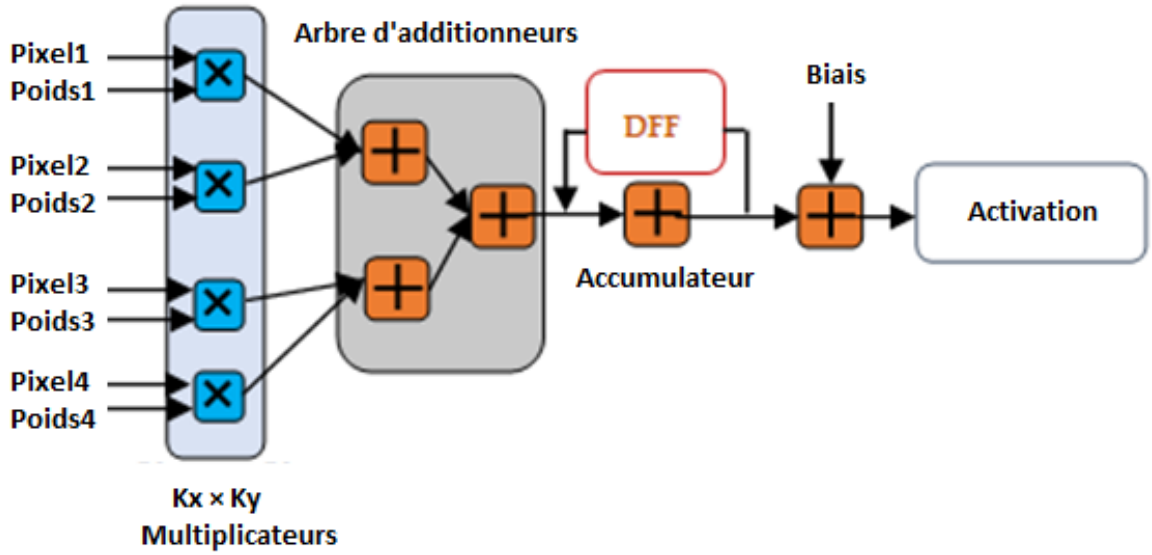


Figure 2.16: Parallélisation des opérations dans une boucle.

Dans ce contexte contraignant, l'implémentation matérielle des CNNs sur des plateformes en vue d'une accélération se révèle être un exercice délicat. Il nécessite de trouver

un juste équilibre entre les performances désirées et les contraintes imposées par les ressources matérielles disponibles sur la plateforme hôte.

2.8 Conclusion

Dans ce chapitre, notre analyse s'est penchée sur deux éléments fondamentaux inhérents au contexte des CNNs. Tout d'abord, une exploration des concepts caractéristiques des réseaux CNNs a été initiée, offrant ainsi une vue d'ensemble détaillée de leur architecture et de leur mode de fonctionnement. Cette démarche a permis d'appréhender la complexité des mécanismes sous-jacents à ces réseaux, soulignant ainsi leur importance dans diverses applications de traitement d'images.

Ensuite, une étude approfondie des principes fondamentaux de l'optimisation a été entreprise, mettant en lumière les principales techniques visant à rehausser les performances des CNNs. Parmi ces techniques, le pipelining des fonctions, le parallélisme des opérations de boucles, et le partitionnement des matrices ont été examinés en détail. Ces méthodes, souvent nécessaires pour gérer l'énorme quantité de données et la complexité des opérations impliquées dans les calculs des CNN.

Dans le prochain chapitre, nous explorerons davantage les travaux connexes et les contributions importantes dans ce domaine. Cela nous aidera à placer notre étude dans le cadre plus large de la recherche en apprentissage automatique et à mettre en lumière les avancées significatives qui ont façonné l'évolution des CNN et de leurs applications.

Chapitre 3

Revue de la Littérature scientifique et Travaux Connexes

3.1 Introduction

Dans ce chapitre, nous entreprendrons une une exploration détaillée de l'état actuel des travaux de recherche sur les CNNs, ainsi que ceux connexes a nos travaux de recherche . Notre objectif est de passer en revue de manière approfondie les études les plus pertinentes et les progrès significatifs réalisés dans ce domaine. Nous cherchons à mettre en évidence les contributions marquantes, les techniques novatrices et les architectures émergentes qui façonnent l'évolution des CNNs.

Avant d'explorer les recherches concernant l'accélération des CNNs sur des plates-formes matérielles, il est important de commencer par examiner les études portant sur les domaines d'application des CNNs. Cette approche est essentielle pour situer notre analyse dans le contexte plus vaste des utilisations et des applications des CNNs.

Dans un second temps, nous proposerons un schéma synthétique récapitulant les axes de recherche principaux dans le domaine des CNNs. Ce schéma offrira une vue d'ensemble claire des diverses orientations explorées par la communauté scientifique, contribuant ainsi à établir un cadre conceptuel détaillé pour ces axes de recherche.

Par la suite, nous entreprendrons une analyse approfondie des progrès significatifs et des perspectives dans chacun des axes de recherche identifiés. Cette démarche consistera à mettre en lumière les travaux les plus influents, les méthodes novatrices et les développements récents ayant contribué à l'évolution des CNN.

3.2 Domaines d'application des CNNs

Les CNNs et les réseaux neuronaux profonds ont profondément transformé notre approche de la résolution de problèmes dans divers domaines. Leur utilisation répandue a permis de relever des défis pour lesquels la connaissance explicite est limitée et où des informations implicites sont présentes dans les données brutes. Cette avancée a ouvert de nouvelles perspectives dans des domaines variés tels que les sciences et les affaires, en offrant des solutions auparavant inenvisageables malgré les efforts antérieurs de la communauté de l'intelligence artificielle. L'aptitude remarquable de l'apprentissage profond à identifier les structures complexes au sein de données de haute dimension a été essentielle dans ces progrès significatifs.

Effectivement, les réseaux de neurones convolutionnels (CNN) ont désormais acquis une place prépondérante dans une vaste palette de domaines, témoignant ainsi de leur adaptabilité et de leur impact transversal sur différentes disciplines. Leur utilisation se manifeste dans divers contextes, notamment :

Dans la reconnaissance de caractères [11], où les CNNs ont été largement exploités pour interpréter et identifier des caractères dans des documents numérisés, contribuant ainsi à des applications telles que la reconnaissance optique de caractères (OCR).

Dans la reconnaissance de gestes [27], les CNNs ont été employés pour analyser et comprendre les mouvements du corps humain, ouvrant la voie à des applications allant de la commande gestuelle à la réalité virtuelle.

Dans la reconnaissance vocale, à travers des assistants virtuels tels que Google Now et Siri [28, 29], les CNNs jouent un rôle essentiel dans la transcription et la compréhension du langage parlé, facilitant ainsi l'interaction homme-machine.

Dans le traitement de documents [30, 31], les CNNs sont utilisés pour extraire des informations significatives à partir de documents textuels ou visuels, facilitant leur analyse et leur exploitation ultérieure.

Dans le traitement du langage naturel [32], les CNNs sont employés pour modéliser et comprendre le langage humain, contribuant ainsi à des tâches telles que la traduction automatique, la génération de texte et l'analyse de sentiment.

Dans la classification vidéo [6, 33, 34], les CNNs sont utilisés pour identifier et catégoriser des séquences vidéo, permettant ainsi la surveillance automatique, la reconnaissance d'actions humaines et la détection d'anomalies.

Dans la classification d'images [35–37], les CNNs sont largement exploités pour la

classification et la caractérisation d'images dans des domaines aussi variés que la médecine, l'astronomie et l'analyse de données satellitaires.

Dans la détection et l'identification faciales [38–40], les CNNs sont utilisés pour identifier des visages dans des images ou des vidéos, facilitant ainsi des applications telles que la reconnaissance faciale pour la sécurité ou la surveillance.

Dans la navigation robotique [41–43], les CNNs sont employés pour la perception de l'environnement, permettant aux robots de cartographier leur environnement, de planifier des trajectoires et d'éviter les obstacles.

Dans la surveillance en temps réel de multiples objets [44], les CNNs sont utilisés pour suivre et identifier plusieurs objets dans des scènes vidéo complexes, facilitant ainsi des applications telles que la surveillance de la circulation ou la détection d'événements anormaux.

Dans les prévisions financières [45], les CNNs sont employés pour analyser des données financières et identifier des tendances, contribuant ainsi à la prise de décisions en matière d'investissement et de gestion des risques.

Dans les systèmes de diagnostic médical [46–48], les CNNs sont utilisés pour analyser des images médicales et diagnostiquer des conditions médicales, facilitant ainsi le dépistage précoce et la prise en charge des patients.

Dans le domaine de l'aérospatiale et de la défense, les réseaux de neurones convolutionnels (CNNs) sont souvent employés pour l'identification d'objets depuis des satellites et l'évaluation de zones sécurisées ou non sécurisées. Des références pertinentes pour ces applications pourraient inclure des travaux tels que ceux de Liu et al. [49] pour l'identification d'objets depuis des images satellites, ainsi que les recherches de zhang et al. [50] sur l'évaluation de la sécurité des zones à partir de données d'imagerie aérienne.

Les exemples mentionnés précédemment illustrent la polyvalence et l'impact significatif des CNNs dans une variété de domaines, ouvrant ainsi la voie à de nouvelles possibilités et à des avancées majeures dans la résolution de problèmes complexes.

3.3 Revue des travaux relatifs à l'implémentation et à l'accélération des CNNs

Une analyse approfondie des travaux de recherche existants dans la littérature révèle une variété d'approches et de techniques pour l'implémentation des CNNs. Ces recherches

ont été menées dans des contextes divers, couvrant la plateforme accueillant ces CNNs, ainsi que les méthodologies et approches de mise en œuvre. Cette diversité souligne l'intérêt spécifique et la polyvalence architecturale des CNNs dans une multitude de champs d'application.

Dans le cadre de notre étude, nous avons entrepris une classification des approches présentées dans la littérature en deux grandes catégories distinctes. La première catégorie se concentre sur les aspects matériels, examinant les diverses stratégies d'intégration et de déploiement des modèles CNNs sur une gamme variée d'architectures matérielles. La seconde catégorie met en avant l'optimisation des performances à travers des approches algorithmiques et mathématiques, visant à améliorer à la fois les capacités de calcul et la précision des CNNs.

Cette distinction nous a permis de diriger notre attention vers les domaines qui correspondent le mieux à nos propres travaux, à savoir l'implémentation des CNNs sur des plateformes FPGA, ainsi qu'aux diverses optimisations des modèles CNNs et à la génération. En identifiant ces deux axes de recherche, nous avons pu cibler les approches les plus pertinentes pour notre étude et concentrer nos efforts sur les domaines où nous pouvons apporter une contribution significative. La Figure 3.1 présente les principales approches émergeant de la deuxième catégorie, offrant ainsi une vue d'ensemble des travaux proposés dans ce domaine dynamique et en constante évolution.

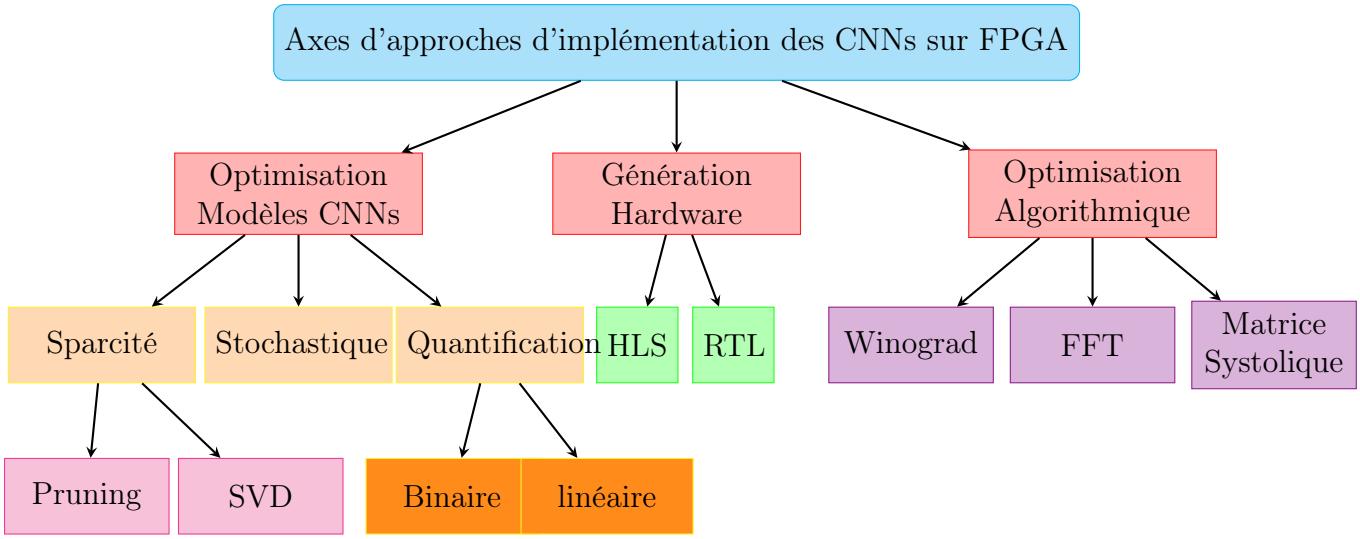


Figure 3.1: Organigramme des différentes catégories d'approches proposées dans la littérature.

3.3.1 Analyse des Approches des CNNs sur les Plateformes Matérielles : Revue de la Littérature

Les exigences matérielles et computationnelles associées à la mise en œuvre des CNNs constituent des contraintes majeures auxquelles les chercheurs sont confrontés. Ces réseaux de neurones convolutifs exigent des ressources de calcul importantes pour effectuer des opérations de convolution complexes sur de grandes quantités de données. De plus, la gestion efficace du stockage des données est également cruciale, car les CNNs traitent souvent des ensembles de données volumineux. Face à ces défis, les chercheurs s'efforcent de proposer des solutions adaptées aux plates-formes matérielles disponibles. Ces solutions visent à optimiser l'utilisation des ressources de calcul et à minimiser les temps d'accès aux données, tout en maintenant des performances élevées pour les applications de traitement d'image et de vision par ordinateur.

Ainsi, de nombreuses implémentations ont été proposées pour différentes plate-formes, notamment l'unité centrale de traitement (CPU) *(Central Processing Unit)* [51], l'unité de traitement graphique (GPU) *(Graphical Processing Unit)* [52], les circuits intégrés spécifiques à une application (ASIC). une revue des différents travaux relatifs a cette dernière plateforme est proposée dans [53], ainsi que les matrices de portes programmables sur site (FPGA) [54–56].

Une étude exhaustive des diverses implémentations basées sur les FPGAs est présentée dans l'article de Mittal et al. [57]. Les FPGAs offrent plusieurs avantages distincts par rapport à d'autres dispositifs, et il est important de les souligner. Tout d'abord, les FPGAs sont réputés pour leur remarquable efficacité énergétique. Contrairement à d'autres technologies, ils peuvent être configurés de manière à utiliser uniquement les ressources nécessaires pour une tâche spécifique, ce qui permet d'optimiser leur consommation d'énergie. De plus, les FPGAs offrent une grande flexibilité grâce à leurs ressources internes reconfigurables. Cette caractéristique permet aux développeurs d'adapter les FPGAs à diverses applications en reconfigurant leur structure interne, ce qui garantit des performances optimales pour un large éventail de tâches. En combinant ces avantages, les FPGAs se révèlent être une solution attrayante pour de nombreuses applications, offrant à la fois une efficacité énergétique et une flexibilité supérieures.

3.3.2 Approches d'implémentation des CNNs sur FPGAs

Outre l'assurance de l'adaptabilité de la plateforme hôte en termes de ressources et de performances, l'implémentation des CNNs soulève d'autres considérations cruciales. Parmi celles-ci figurent notamment les stratégies d'optimisation spécifiques à l'architecture et aux environnements de développement employés. Cette section se focalise donc sur les principaux axes identifiés dans la figure 3.1.

3.3.2.1 Environnement de génération Hardware

La mise en œuvre matérielle des CNNs sur des plateformes FPGA exige une double expertise, à la fois dans l'architecture du modèle et dans les caractéristiques de la plateforme elle-même. Cela signifie que pour déployer efficacement un CNN sur un FPGA, il est nécessaire de comprendre à la fois la structure du réseau neuronal lui-même, y compris les détails de ses couches, et les spécificités du FPGA, telles que les ressources matérielles disponibles, la connectivité et les contraintes de performance. Dans ce contexte, deux approches sont couramment utilisées pour implementer les CNNs sur des FPGAs, à savoir le (RTL) *(Register Transfer Level)* et le (HLS) *(High Level Synthesis)* ainsi que ses derivées (OpenCl) *(Open Computer Language)* et (OpenCV) *(Open Computer Vision)*.

La mise en œuvre de circuits électroniques repose fréquemment sur l'utilisation de langages de description matérielle tels que (VHDL) *(Very high speed integrated circuits Hardware Description Language)* ou (Verilog HDL) *(Verification logic Hardware Description Language)*. Ces langages offrent aux concepteurs la possibilité de décrire de manière abstraite le comportement et la structure des circuits électroniques, ce qui s'avère extrêmement précieux pour simplifier la conception, effectuer des simulations, et valider les conceptions avant de passer à la phase de fabrication physique. Ainsi, l'implémentation des CNNs sur des plateformes FPGA ne fait pas exception à cette règle bien établie. Elle implique également le recours à ces langages de description matérielle pour définir de manière précise et détaillée les opérations matérielles nécessaires à l'exécution des CNN sur le FPGA, notamment les opérations de convolution, le transfert des données, la gestion des mémoires, et bien d'autres aspects critiques de l'architecture matérielle. Cela permet d'adapter le FPGA de manière optimale aux besoins spécifiques de l'application.

Dans la littérature, les travaux novateurs concernant l'implémentation des CNNs sur des plateformes matérielles ont principalement reposé sur l'approche RTL [8, 58, 59].

Au cours de la dernière décennie, l'émergence du HLS, d'OpenCL et d'OpenCV a

coïncidé avec une demande croissante pour des approches novatrices dans le domaine de l'implémentation des réseaux neuronaux, dans le but d'accélérer leurs performances. Ces avancées ont révolutionné l'approche des concepteurs vis-à-vis du développement d'applications sur des plateformes matérielles. En conséquence, la littérature a présenté plusieurs approches exploitant ces outils d'implémentation pour répondre à cette demande croissante de performances [60–62]. La Figure 3.2 illustre les étapes clés de la transition du HLS vers le RTL.

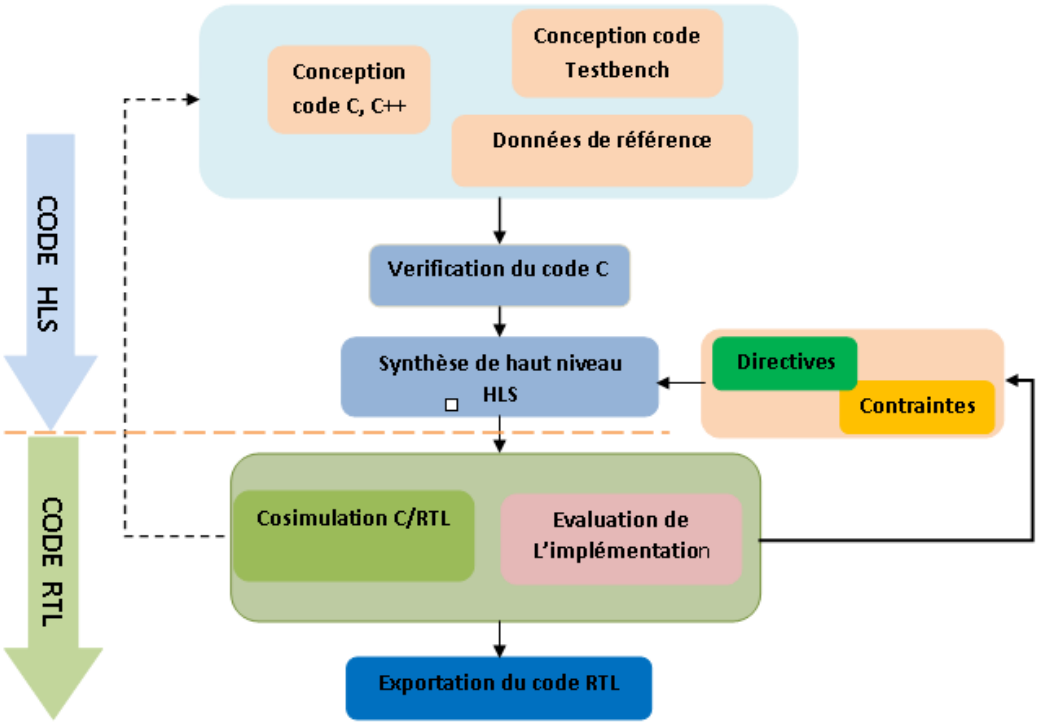


Figure 3.2: Organigramme d'implémentation Avec le HLS.

3.3.2.2 Optimisation des Modèle CNNs

1. **Optimisation du modèle à base de Quantification :** La quantification des CNNs est une technique cruciale pour réduire les exigences en termes de mémoire et de puissance de calcul des réseaux neuronaux profonds. Plusieurs approches ont été développées à cet effet, visant à quantifier les poids, les activations et les opérations des réseaux de neurones. Ces méthodes permettent une exécution plus efficace sur des plateformes à ressources limitées telles que les FPGA ou les dispositifs mobiles. Des travaux, tels que ceux menés par Jacob et al. (2018) [63], ont

exploré la quantification des poids et des activations en utilisant des schémas de quantification entière. De même, Zhou et al. (2017) [64] ont proposé une méthode d'incrémentation progressive pour atteindre des réseaux sans perte avec des poids de faible précision. En parallèle, les travaux de Zhu et al. (2016) [65] ont introduit la quantification ternaire, une technique où les poids sont quantifiés en seulement trois valeurs distinctes.

D'autres approches, comme XNOR-Net [66, 67] , ont exploré la quantification binaire pour les poids et les activations, permettant ainsi une réduction significative des exigences de stockage et de calcul. En outre, des méthodes de compression telles que Deep Compression [64, 68] ont combiné la quantification avec d'autres techniques telles que l'élagage et le codage Huffman pour obtenir des réseaux plus compacts tout en préservant les performances.

Enfin, des recherches comme celles de Lin et al. (2016) [69] ont examiné la quantification en virgule fixe pour les CNNs, fournissant des méthodes plus précises de quantification adaptées aux contraintes matérielles spécifiques. Ces diverses approches offrent des solutions flexibles pour adapter les CNNs aux différentes plates-formes matérielles et aux exigences d'application, permettant ainsi une exécution efficace des réseaux neuronaux profonds dans des environnements à ressources limitées.

2. **Optimisation stochastique :** L'optimisation stochastique des modèles CNN revêt une importance capitale pour améliorer l'efficacité de leur entraînement. Parmi les méthodes d'optimisation largement utilisées, l'algorithme Adam, présenté par Kingma et Ba (2014) [70], se démarque grâce à ses mises à jour adaptatives des taux d'apprentissage, offrant ainsi une approche efficace pour l'optimisation stochastique. Les travaux de Reddi et al. (2018) [71] ont enrichi la compréhension de l'algorithme Adam en proposant des ajustements visant à améliorer sa convergence, ce qui s'avère précieux pour son application dans l'entraînement des CNNs. Par ailleurs, des approches classiques telles que la descente de gradient stochastique, comme détaillé par Bottou (2012) [72] dans son étude sur les astuces associées à cette technique, sont également explorées pour leur applicabilité dans l'optimisation des CNNs. RMSprop, présenté par Hinton (2012) [73], est une autre méthode

efficace qui adapte de manière adaptative le taux d'apprentissage pour chaque paramètre du modèle, augmentant ainsi son efficacité. Enfin, les travaux de Glorot et Bengio (2010) [74] ont permis d'identifier et d'analyser les défis inhérents à l'entraînement des réseaux de neurones profonds, offrant ainsi une compréhension approfondie des problèmes d'optimisation rencontrés, ce qui se révèle également pertinent dans le contexte de l'optimisation stochastique des CNNs

3. **optimisation du modèle par l'utilisation de la sparcité** La notion de sparsité dans les CNNs évoque la présence fréquente de nombreux éléments des tenseurs de poids ou d'activation ayant des valeurs nulles. Cette particularité émerge souvent de diverses influences, telles que les régularisations appliquées durant l'entraînement, les méthodes de compression des modèles, ou encore les caractéristiques propres aux données utilisées. Exploiter cette sparsité permet de réduire la complexité et la taille des modèles, d'améliorer l'efficacité computationnelle, et de diminuer les besoins en stockage et en communication. Les retombées positives de la sparsité incluent une baisse significative de la charge de calcul lors des phases d'inférence, une utilisation plus judicieuse des ressources matérielles comme la mémoire et les unités de calcul, ainsi qu'une facilité accrue pour le déploiement sur des appareils aux ressources limitées. Cependant, la gestion appropriée de cette sparsité nécessite des algorithmes adaptés pour l'entraînement et l'inférence, ainsi que des techniques de compression et de régularisation efficaces pour maintenir les performances du modèle malgré la potentialité de perte d'information liée à la sparsité.

   Deux grandes approches sont largement utilisées pour obtenir la sparsité dans les CNNs, le Pruning et la (SVD) *(Singular Value Decomposition)*

   Pruning : Le pruning des CNNs, ou élagage, est une technique largement adoptée visant à réduire la complexité des réseaux neuronaux en supprimant les poids des connexions moins significatives. Son objectif est de compresser le modèle tout en maintenant ses performances. En identifiant et en éliminant les connexions neuronales peu importantes, le réseau devient plus léger et plus efficient, ce qui facilite son déploiement sur des dispositifs avec des ressources limitées et réduit les coûts de calcul.

   La Figure 3.3 présente graphiquement deux types de techniques de pruning, à savoir l'élagage des connexions et l'élagage des neurones.

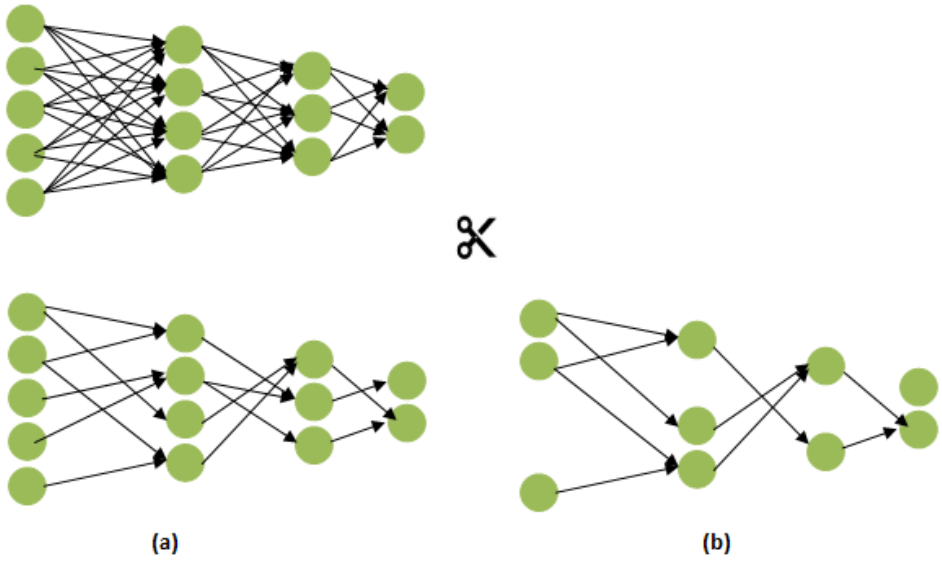


Figure 3.3: Opérations d'élagage ou Pruning : (a) Élagage des connexions, (b) Élagage des neurones..

SVD : Dans le même souci de réduction des paramètres du réseau, on trouve dans la littérature des approches basées sur la décomposition des valeurs singulières (SVD) *(Singular Value Decomposition)* comme décrit dans les travaux de Denton et al. (2014) [75] et Lebedev et al. [76]. La SVD est une technique de factorisation matricielle qui permet de décomposer une matrice en trois matrices plus petites : $\cup$, Σ, et $\vee$, Ce qui permet de réduire la complexité des réseaux profonds en représentant leurs poids de manière plus compacte. Il convient de noter que jusqu'à présent, cette approche a été appliquée uniquement aux couches entièrement connectées et non aux couches convolutives.

3.3.2.3 Optimisations algorithmiques

1. **approche à base de l'algorithme de Winograd :**

   Dans les parties précédentes, nous avons souligné la contrainte de la charge computationnelle liée à l'opération de convolution, en particulier dans des scénarios impliquant de grandes quantités de données d'entrée et des dimensions significatives de filtre. A titre récapitulatif, l'exécution d'une opération de convolution 2D avec un filtre de dimensions r×r sur une image d'entrée de dimensions (L×H×C) requiert (L - r + 1)×(H - r + 1) ×r×r×C multiplications. Pour substantiellement réduire la complexité en termes de multiplications dans la couche de convolution, l'algorithme de Winograd se positionne comme l'une des solutions éprouvées.

L'approche minimale de convolution de Winograd appliquee pour m=2 sorties, utilisant des filtres de taille r×r=3×3 et des dimensions de fenêtre d'entrée de 4×4, notée μF=(m , r)=(2 , 3), permet de réduire le nombre de multiplications jusqu'à (m+r-1)=4 au lieu de m×r=2×3=6 habituellement dans une configuration classique de convolution. cette Approche Winograd est basée sur une série de transformation utilisant des matrices de transformation précalculées. La Figure 3.4 fournit une représentation graphique des opérations de transformation clés impliquées dans l'application de l'approche minimale de Winograd pour accomplir une convolution 2D. Ces opérations détaillées mettent en évidence le processus par lequel la transformée de Winograd permet de réduire le nombre total d'opérations nécessaires, contribuant ainsi à l'amélioration des performances computationnelles de manière significative.

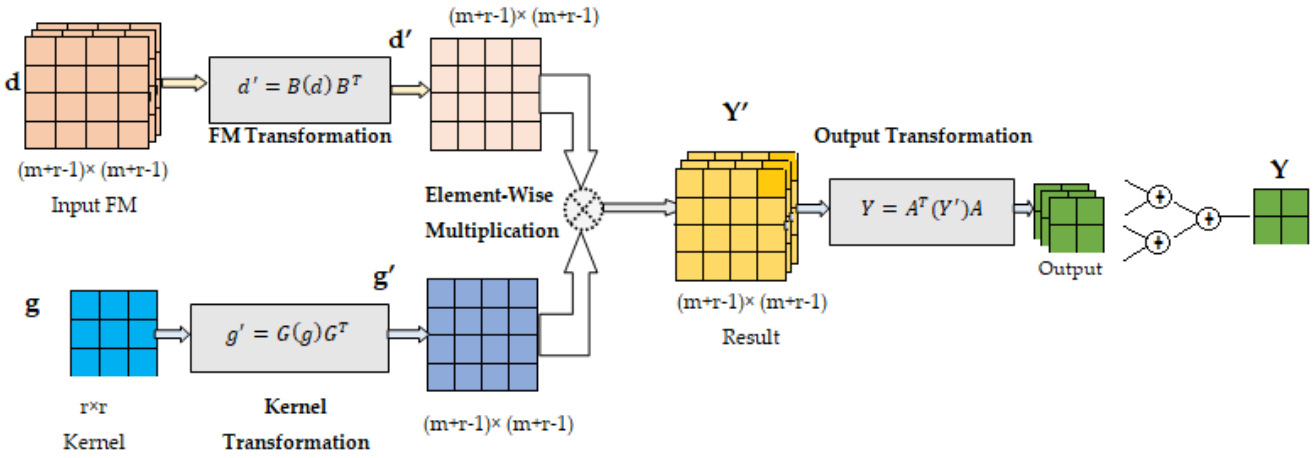


Figure 3.4: Opérations de transformation de Winograd.

Le résultat du calcul de m sorties avec une taille de filtre r, est noté F(m, r). En employant la convolution spatiale pour F(2, 3), cela requiert 2x3=6 multiplications. L'algorithme de Winograd procède au calcul de F(2, 3) selon la méthode suivante, nécessitant ainsi seulement 4 multiplications :

$$\text{d}=\begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}^T \qquad g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T \qquad Y = \begin{bmatrix} y_0 & y_1 \end{bmatrix}^T$$

$$\begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 + m_4 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \tag{3.1}$$

$\mathrm{m}_1, m_2, m_3 et m_4 sont$:

$$\begin{aligned} m_1 &= (d_0 - d_2)g_0 & m_2 &= (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2} \\ m_4 &= (d_1 - d_3)x_2 & m_2 &= (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2} \end{aligned} \tag{3.2}$$

La convolution 1D à l'aide de l'algorithme de Winograd peut être formulée en utilisant les matrices de transformation A, B et G comme suit,

$$Y = A^T[(Gg) \odot (B^T d)] \tag{3.3}$$

Avec ;

$$\mathrm{B}^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \qquad B^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

Dans cette formulation, le symbole $\odot$ représente la multiplication matricielle élément par élément (EWMM) *(Element Wise Matrix Multiplication)* ou Hadamard Product. Dans le cas de l'algorithme Winograd 2D, F (m m,r r), où la taille de la fenêtre de sortie est m×m, la taille du filtre est r×r et la taille de la fenêtre d'entrée est n×n (n = m + r 1). En imbriquant l'algorithme Winograd 1D avec lui-même, l'algorithme Winograd 2D peut être formulé comme suit :

$$Y = A^T[(GgG^T) \odot (B^T dB)]A \tag{3.4}$$

La littérature présente diverses approches d'optimisation des CNNs fondées sur la transformation de Winograd. Par exemple, Liu et al. [77] ont proposé une méthode novatrice nommée "Sparse-Winograd", exploitant à la fois la sparsité des données et la transformation de Winograd. Cette approche vise à réduire le nombre de paramètres grâce à la

sparsité, ce qui entraîne une diminution des ressources de stockage, tandis que la transformation de Winograd est utilisée pour réduire la complexité computationnelle.

2. **FFT :**

L'intégration de la Transformée de Fourier Rapide (FFT) *(Fast Fourier Transform)* dans l'implémentation des CNNs représente une avancée significative pour accélérer les opérations de convolution. La FFT, une technique bien connue en traitement du signal, permet de convertir les signaux temporels en leur représentation fréquentielle, facilitant ainsi les convolutions dans le domaine fréquentiel. Dans le cadre des CNNs, cette approche consiste à transformer les images d'entrée et les filtres de convolution en leur domaine fréquentiel respectif. En exploitant les propriétés de la convolution dans ce domaine, les opérations de convolution deviennent plus rapides grâce à des multiplications simples, une alternative aux convolutions spatiales coûteuses. Cette méthode réduit considérablement le temps de calcul requis, notamment pour les filtres de grande taille, tout en optimisant l'utilisation de la mémoire. Cependant, l'utilisation de la FFT nécessite une gestion minutieuse des artefacts fréquentiels et des effets de bord pour garantir la précision des résultats.

De fait, plusieurs approches d'optimisation des CNNs sont proposées dans la littérature. Ainsi, Ito et al. [78] ont proposé un travail qui présente une méthodologie qui utilise les données de capteurs d'accélération et de gyroscope pour générer des spectrogrammes FFT, qui sont ensuite utilisés comme entrée pour un CNN. L'objectif principal de cette approche est de reconnaître les différentes activités humaines à partir des données des capteurs.

Dans [79] les auteurs explore une approche novatrice pour accélérer les opérations de convolution dans les CNNs en utilisant des convolutions fractionnées basées sur la transformée de Fourier rapide (FFT). La méthode divise la convolution en opérations plus petites, facilitant ainsi l'application de la FFT pour accélérer le calcul.

En outre, le travail de Ravi et al. [80] présente une approche visant à développer une interface cerveau-ordinateur (BCI) *(Brain-Computer Interface)* basée sur les potentiels évoqués visuellement (SSVEP) *(Steady-State Visual Evoked Potentials)* qui soit indépendante de l'utilisateur. Les SSVEP sont des signaux électrophysiologiques du cerveau qui émergent en réponse à une stimulation visuelle périodique, et ils sont couramment utilisés dans les BCI. La méthode proposée repose sur l'utilisation de caractéristiques basées sur la Transformée de Fourier Rapide (FFT) complexe pour extraire des informations per-

tinentes des signaux SSVEP. Ces caractéristiques sont ensuite intégrées comme entrées dans un réseau CNN afin de classifier les intentions de l'utilisateur.

3. **Matrices systoliques** Les matrices systoliques, souvent utilisées dans le contexte des systèmes de calcul parallèle, offrent une méthode efficace pour effectuer des calculs matriciels de manière distribuée et synchrone. Leur architecture parallèle est conçue pour exploiter pleinement le parallélisme des données et des instructions, ce qui permet d'accélérer significativement les opérations de calcul.

   Dans le domaine des réseaux de neurones convolutifs (CNNs), les matrices systoliques sont de plus en plus adoptées pour leur capacité à accélérer les opérations de convolution, qui sont au cœur des calculs effectués par les CNNs. En utilisant des matrices systoliques, les opérations de convolution peuvent être réparties entre plusieurs unités de calcul en parallèle, permettant ainsi d'effectuer des convolutions sur de grandes matrices d'entrée avec une efficacité et une vitesse accrues. Cette approche se révèle particulièrement utile pour les applications nécessitant un traitement rapide des données en temps réel.

   Par ailleurs, Le principe des matrices systoliques dans la convolution repose sur la structure de Processeur Élémentaire (PE) *(Processing Element)* agissant en parallèle au sein d'une matrice spécialement conçue. Chaque PE joue un rôle crucial dans l'accomplissement des opérations de multiplication, d'addition et d'accumulation, et leur synchronisation avec les éléments adjacents permet d'effectuer ces opérations de manière hautement parallèle. Les données d'entrée traversent la matrice systolique, où chaque PE effectue des opérations sur ces données en cours de traitement. Les résultats sont accumulés progressivement à mesure qu'ils traversent la matrice, et le résultat final de la convolution est obtenu à la sortie de la matrice systolique. Cette structure de PE dans la matrice systolique permet de réaliser des calculs continus, sans nécessiter de stockage intermédiaire, contribuant ainsi à l'accélération des opérations de convolution, à la réduction de la latence et à l'amélioration des performances dans les implémentations matérielles des CNNs, en particulier sur les plates-formes FPGA. Dans ce contexte, trois principaux modes de configuration d'implémentation des couches de convolution basées sur des matrices systoliques sont identifiés dans la littérature : les matrices systoliques à poids stationnaires, à sorties stationnaires et à entrées stationnaires. Les matrices systoliques à poids stationnaires préchargent la matrice de poids dans les tampons de poids dans chaque PE correspondant avant d'initier les opérations. La Figure 3.5 illustre une matrice systolique

à poids stationnaires.

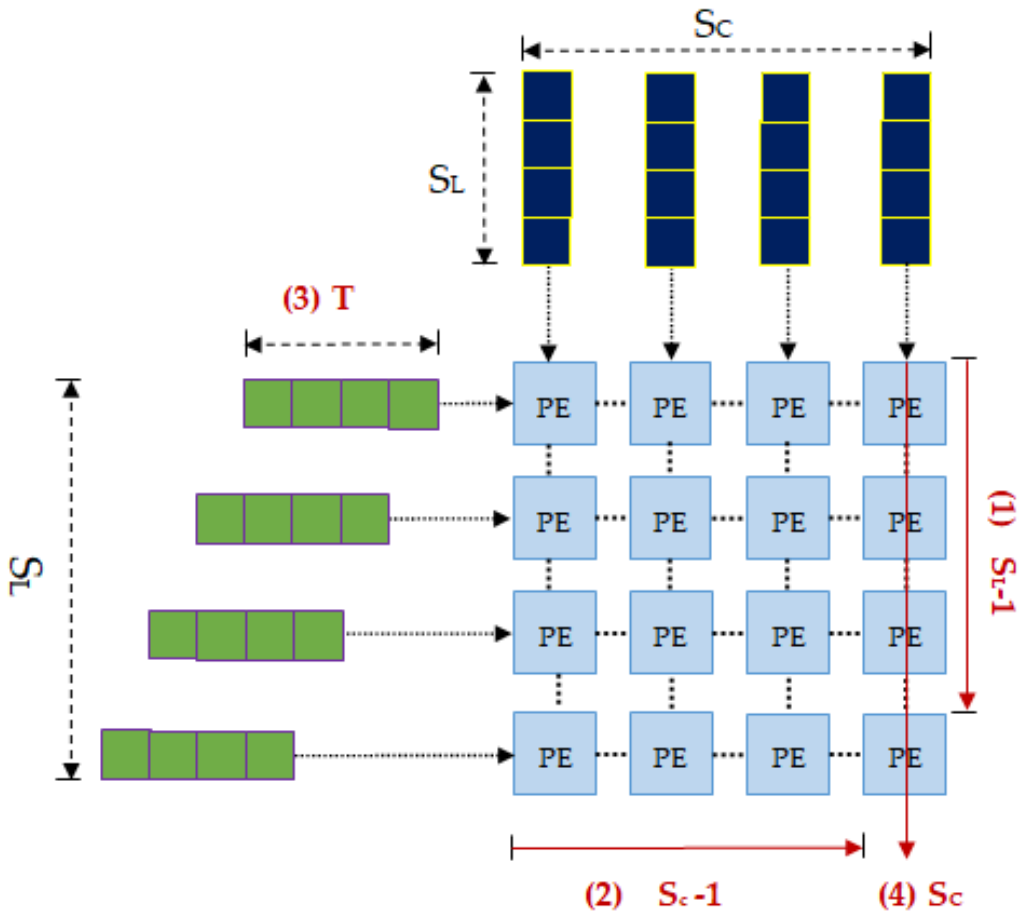


Figure 3.5: Modélisation en mode poids stationnaires.

Les matrices systoliques à entrées stationnaires fonctionnent en chargeant la matrice d'entrée dans les accumulateurs internes des processeurs élémentaires (PE) avant d'effectuer les opérations de convolution. En revanche, les matrices systoliques à sorties stationnaires chargent la matrice de sortie dans les accumulateurs des PE, tandis que les matrices systoliques à poids stationnaires préchargent la matrice de poids dans les tampons de poids avant d'initier les opérations. Cette diversité de configurations offre une flexibilité dans le choix de l'approche la mieux adaptée à des scénarios spécifiques, permettant ainsi d'optimiser les performances des systèmes basés sur des matrices systoliques pour les couches de convolution dans les réseaux de neurones convolutifs (CNN). La Figure 3.6 présente la configuration en mode sorties stationnaires.

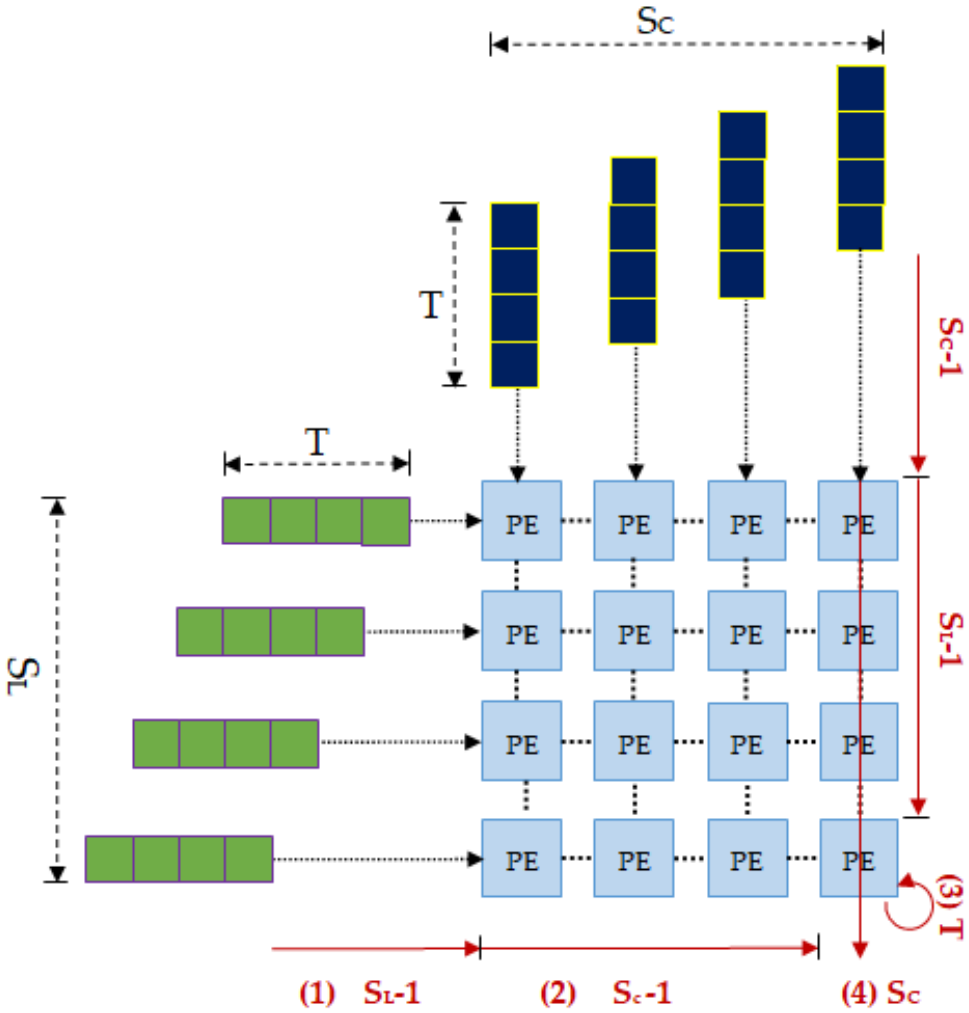


Figure 3.6: Modélisation en mode sorties stationnaires.

de nombreux chercheurs ont exploré et proposé des approches novatrices basées sur l'utilisation de matrices systoliques pour accélérer les opérations de convolution. Par exemple, le travail de Chen et al. [5] a introduit Eyeriss, une architecture de puce accélératrice spécialement conçue pour les CNN, exploitant des matrices systoliques pour une efficacité énergétique élevée et des performances optimales sur une variété de tâches de vision par ordinateur. De même, le projet de Ren et al. [7] a présenté DeepMon, une architecture FPGA basée sur des matrices systoliques pour l'inférence de CNN, offrant une accélération significative des calculs de convolution tout en réduisant la consommation d'énergie. En outre, le travail de Chen et al. [81] a décrit Eyeriss V2, une version améliorée de l'architecture Eyeriss, avec des améliorations dans la gestion de la mémoire et une efficacité énergétique encore plus élevée. Par ailleurs, Zhang et al. [82] ont proposé

une approche utilisant des matrices systoliques pour la conception d'une architecture de réseau de neurones à faible latence. Enfin, l'étude de Kim et al. [83] a exploré l'utilisation de matrices systoliques dans le contexte de l'accélération matérielle pour les applications de vision par ordinateur, offrant ainsi des perspectives intéressantes pour l'optimisation des performances des CNNs.

3.4 Conclusion

Dans cette exploration approfondie des recherches sur les CNNs et des travaux associés à notre domaine d'étude, nous avons examiné les contributions significatives ainsi que les avancées réalisées dans ce domaine dynamique. Notre objectif était de mettre en évidence les travaux les plus influents, les techniques novatrices et les architectures émergentes qui façonnent l'évolution des CNNs.

Avant d'aborder notre analyse sur l'accélération des CNNs sur des plates-formes matérielles, nous avons jugé essentiel d'examiner les recherches relatives aux domaines d'application des CNNs. Cette démarche visait à contextualiser notre examen dans le large spectre des utilisations et des applications des CNNs.

Par la suite, nous avons présenté un schéma synthétique récapitulant les principaux axes de recherche dans le domaine des CNNs. Ce schéma a permis une vue d'ensemble claire des différentes orientations explorées par la communauté scientifique, contribuant ainsi à établir un cadre conceptuel détaillé pour ces axes de recherche.

Dans le prochain chapitre, nous aborderons une approche spécifique d'implémentation des CNNs basée sur le High-Level Synthesis (HLS). Nous explorerons les techniques et les méthodes utilisées pour concevoir et accélérer les CNNs en tirant parti des avantages offerts par le HLS. Cette approche promet de contribuer de manière significative à la mise en œuvre efficace des CNNs sur des plates-formes matérielles.

Chapitre 4

Implémentation Utilisant la Synthèse de Haut Niveau : Approche et Méthodologie

4.1 Introduction

Nos travaux de recherche se concentrent sur l'optimisation de l'implémentation de l'inférence sur une plateforme FPGA. Toutefois, il est crucial de souligner que la validation empirique des méthodes avancées repose sur l'utilisation de données réelles, intégrant les poids et les biais réels des différents filtres (ou noyaux) du modèle CNN adopté. Nous avons ainsi adopté une approche exhaustive, englobant à la fois la phase d'apprentissage, durant laquelle le modèle acquiert les caractéristiques du jeu de données, et la phase d'inférence, pendant laquelle il est mis en œuvre pour réaliser des prédictions sur de nouvelles données.

Cependant, une attention particulière est accordée à la phase d'inférence dans notre approche. C'est à ce stade que le modèle démontre sa valeur en prenant des décisions en temps réel, avec pour objectif principal l'amélioration de ses performances. Nous considérons que la validation de notre implémentation à l'aide de données réelles est indispensable pour confirmer la fiabilité et l'efficacité de notre approche.

Pour ce faire, nous avons opté pour une approche combinant différents environnements d'implémentation. Pendant la phase d'apprentissage, nous avons utilisé le langage de programmation Python, largement répandu dans le domaine de l'apprentissage automatique, pour entraîner notre modèle CNN sur les données d'entraînement. Pour la phase d'inférence, nous avons utilisé le HLS (voir la section 3.3.2.1), un environnement d'implé-

mentation qui permet de traduire efficacement des algorithmes de haut niveau en circuits matériels, spécifiquement conçu pour une implémentation accélérée des CNN sur FPGA.

4.2 Synthèse de l'Approche Méthodologique Adoptée

la Figure 4.1 offre une vue d'ensemble à un niveau élevé de l'approche proposée et met en évidence ses composantes clés, notamment les phases d'apprentissage et d'implémentation matérielle du modèle CNN. Chacune des deux phases est divisée en une série de sous-étapes, elles-mêmes subdivisées en tâches qui sont détaillées dans les sections suivantes.

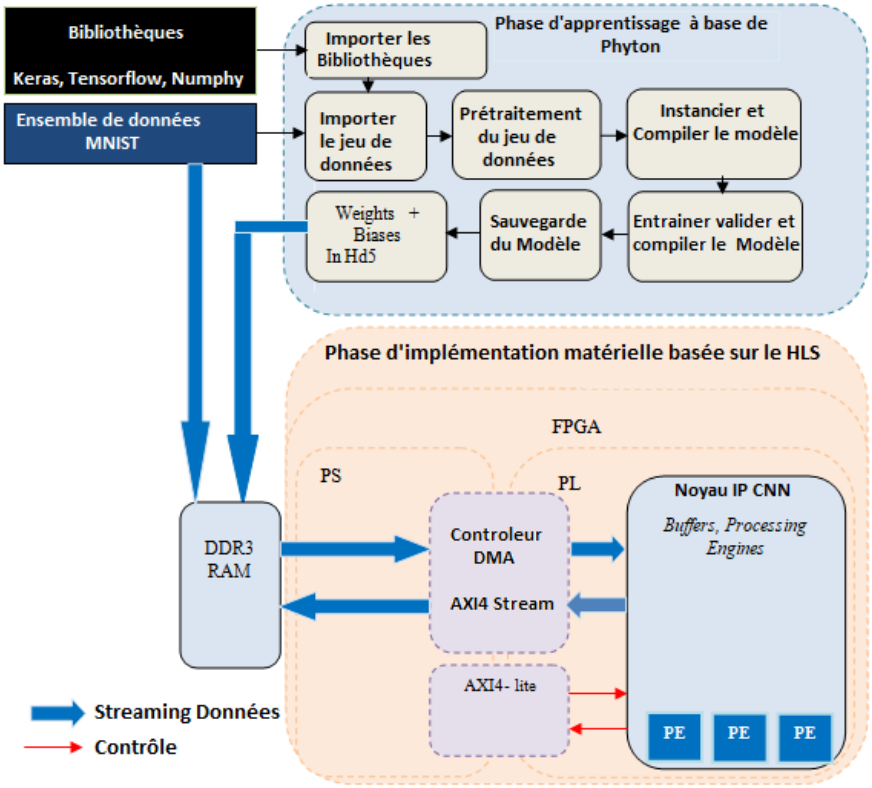


Figure 4.1: Aperçu complet de l'approche adoptée.

Le processus d'entraînement est réalise sous l'environnement Python, tandis que l'implémentation matérielle est réalisée sous l'environnement Vivado HLS. En début de la phase d'entraînement, un jeu de données est importé et prétraité en vue de l'entraînement du modèle CNN. Une fois que le processus de validation est achevé, les poids et les biais du modèle sont exportés au format HDF5 (consultez l'Annexe A.1) en vue de les utiliser ultérieurement lors de la phase d'implementation de l'inférence sur FPGA.

4.3 Phase d'apprentissage

L'objectif de cette phase est d'entraîner le modèle à l'aide d'un ensemble de données sur plusieurs époques. Cette étape vise à ajuster les paramètres des filtres du modèle CNN, essentielle pour garantir la qualité de l'implémentation ultérieure de l'inférence . Au cours de cette phase, le modèle progresse dans l'assimilation des caractéristiques et des motifs présents dans les données d'entraînement, avec chaque époque représentant une itération du processus d'apprentissage, où les paramètres du modèle sont réajustés pour minimiser l'erreur ou la perte. Lorsque le modèle a convergé ou atteint une performance satisfaisante, il est prêt à être utilisé pour des tâches de validation ou d'inférence. Les paramètres des filtres du modèle CNN, optimisés au cours de cette phase, sont importants pour garantir la qualité des résultats lors de la mise en œuvre du modèle pour des tâches concrètes. Durant cette phase d'apprentissage, nous avons utilisé un ensemble de données (MNIST) *(Modified National Institute of Standards and Technology database)*.De plus amples détails sur cet ensemble de données sont disponibles dans l'annexe (A.2).

En outre, nous avons préalablement installé plusieurs bibliothèques utiles dans notre environnement Python. Chacune de ces bibliothèques est spécialisée dans une tâche spécifique, offrant une gamme variée de fonctionnalités. Parmi celles-ci, on trouve notamment, mais non exhaustivement :

1. **Keras ;** Cette bibliothèque offre une interface de haut niveau qui permet de concevoir, de former et d'évaluer des modèles de manière intuitive, sans avoir à plonger profondément dans les détails de l'implémentation.
2. **TensorFlow ;** Cette bibliothèque simplifie considérablement la création, la formation et la mise en service de modèles d'apprentissage automatique, que ce soient des réseaux de neurones convolutionnels, des réseaux de neurones récurrents ou d'autres types d'architectures.
3. **Numphy ;** Cette bibliothèque offre une gamme étendue de fonctionnalités pour la manipulation de tableaux multidimensionnels. Ces fonctionnalités incluent des opérations mathématiques, statistiques et algébriques, ce qui facilite le traitement et la manipulation de données numériques.

La collaboration harmonieuse entre ces trois bibliothèques et d'autres, facilite la mise en œuvre de projets d'apprentissage automatique, en accélérant le processus de développement et en améliorant les performances des modèles.

4.3.1 Prétraitement du jeu de données

Avant d'être utilisé dans la tâche d'apprentissage, le jeu de données MNIST fait l'objet d'un prétraitement. dans un premier temps et Une fois importé, l'ensemble de données est réparti en trois catégories distinctes :

1. *Le jeu de données d'entraînement :* Il est utilisé par le modèle CNN pendant la phase d'apprentissage. Il sert à former le modèle, à ajuster les poids des neurones et à permettre au réseau de s'adapter aux données d'entraînement.
2. *Le jeu de données de validation :* Il intervient également pendant la phase d'entraînement. Cependant, sa principale fonction est d'évaluer les performances du CNN à différents époques de l'entraînement. Cela permet de surveiller la capacité du modèle à généraliser à partir des données d'entraînement et à détecter d'éventuels problèmes de surapprentissage.
3. *Le jeu de données de test :* Il est utilisé après la fin de la phase d'entraînement. Il sert à évaluer les performances finales du modèle, en mesurant sa capacité à prendre des décisions précises sur des données qu'il n'a jamais rencontrées auparavant. Ce jeu de données de test est essentiel pour évaluer l'efficacité du modèle dans des conditions réelles et pour estimer sa capacité à généraliser à de nouvelles données tout en maintenant des niveaux de précision et de perte acceptables.

Le prétraitement du jeu de données assure le fonctionnement optimal du modèle et la production de prédictions précises. Pour commencer, les images du jeu de données, initialement de taille 28x28, sont redimensionnées en ajoutant une valeur de rembourrage "*Padding*" 2 de chaque côté. Cette opération vise à ajuster les images pour les adapter aux exigences de la couche d'entrée du modèle LeNet5, qui requiert des images de taille 32x32. Ensuite, les images subissent une transformation les convertissant en vecteurs unidimensionnels, afin qu'elles s'alignent parfaitement avec le format requis par le modèle. De plus, les valeurs des pixels de l'ensemble de données sont normalisées de manière à être comprises dans une plage allant de 0 à 1. Les Figures 4.2, 4.3, 4.4 et 4.5 présentent de manière séquentielle les étapes de prétraitement nécessaires pour le transfert d'apprentissage sur l'ensemble de données.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Figure 4.2: Chargement et répartition de l'ensemble de données.

```
X_train = np.array([np.pad(X_train[i], pad_width=2) for i in range(X_train.shape[0])])
X_test = np.array([np.pad(X_test[i], pad_width=2) for i in range(X_test.shape[0])])
```

Figure 4.3: Redimensionnement de l'ensemble de données en 32x32 en utilisant des valeurs de rembourrage de 2.

```
X_train = X_train.reshape(X_train.shape[0], 32, 32, 1)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 1)
```

Figure 4.4: Transformation de l'ensemble de données en un vecteur unidimensionnel 1-D.

```
X_train = X_train / 255
X_test = X_test / 255
```

Figure 4.5: Normalisation des valeurs de pixel des ensembles d'entraînement et de test.

4.3.1.1 Construction du Modèle CNN

Cette étape de construction repose sur le modèle CNN LeNet5, dont l'architecture est présentée dans la Section 2.6. Il a été spécifiquement conçu pour la reconnaissance de chiffres écrits à la main. Un aspect notable de sa conception est que toutes les couches de convolution ont un noyau de taille 5x5, utilisent un pas de glissement de 1, et appliquent une fonction d'activation ReLU. De même, toutes les couches de mise en commun ont un noyau de taille 2x2 et un pas de glissement de 2. Les trois couches de convolution associées utilisent respectivement 6, 16, et 120 filtres. La couche de sortie est composée de dix neurones, chacun représentant un chiffre de 0 à 9. Notons que le nombre total de paramètres (poids et biais) dans ce réseau dépasse les 61 000. La Figure 4.6 résume les diverses couches qui composent LeNet5, en fournissant également les paramètres spécifiques de chaque couche après l'achèvement de la construction du modèle.

```
lenet_5_model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 28, 28, 6)         156

 average_pooling2d (Average  (None, 14, 14, 6)         0
 Pooling2D)

 conv2d_2 (Conv2D)           (None, 10, 10, 16)        2416

 average_pooling2d_1 (Avera  (None, 5, 5, 16)          0
 gePooling2D)

 conv2d_3 (Conv2D)           (None, 1, 1, 120)         48120

 flatten_1 (Flatten)         (None, 120)               0

 dense_2 (Dense)             (None, 84)                10164

 dense_3 (Dense)             (None, 10)                850

=================================================================
Total params: 61706 (241.04 KB)
Trainable params: 61706 (241.04 KB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 4.6: Sommaire du modèle CNN LeNet5 construit .

4.3.2 Compilation du modèle

Avant de démarrer le processus d'apprentissage, il est nécessaire d'effectuer un ajustement préliminaire de plusieurs paramètres clés. Parmi ces paramètres figurent le nombre d'époques (ou d'itérations), qui détermine le nombre de fois que l'ensemble complet des données d'entraînement sera parcouru, la taille du lot (batch size), qui spécifie la quantité d'exemples de données utilisée lors de chaque itération d'entraînement, la fonction de perte destinée à évaluer l'écart entre les prédictions du modèle et les valeurs réelles, ainsi que l'optimiseur responsable de mettre à jour les poids du modèle dans le but de minimiser cette fonction de perte. Pour ce dernier paramètre, nous avons opté pour l'optimiseur de type Descente de Gradient stochastique (SGD) *(Stochastic Gradient Descent)* en raison de son efficacité avérée dans l'ajustement des poids du modèle tout en minimisant les pertes. Le SGD, ou Descente de gradient stochastique, est reconnu pour sa capacité à converger rapidement vers des solutions optimales tout en étant particulièrement adapté à des ensembles de données volumineux.

```
EPOCHS = 10
BATCH_SIZE = 32
LOSS = keras.metrics.categorical_crossentropy
OPTIMIZER = keras.optimizers.SGD()
METRIC = ['accuracy']
```

Figure 4.7: Ajustement des hyperparamètres de compilation.

4.3.3 Validation et sauvegarde des poids et des biais du modèle

En complément aux métriques classiques telles que les courbes de précision et de perte fréquemment utilisées lors de la validation de la phase d'apprentissage, la matrice de confusion émerge en tant qu'indicateur de performance robuste. Cette matrice fournit une visualisation structurée des prédictions générées par un modèle en comparaison avec les instances réelles associées à ces prédictions. Dans le contexte de nos travaux, la matrice de confusion adopte généralement des dimensions carrées pour les problèmes de classification binaire. Elle comporte quatre catégories distinctes, à savoir les vrais positifs (VP), les vrais négatifs (VN), les faux positifs (FP), et les faux négatifs (FN). Les vrais positifs se réfèrent aux instances correctement classées comme appartenant à la classe positive, tandis que les vrais négatifs représentent les instances correctement classées comme n'appartenant pas à la classe positive. Les faux positifs désignent les instances incorrectement classées comme appartenant à la classe positive, tandis que les faux négatifs concernent les instances incorrectement classées comme n'appartenant pas à la classe positive. La Figure 4.8 illustre la configuration et les paramètres de la matrice de confusion.

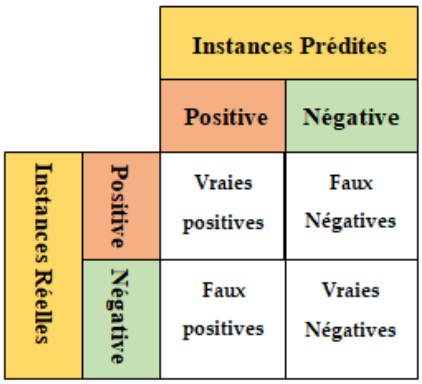


Figure 4.8: Structure et paramètres de la Matrice de confusion.

Les expressions 4.1, 4.2 et 4.3 fournissent un aperçu statistique de la performance de

chaque classe.

$$Precision = \frac{VP}{VP + FP} \tag{4.1}$$

$$Recall = \frac{VP}{VP + FN} \tag{4.2}$$

$$FScore = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.3}$$

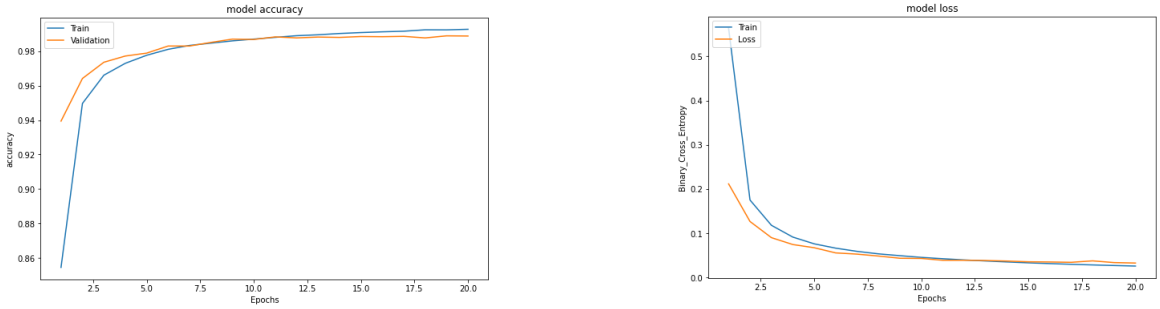
4.4 Analyse des Résultats et Discussion de la Phase d'Apprentissage

Au cours de la phase d'apprentissage, le modèle CNN LeNet5 a été soumis à un entraînement initial sur dix époques, suivi d'un second entraînement sur vingt époques, en utilisant un ensemble de 600 images MNIST. Par la suite, les courbes de précision et de perte ont été générées à partir de l'historique d'apprentissage pour les deux scénarios, offrant ainsi une vision détaillée des performances du modèle à différentes étapes de l'entraînement ainsi que pour les deux scénarios.

Comme le montrent les figures 4.9 et 4.10, les courbes de précision et de perte des deux cas indiquent que le modèle apprend rapidement, avec une augmentation linéaire de la précision au cours des deux premières époques. Ensuite, la courbe commence à s'aplatir, suggérant que l'apprentissage commence à se stabiliser et ne nécessite pas autant d'époques pour terminer le processus d'apprentissage du modèle. De même, la courbe de validation augmente linéairement au cours des deux premières époques, puis commence à s'aplanir, finissant par suivre pleinement la courbe de précision à partir de la septième époque. Cela indique que le modèle fonctionne efficacement, en apprenant rapidement au cours des premières époques avant de parvenir à une certaine stabilité.

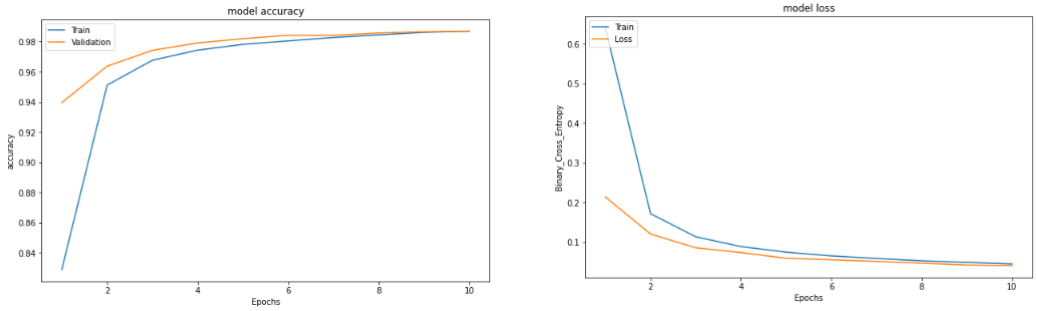
Dans la phase de test, la perte diminue lentement au cours des quatre premières époques, puis tend à se stabiliser pour les époques subséquentes, indiquant ainsi que notre modèle CNN généralise efficacement à des données précédemment inconnues. En guise de conclusion, il est à noter que dix époques se révèlent être suffisantes pour que le processus de transfert d'apprentissage s'achève avec une précision dépassant les 98%. Cette observation met en lumière la capacité du modèle à assimiler efficacement les connaissances acquises lors de la phase d'apprentissage, démontrant ainsi la convergence du processus

avec un niveau élevé de précision.



(a) **(b)**

Figure 4.9: Courbes de performance de la phase d'apprentissage sur 20 époques : (a) évolution de la précision du modèle ; (b) évolution de la perte du modèle.



(a) **(b)**

Figure 4.10: Courbes de performance de la phase d'apprentissage sur 10 époques : (a) évolution de la précision du modèle ; (b) évolution de la perte du modèle.

Nous avons également effectué une synthèse des performances de la phase d'entraînement en utilisant une matrice de confusion, comme abordée dans la section 4.3.3, et dont le graphique est présenté dans la Figure 4.11. Cette approche nous a permis de mener une analyse statistique approfondie du paramètre F-Score, qui combine à la fois la précision et le rappel. En observant la Figure 4.12, nous remarquons que, bien que les étiquettes 2, 3, 6 et 9 présentent un F-Score légèrement inférieur à la moyenne, elles maintiennent un F-Score dépassant les 97%, ce qui est considérablement satisfaisant. Pour illustrer davantage, examinons la prédiction de l'étiquette 8, qui affiche le F-Score le plus bas.

Cette diminution est principalement attribuable à la présence de 20 faux positifs dans la prédiction de cette étiquette par rapport à l'ensemble de toutes les étiquettes réelles.

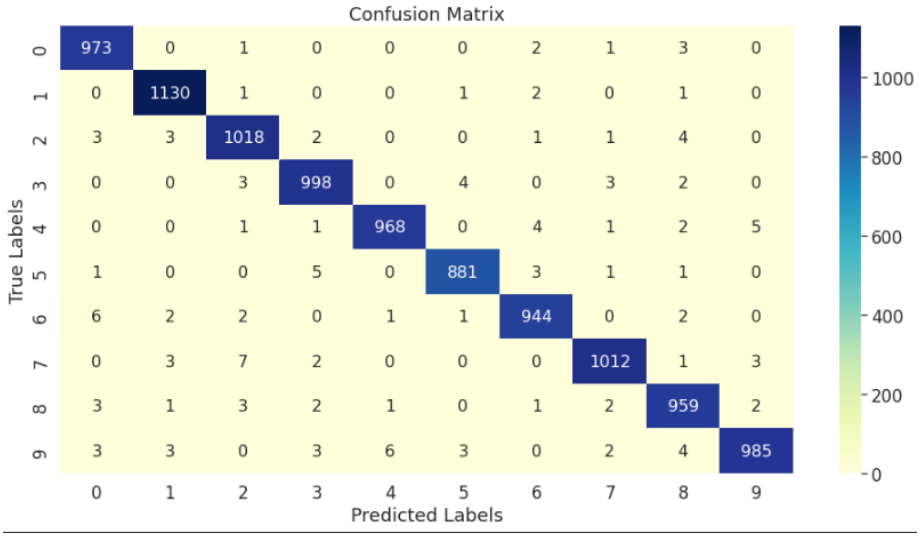


Figure 4.11: Matrice de confusion.

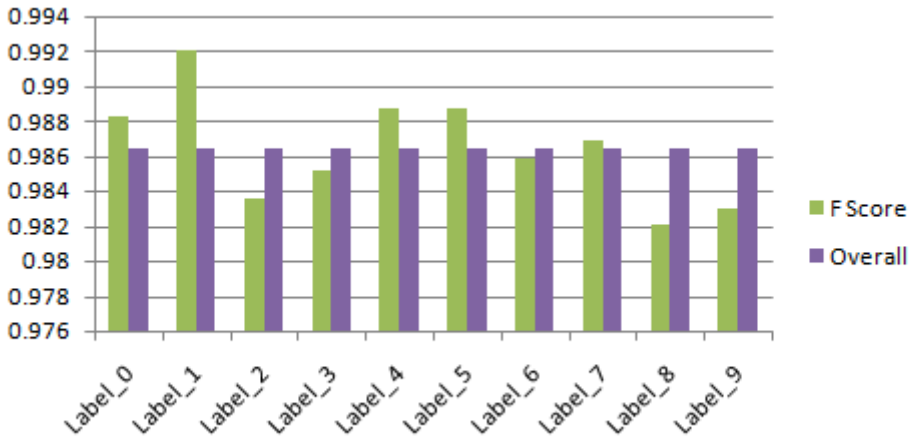


Figure 4.12: Histogramme Histogramme FScore.

4.5 Implémentation Matérielle de l'Inférence et Stratégies d'optimisation

Durant cette phase d'inférence, notre objectif principal réside dans la mise en œuvre du modèle sur une plateforme FPGA, avec pour ambition d'accélérer significativement le processus tout en respectant les contraintes de performances et de ressources disponibles dans

la plateforme. Dans cette optique, nous avons adopté une approche progressive d'optimisation tout au long de cette phase. À chaque itération d'optimisation, nous avons généré un rapport d'analyse et de synthèse afin d'évaluer l'impact de chaque amélioration sur le système. Cette méthode méthodique nous permet d'examiner attentivement les modifications apportées et de prendre des décisions éclairées pour une optimisation continue du système.

Par ailleurs, et comme précédemment mentionné dans la section (2.7), nous rappelons que la couche de convolution représente plus de 90% du temps de calcul. En conséquence, nos efforts d'optimisation se concentrent principalement sur cette couche. Nous nous attelons principalement à deux aspects importants. En premier lieu, nous cherchons à optimiser le flux de données afin de réduire les accès mémoire et, par conséquent, la consommation d'énergie. Ensuite, nous nous efforçons d'optimiser le temps d'exécution en utilisant la parallélisation des opérations. Il est important de noter que ces optimisations ne peuvent être mises en œuvre que si l'architecture adoptée le permet et s'y conforme. À cet égard, nous avons opté pour l'architecture en flux de données, qui s'est révélée être une base propice pour ce type d'optimisation.

Cependant, avant de mettre en œuvre tous les modules du modèle CNN en une seule fois, y compris les couches de convolution, les couches de mise en commun et les couches entièrement connectées, nous avons d'abord adopté une approche modulaire. Dans le cadre de cette approche, nous avons mené une validation fonctionnelle distincte pour chaque module. De plus, nous avons exploré le format de quantification idéal pour la suite du processus. Ce dernier commence par l'établissement du module de convolution, suivi de près par le développement du module de mise en commun, pour enfin aboutir à l'implémentation du module entièrement connecté. Cette approche modulaire présente plusieurs avantages significatifs :

— Tout d'abord, elle facilite les phases de test et de validation individuelle de chaque module, permettant ainsi une évaluation approfondie de la performance de chaque module avant l'intégration complète.
— En outre, cette approche modulaire offre une meilleure flexibilité, autorisant des améliorations ou des ajustements spécifiques à un module sans impacter de manière majeure le reste du modèle CNN.
— De plus, elle permet la réutilisation du module avec des ajustements minimes dans divers contextes du modèle.

4.5.1 Implémentation Fonctionnelle des Modules et Exploration de la Quantification

Dans cette section, nous nous attelons à l'exploration et à la validation non seulement de la fonctionnalité des divers modules ou couches, mais aussi au choix optimal du type et de la taille de la quantification des données et des paramètres, tout en préservant la précision de prédiction du modèle.

4.5.1.1 Élaboration et Implémentation du Module de Convolution

L'implémentation du module de convolution repose principalement sur l'algorithme décrit dans l'algorithme 2.1, dont le développement détaillé est exposé dans la section (2.7). Le schéma bloc de l'interface du module de convolution doit inclure plusieurs signaux de contrôle en entrée, notamment le signal d'horloge **ap_clk**, le signal de réinitialisation **ap_rst** et le signal de départ **ap_start**. En sortie, il doit fournir les signaux processus termine **ap_done**, processus inactif **ap_idle** et enfin processus Prêt **ap_ready**. La figure 4.13 présente l'illustration de l'interface du module de convolution à mettre en œuvre, avec les divers signaux nécessaires sus-cités.

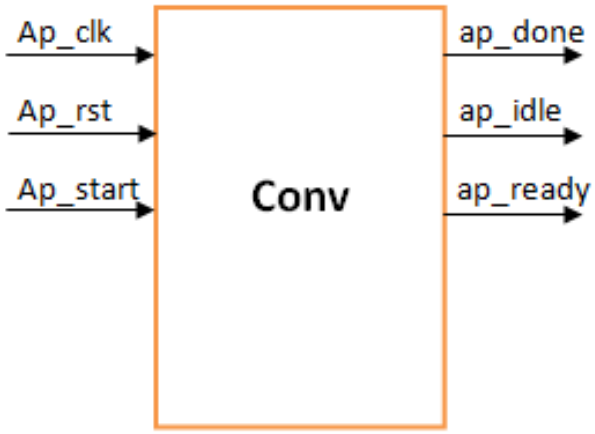


Figure 4.13: Schéma bloc représentant l'interface du module de convolution.

Dans le cadre de cette mise en œuvre du module de convolution, deux objectifs spécifiques sont clairement définis. Tout d'abord, il s'agit de vérifier la fonctionnalité du module, assurant ainsi sa conformité avec les spécifications attendues. Ensuite, l'exploration du format de quantification des données à adopter constitue un second objectif

majeur. Ce dernier vise fondamentalement à déterminer la taille minimale de la quantification des données nécessaire pour garantir que la performance et la fonctionnalité du module demeurent à un niveau optimal. L'objectif ultime de cette démarche est de parvenir à définir une taille minimale de quantification des données qui, tout en conservant une efficacité optimale, contribue à la réduction des ressources nécessaires, tant en termes de mémoire que de puissance de calcul, et par conséquent, à une mise en œuvre efficiente et performante.

Dans cette perspective, nous avons opté pour l'utilisation de la représentation en virgule fixe, motivée par sa capacité à fournir un compromis optimal entre précision, efficacité et simplicité. La Figure 4.14 présente le format de la représentation en virgule fixe, mettant en évidence sa pertinence dans le cadre général d'optimisation.

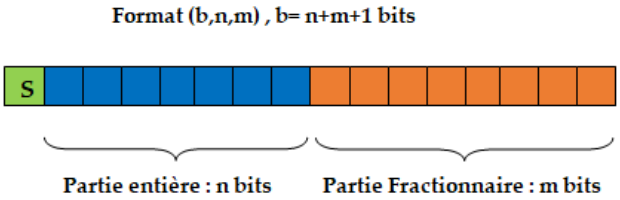


Figure 4.14: Format de la représentation en virgule fixe.

Dans le cadre de la réduction du nombre de bits utilisés pour représenter les données, notre approche a débuté par la détermination de l'intervalle des valeurs des poids et des biais des filtres générés lors de la phase d'apprentissage. Cela nous a fourni une vision initiale du format de la représentation en virgule fixe, à partir duquel notre exploration a commencé. Dans une seconde étape, un seuil d'erreur a été établi entre la valeur calculée par le module et la valeur prédite lors de la phase d'apprentissage. Cette démarche vise à vérifier, au cours de cette exploration, le taux de perte, c'est-à-dire le nombre de cartes caractéristiques de sortie issues du module qui ne correspondent pas à celles prédites.

En ce qui concerne la gamme des valeurs des poids et des biais dans notre démarche, nous avons méthodiquement entrepris une phase préliminaire en déterminant les valeurs maximales et minimales associées. Une exploration approfondie à travers l'ensemble de la base de données a révélé que la valeur minimale des paramètres (poids, biais) est de -0.535074, tandis que la valeur maximale atteint 0.520299.

Par ailleurs, les valeurs de l'ensemble des données des images ont été normalisées pour être représentées en niveaux de gris, variant de $0/255$ à $255/255$, soit dans l'intervalle $[0, 1]$.

Cette normalisation assure une uniformité dans la représentation des données, facilitant ainsi le processus d'analyse et d'apprentissage. De plus, nous avons pris des précautions spécifiques lors des opérations de multiplication et d'addition appliquées sur l'ensemble de ces données pendant les opérations de convolution. Ceci inclut la considération de l'étendue du format pour garantir la précision des calculs et éviter tout risque de dépassement ou de perte d'information pendant ces manipulations.

À la lumière de ces données et considérations, le choix du format de la représentation en virgule fixe doit inclure un bit de signe, deux à trois, voire quatre bits pour la partie entière, et un nombre déterminé de bits pour la partie fractionnaire. Afin de déterminer précisément le format à appliquer lors de l'implémentation finale du modèle CNN, nous avons exploré plusieurs formats et évalué leur impact sur la perte de précision. En plus, nous avons défini un seuil de tolérance d'environ $\epsilon = 10^{-6}$ entre la valeur de la carte caractéristique de sortie calculée et celle prédite. Ce processus d'exploration se déroule en deux étapes.

Comme le format en virgule fixe se compose de deux segments distincts : la partie entière et la partie fractionnaire. L'analyse approfondie des deux segments est entreprise dans un premier temps en déterminant le nombre de bits de la partie entière de manière à prévenir toute perte de précision. Ensuite, la partie fractionnaire est examinée avec le seuil de tolérance .

4.5.1.2 Analyse et Discussion des Résultats

La figure 4.15, issue via l'analyse synthétique de l'environnement de développement Vivado HLS, illustre le résultat final de l'interface du module de convolution implémenté. Cette interface comprend les signaux de contrôle externes prévus ainsi que les paramètres internes de la convolution. Les caractéristiques de chaque paramètre sont également présentées, notamment la direction, le nombre de bits, le protocole et le type de données, offrant ainsi une vue détaillée de leur fonction et de leur utilisation dans le module de convolution.

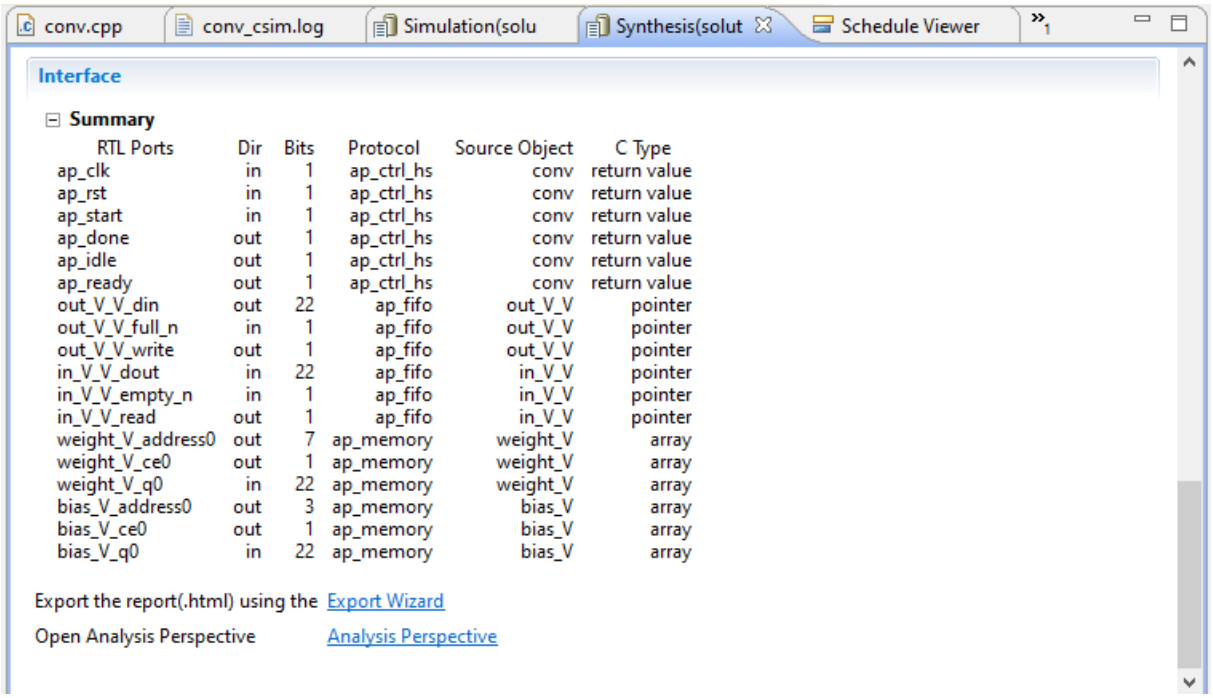


Figure 4.15: Interface du module de convolution implémentée.

À ce stade, seuls les fonctionnalités du module de convolution ont été validées. Les résultats de l'exploration du format de quantification sont les suivants :

1. **Format(24,4,20) :**

   Nous avons entrepris l'analyse du format de représentation des données et des paramètres en adoptant un format étendu de 24 bits dans le but de garantir la fonctionnalité du module lors de la cosimulation avec le banc d'essai "Testbench", visant ainsi à minimiser, voire éliminer, toute perte de précision et à atteindre un taux de perte équivalent à 0%. Dans ce schéma, la partie entière est codée sur 4 bits, tandis que la partie fractionnaire occupe 20 bits. La Figure 4.16 présente le rendu de la cosimulation du module de convolution. La section délimitée en rouge présente le nombre de correspondances entre les cartes de caractéristiques de sortie calculées par le module de convolution et celles prédites au cours de la phase d'apprentissage, accompagné du pourcentage de précision associé. Dans sa configuration actuelle, un taux de précision de 100% est accompli.

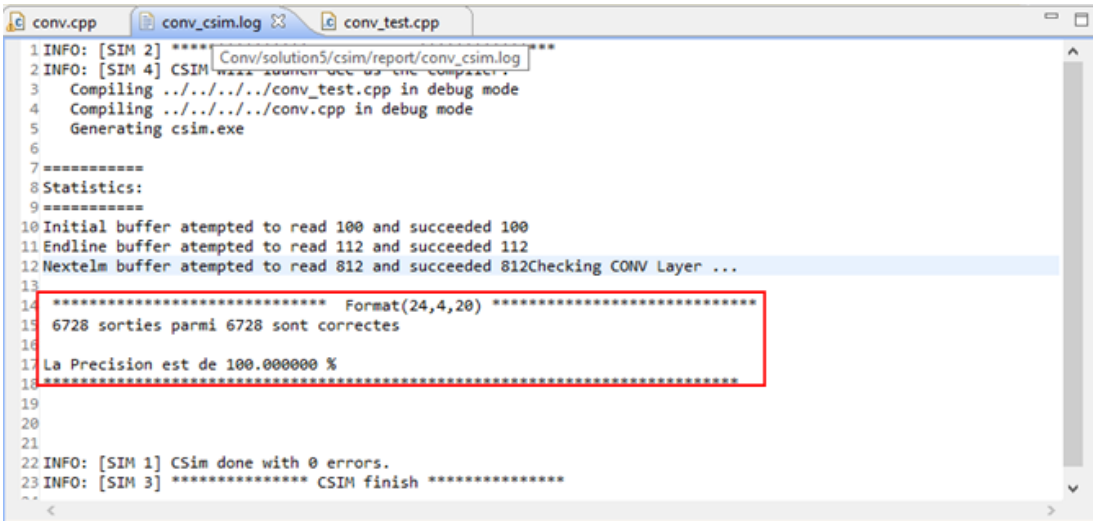


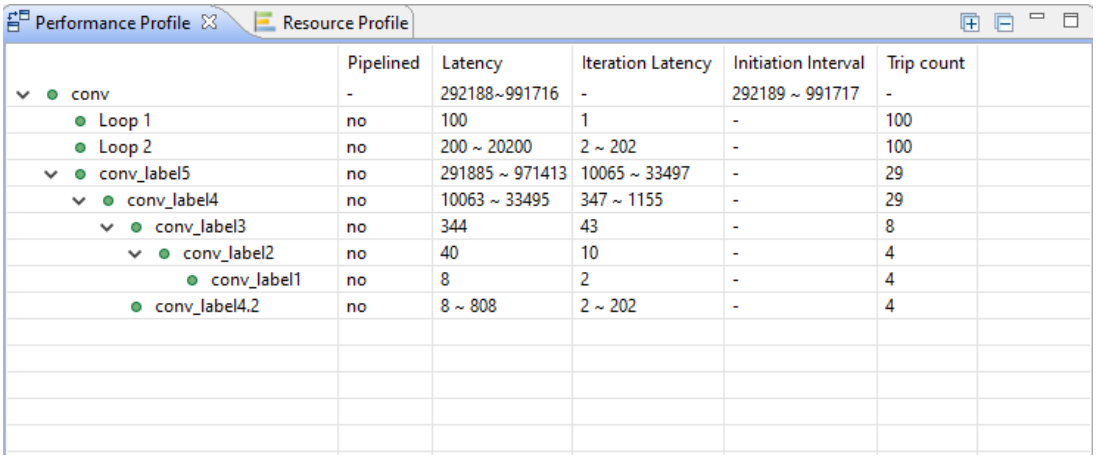
Figure 4.16: Cosimulation du module de convolution Cosimulation du module de convolution à base du format (24,4,20).

La conception architecturale du module de convolution, conformément à base de ce format, a engendré des métriques de profil liées à la performance et aux ressources. Ces dernières sont respectivement exposées de manière détaillée dans les figures 4.17 et 4.18. Il est important de souligner que la taille du format a un impact sur les ressources, mais n'affecte pas les performances en termes de latence. Ainsi, la latence demeure constante pour tous les formats explorés et est égale à 292189 picosecondes pour une fréquence de fonctionnement de la FPGA de 100MHz.

Performance Profile | Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	1	1	234	797						
I/O Ports(4)					96					
bias_V					24					
in_V_V					24					
out_V_V					24					
weight_V					24					
Instances(1)	0	0	41	92						
Memories(1)	1		0	0	24			1	100	2400
Expressions(36)	0	1	0	457	249	213	1			
Registers(29)			193		261					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	145			0		
DSP(1)		0								

Figure 4.17: Profil des ressources du module de convolution à base du format(24,4,20).



	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
conv	-	292188~991716	-	292189 ~ 991717	-
Loop 1	no	100	1	-	100
Loop 2	no	200 ~ 20200	2 ~ 202	-	100
conv_label5	no	291885 ~ 971413	10065 ~ 33497	-	29
conv_label4	no	10063 ~ 33495	347 ~ 1155	-	29
conv_label3	no	344	43	-	8
conv_label2	no	40	10	-	4
conv_label1	no	8	2	-	4
conv_label4.2	no	8 ~ 808	2 ~ 202	-	4

Figure 4.18: Profile de performances du module de convolution à base du format (24,4,20).

2. **Format(23,3,20) :** Dans le cadre de l'optimisation de la partie entière, la dimension de la partie fractionnaire reste inchangée, alors qu'une diminution de 1 bit est appliquée à la partie entière. La figure 4.19 expose les résultats de la cosimulation. En accord avec cette représentation graphique, un taux de précision de 100% est observé, indiquant la possibilité d'une réduction supplémentaire de cette partie entière.

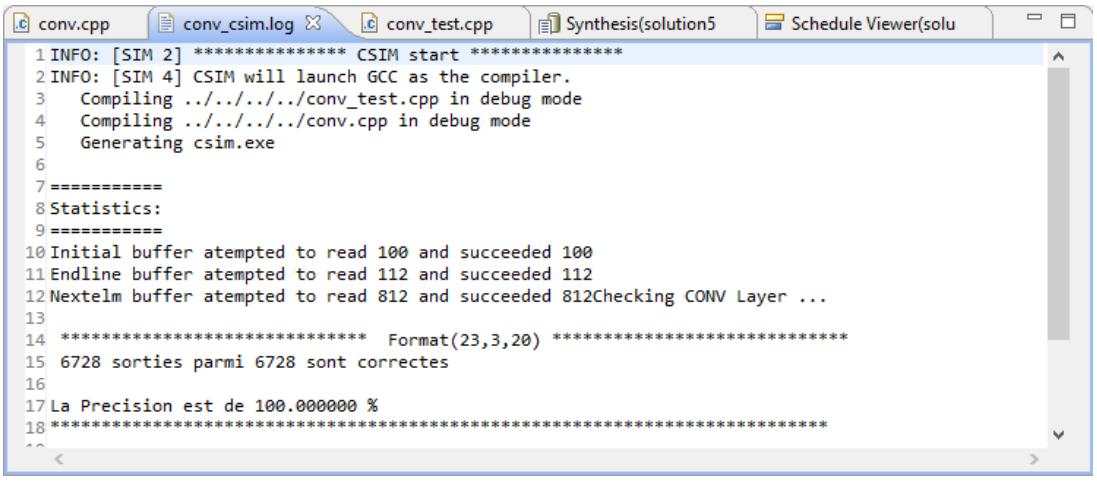


Figure 4.19: Cosimulation du module de convolution à base du format (23,3,20).

Les profils de ressources et de performances sont représentés respectivement dans les figures 4.20 et 4.21.

Performance Profile | Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	**1**	**1**	**229**	**785**						
I/O Ports(4)					**92**					
bias_V					23					
in_V_V					23					
out_V_V					23					
weight_V					23					
Instances(1)	**0**	**0**	**40**	**92**						
grp_InsertBack_fu_318	0	0	40	92						
Memories(1)	**1**		**0**	**0**	**23**			**1**	**100**	**2300**
conv_buff_array_V_U	1		0	0	23			1	100	2300
Expressions(36)	**0**	**1**	**0**	**445**	**244**	**208**	**1**			
Registers(29)			**189**		**257**					
Channels(0)	**0**		**0**	**0**	**0**			**0**	**0**	**0**
Multiplexers(18)	**0**		**0**	**248**	**141**			**0**		
DSP(1)		**0**								

Figure 4.20: Profil des ressources du module de convolution à base du format (23,3,20).

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
conv	-	292188~991716	-	292189 ~ 991717	-
Loop 1	no	100	1	-	100
Loop 2	no	200 ~ 20200	2 ~ 202	-	100
conv_label5	no	291885 ~ 971413	10065 ~ 33497	-	29
conv_label4	no	10063 ~ 33495	347 ~ 1155	-	29
conv_label3	no	344	43	-	8
conv_label2	no	40	10	-	4
conv_label1	no	8	2	-	4
conv_label4.2	no	8 ~ 808	2 ~ 202	-	4

Figure 4.21: Profile de performances du module de convolutionà base du format (23,3,20).

3. **Format(22,2,20) :** Puisque la précision du format précédent est parvenue à 100 %, nous poursuivons dans la même direction en réduisant une dernière fois d'un bit supplémentaire la partie entière dans le format actuel, tout en veillant à maintenir une précision de 100 %.

conv.cpp | conv_csim.log | conv_test.cpp | Synthesis(solut | Simulation(solu

```
INFO: [SIM 2] ************** CSIM start **************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.

===========
Statistics:
===========
Initial buffer atempted to read 100 and succeeded 100
Endline buffer atempted to read 112 and succeeded 112
Nextelm buffer atempted to read 812 and succeeded 812Checking CONV Layer ...

 ****************************  Format(22,2,20) ****************************
 6728 sorties parmi 6728 sont correctes

La Precision est de 100.000000 %
********************************************************************************



INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] ************** CSIM finish **************

```

Figure 4.22: Cosimulation du module Convolution à base du format (22,2,20).

Performance Profile | Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	1	1	224	773						
I/O Ports(4)					88					
bias_V					22					
in_V_V					22					
out_V_V					22					
weight_V					22					
Instances(1)	0	0	39	92						
Memories(1)	1		0	0	22			1	100	2200
conv_buff_array_V_U	1		0	0	22			1	100	2200
Expressions(36)	0	1	0	433	239	203	1			
Registers(29)			185		253					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	137			0		
DSP(1)		0								

Figure 4.23: Profil de ressources à base du format (22,2,20).

4. **Format(21,2,19) :** À ce point, seule la partie entière de la représentation en virgule fixe a été examinée et est désormais fixée à deux bits. Dans la poursuite de l'analyse, une réduction de 1 bit est appliquée à chaque nouveau format exploré. La Figure 4.24 illustre le résultat de la cosimulation du module de convolution avec une précision de 100 %, tandis que la Figure 4.25 récapitule toutes les ressources utilisées.

conv.cpp | conv_csim.log | Simulation(solu | Synthesis(solut | Schedule Viewer

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../conv_test.cpp in debug mode
   Compiling ../../../../conv.cpp in debug mode
   Generating csim.exe

===========
Statistics:
===========
Initial buffer atempted to read 100 and succeeded 100
Endline buffer atempted to read 112 and succeeded 112
Nextelm buffer atempted to read 812 and succeeded 812Checking CONV Layer ...

 ****************************  Format(21,2,19) ****************************
 6728 sorties parmi 6728 sont correctes

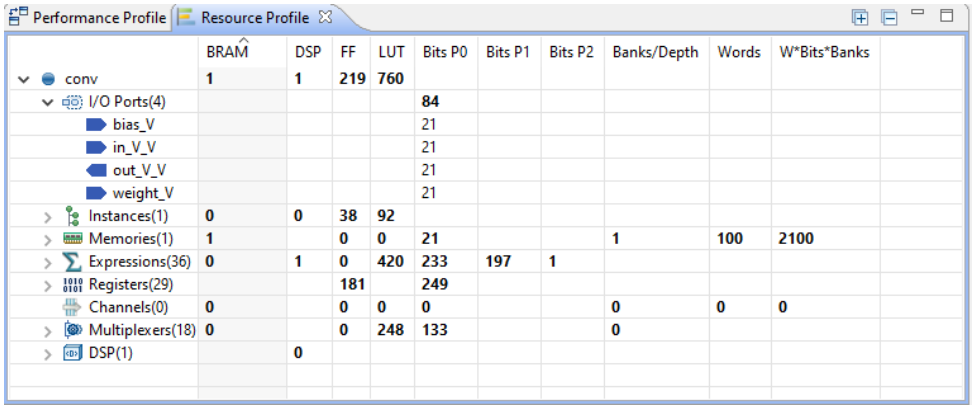
La Precision est de 100.000000 %
*********************************************************************************



INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

Figure 4.24: Cosimulation du module Convolution à base du format (21,2,19).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	1	1	219	760						
I/O Ports(4)					84					
bias_V					21					
in_V_V					21					
out_V_V					21					
weight_V					21					
Instances(1)	0	0	38	92						
Memories(1)	1		0	0	21			1	100	2100
Expressions(36)	0	1	0	420	233	197	1			
Registers(29)			181		249					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	133			0		
DSP(1)		0								

Figure 4.25: Profil de ressources à base du format (21,2,19).

5. **Format(20,2,18) :** La Figure 4.26 illustre le résultat de la cosimulation du module de convolution avec une précision de 100 %, tandis que la Figure 4.27 récapitule toutes les ressources utilisées.

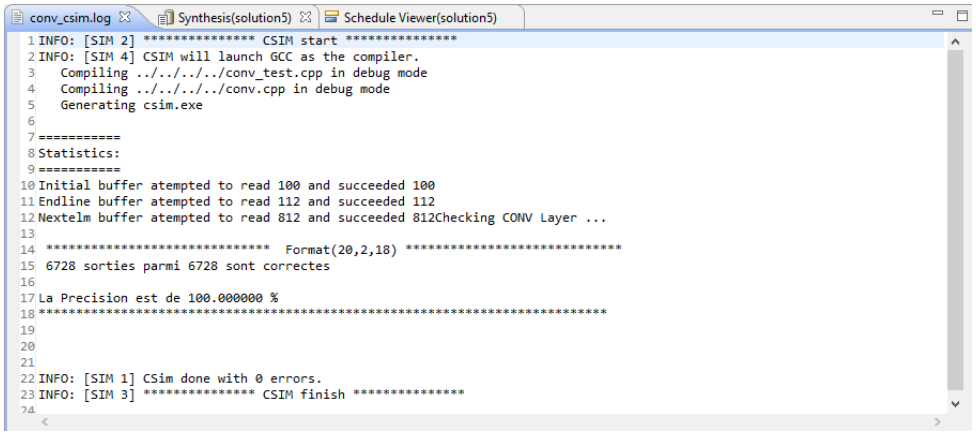
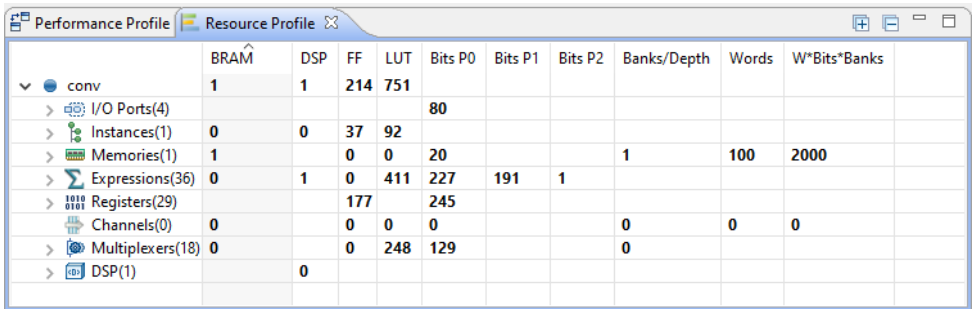


Figure 4.26: Cosimulation du module Convolution à base du format (20,2,18).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	1	1	214	751						
I/O Ports(4)					80					
Instances(1)	0	0	37	92						
Memories(1)	1		0	0	20			1	100	2000
Expressions(36)	0	1	0	411	227	191	1			
Registers(29)			177		245					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	129			0		
DSP(1)		0								

Figure 4.27: Profil de ressources à base du format (20,2,18).

6. **Format(19,2,17) :** La Figure 4.28 présente les résultats de la cosimulation du module de convolution avec une précision de 98%. À partir de ces données, nous

observons le début d'une régression de la précision. En outre, la Figure 4.29 résume l'ensemble des ressources utilisées.

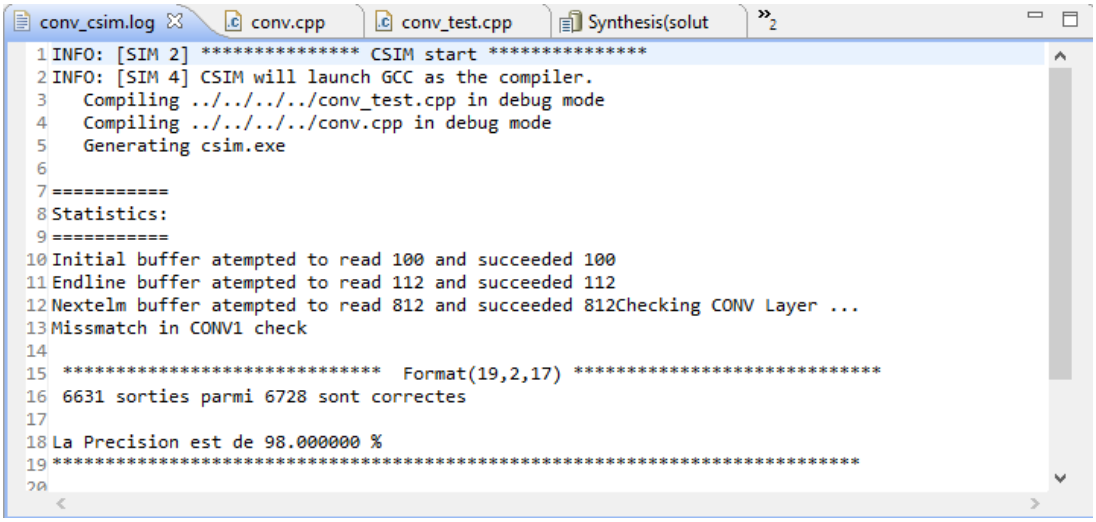
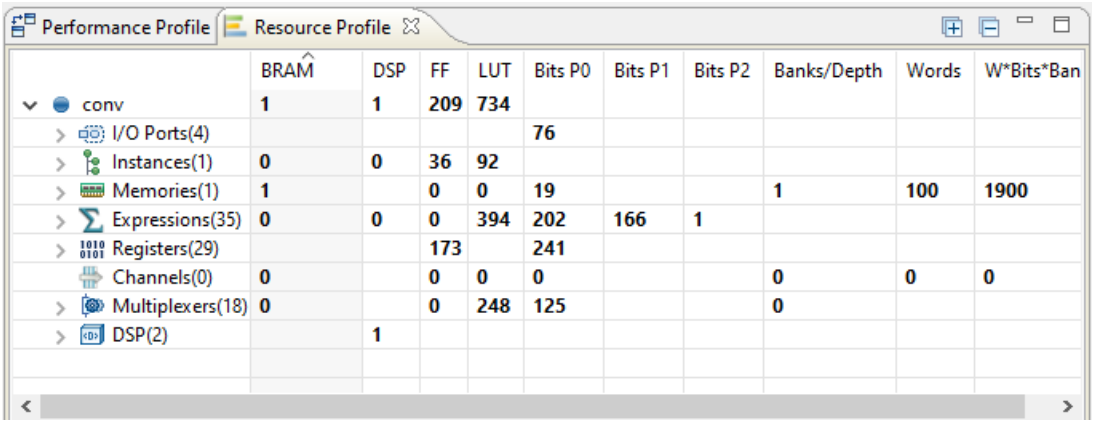


Figure 4.28: Cosimulation du module Convolution à base du format (19,2,17).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Ban
conv	1	1	209	734						
I/O Ports(4)					76					
Instances(1)	0	0	36	92						
Memories(1)	1		0	0	19			1	100	1900
Expressions(35)	0	0	0	394	202	166	1			
Registers(29)			173		241					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	125			0		
DSP(2)		1								

Figure 4.29: Profil de ressources à base du format (19,2,17).

7. **Format(18,2,16) :** La précision de la cosimulation du module de convolution diminue progressivement à 89 %, comme illustré dans la Figure 4.30, tandis que la Figure 4.31 présente un résumé de toutes les ressources utilisées.

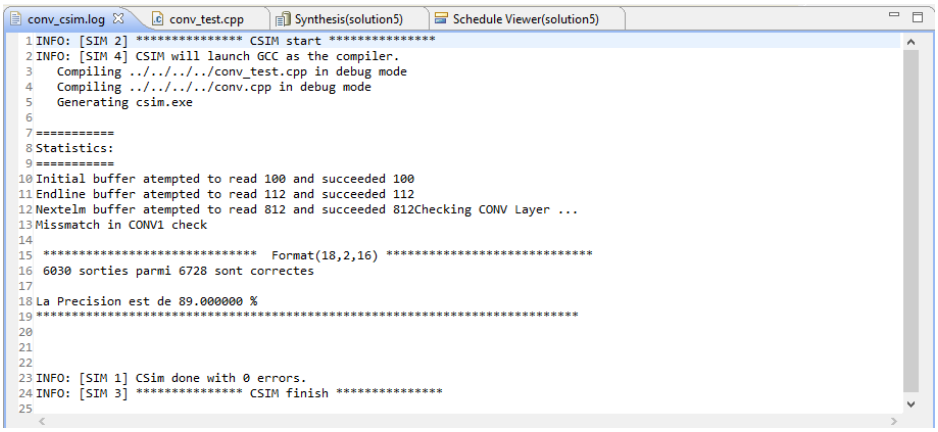


Figure 4.30: Cosimulation du module Convolution à base du format (18,2,16).

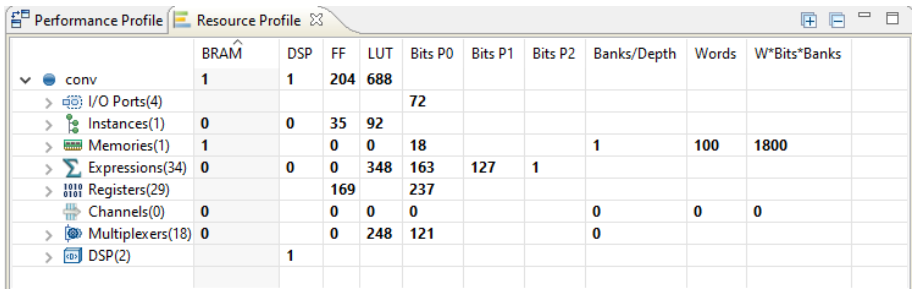


Figure 4.31: Profil de ressources à base du format (18,2,16).

8. **Format(17,2,15) :** La Figure 4.32 illustre le résultat de la cosimulation du module de convolution avec une précision de 85 %, tandis que la Figure 4.33 récapitule toutes les ressources utilisées.

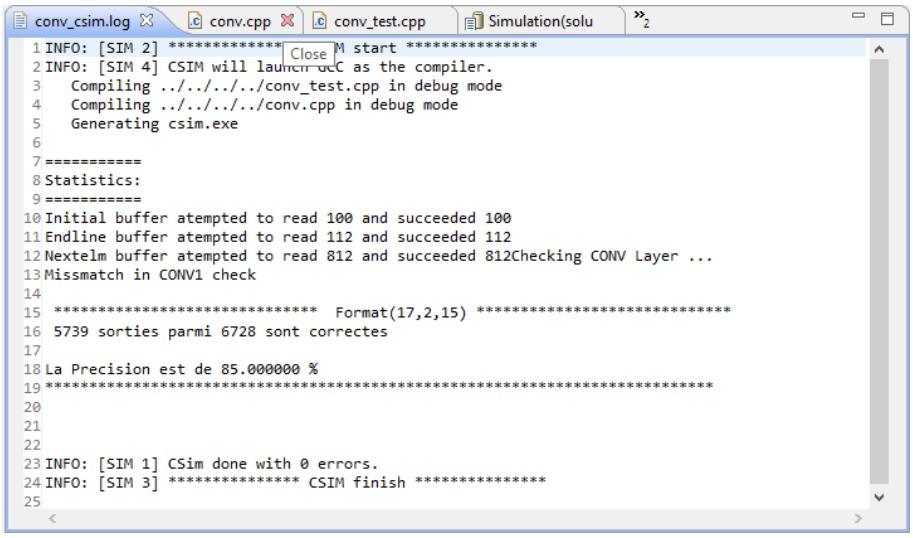
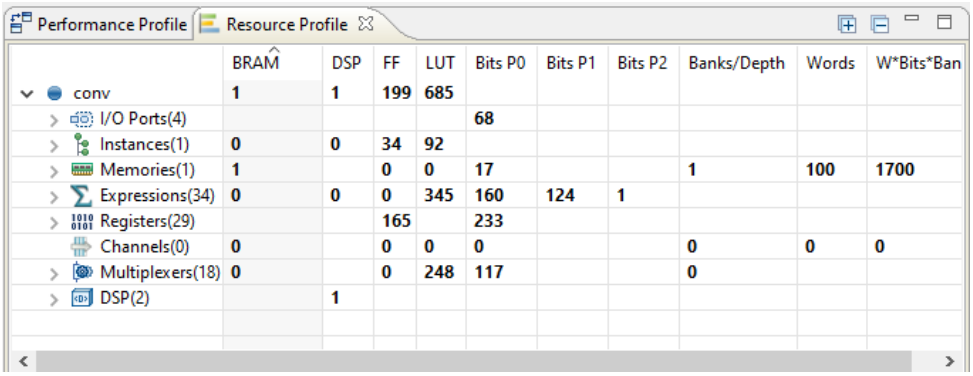


Figure 4.32: Cosimulation du module Convolution à base du format (17,2,15).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Ban
conv	1	1	199	685						
I/O Ports(4)					68					
Instances(1)	0	0	34	92						
Memories(1)	1		0	0	17			1	100	1700
Expressions(34)	0	0	0	345	160	124	1			
Registers(29)			165		233					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	117			0		
DSP(2)		1								

Figure 4.33: Profil de ressources à base du format (17,2,15).

9. **Format(16,2,14) :** La Figure 4.34 illustre le résultat de la cosimulation du module de convolution avec une précision de 80 %, tandis que la Figure 4.35 récapitule toutes les ressources utilisées.

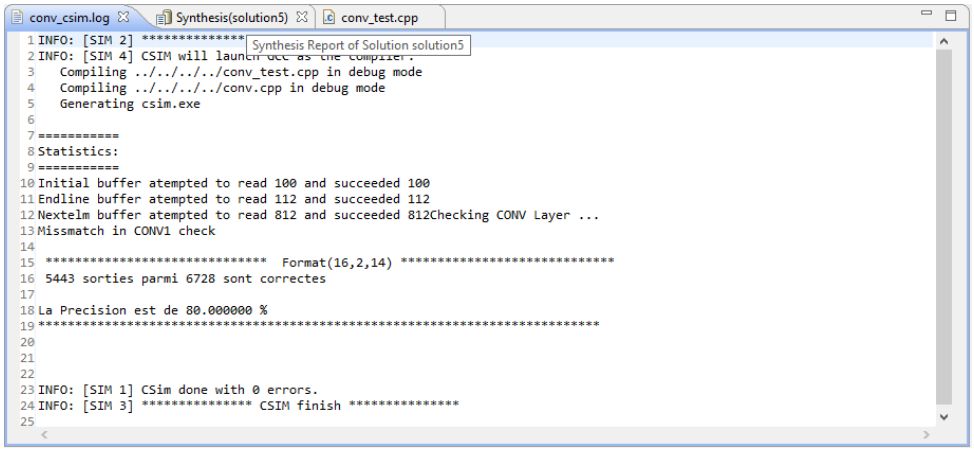
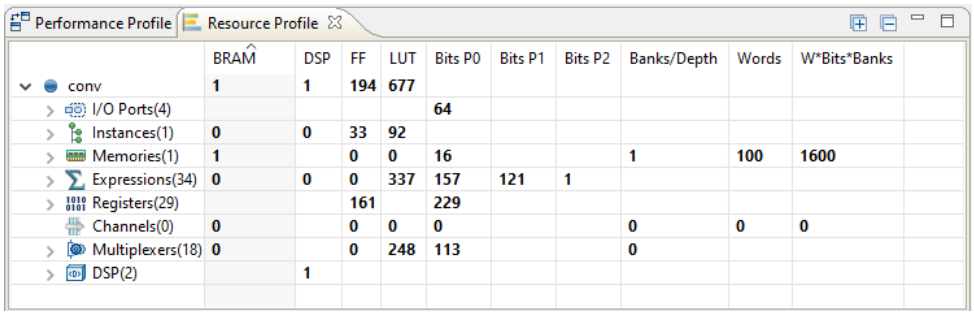


Figure 4.34: Cosimulation du module Convolution à base du format (16,2,14).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	1	1	194	677						
I/O Ports(4)					64					
Instances(1)	0	0	33	92						
Memories(1)	1		0	0	16			1	100	1600
Expressions(34)	0	0	0	337	157	121	1			
Registers(29)			161		229					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	113			0		
DSP(2)		1								

Figure 4.35: Profil de ressources à base du format (16,2,14).

10. **Format(15,2,13) :** La Figure 4.36 illustre le résultat de la cosimulation du module de convolution avec une précision de 78 %, tandis que la Figure 4.37 récapitule

toutes les ressources utilisées.

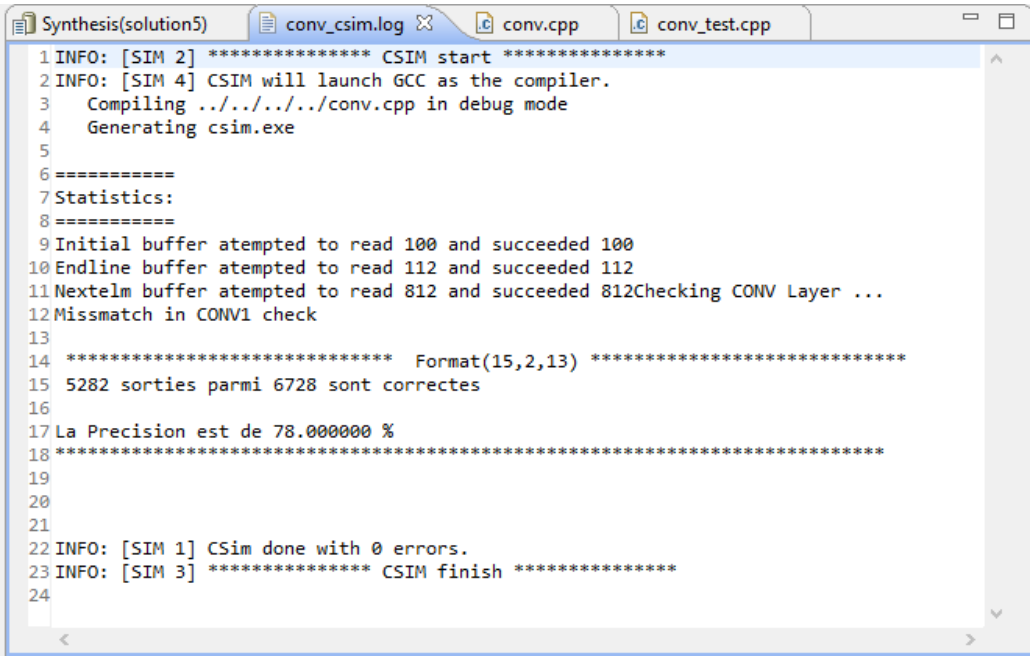
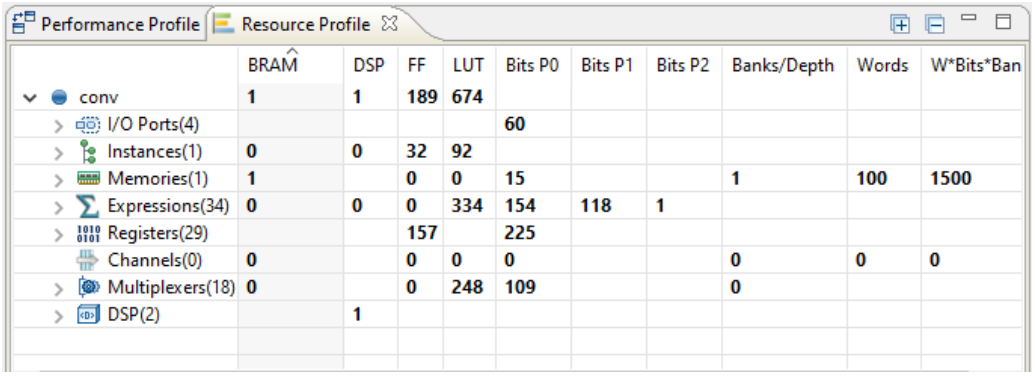


Figure 4.36: Cosimulation du module de Convolution à base du format (15,2,13).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Ban
conv	1	1	189	674						
I/O Ports(4)					60					
Instances(1)	0	0	32	92						
Memories(1)	1		0	0	15			1	100	1500
Expressions(34)	0	0	0	334	154	118	1			
Registers(29)			157		225					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	109			0		
DSP(2)		1								

Figure 4.37: Profil de ressources à base du format (15,2,13).

11. **Format(14,2,12) :** La Figure 4.38 illustre le résultat de la cosimulation du module de convolution avec une précision de 78 %, tandis que la Figure 4.39 récapitule toutes les ressources utilisées.

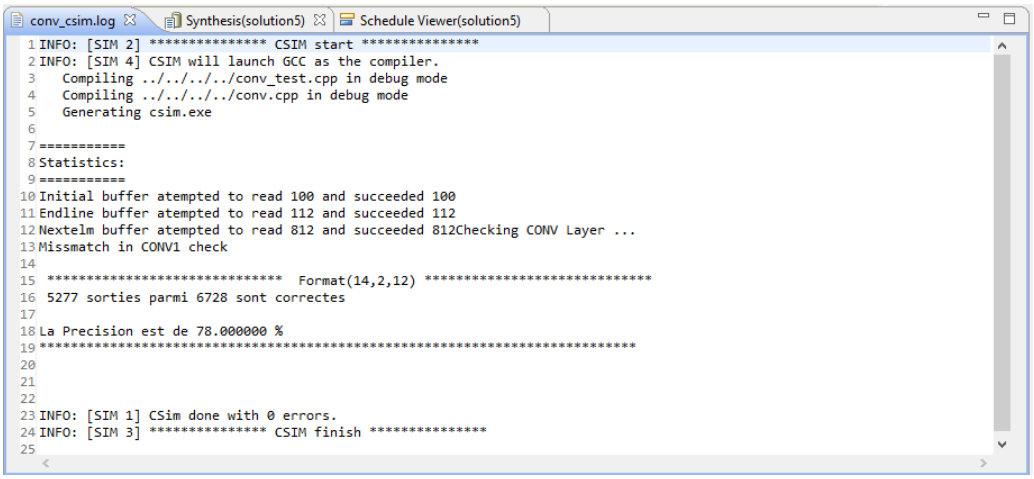
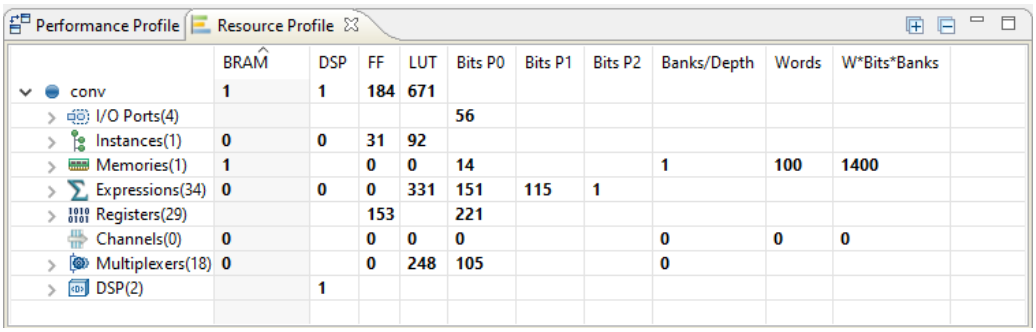


Figure 4.38: Cosimulation du module Convolution à base du format (14,2,12).



	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words	W*Bits*Banks
conv	1	1	184	671						
I/O Ports(4)					56					
Instances(1)	0	0	31	92						
Memories(1)	1		0	0	14			1	100	1400
Expressions(34)	0	0	0	331	151	115	1			
Registers(29)			153		221					
Channels(0)	0		0	0	0			0	0	0
Multiplexers(18)	0		0	248	105			0		
DSP(2)		1								

Figure 4.39: Profil de ressources à base du format (14,2,12).

La Figure 4.40 présente une représentation graphique de l'évolution de la précision en fonction de la dimension du format de représentation en virgule fixe, suite à l'exploration de la quantification sur le module de convolution. Cette visualisation permet de saisir rapidement l'impact de différentes dimensions de formats de virgule fixe sur la précision des résultats.

Par ailleurs, les Figures 4.41 et 4.42 offrent un aperçu détaillé des ressources utilisées dans chaque format exploré. Ces visualisations mettent en évidence la variation du nombre de bascules (flip-flops) et de tables de consultation (look-up tables) en fonction du format de virgule fixe utilisé. Cette analyse permet de comprendre comment les choix de représentation impactent non seulement la précision des résultats, mais aussi les ressources matérielles nécessaires pour les mettre en œuvre efficacement.

À la lumière des métriques d'analyse et de synthèse, le souci majeur de l'exploration de la taille optimale du format de virgule fixe à adopter pour la suite du processus

repose principalement sur la précision des résultats lors de la cosimulation. D'après la figure 4.40, on remarque qu'au-dessous du format 20, le module de convolution commence à perdre de sa précision. Par exemple, pour le format 19, on observe une précision de 98 %, tandis que pour le format 18, cette précision tombe à 89 %. Après avoir examiné en détail les différentes options disponibles et pris en considération les contraintes spécifiques associées à chacune d'elles, il est capital de prendre une décision éclairée concernant le format à adopter pour la suite de l'implémentation du modèle CNN en entier. Cette décision devrait être guidée par plusieurs facteurs, tels que la précision, la compatibilité avec les ressources disponibles, la facilité de mise en œuvre, les performances attendues et les exigences spécifiques du projet. Dans notre cas, nous avons accordé une importance particulière à la précision par rapport aux autres critères, car elle est fondamentale pour garantir la fiabilité et la qualité des résultats obtenus. Ainsi, nous avons choisi le format 19, qui offre une précision de 98%.

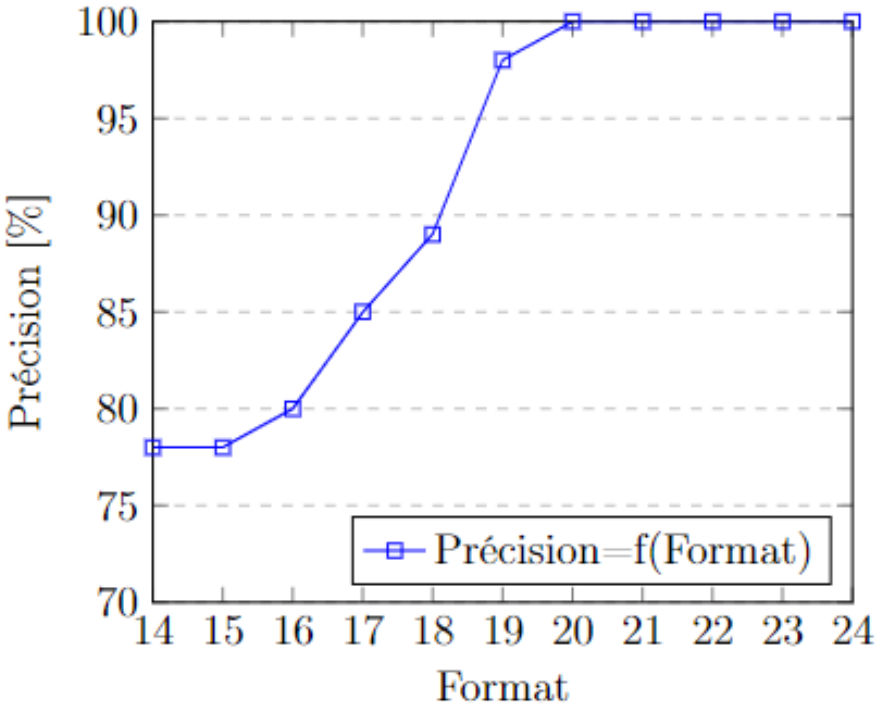


Figure 4.40: Évolution de la précision du module de convolution en fonction de la taille du format en virgule fixe.

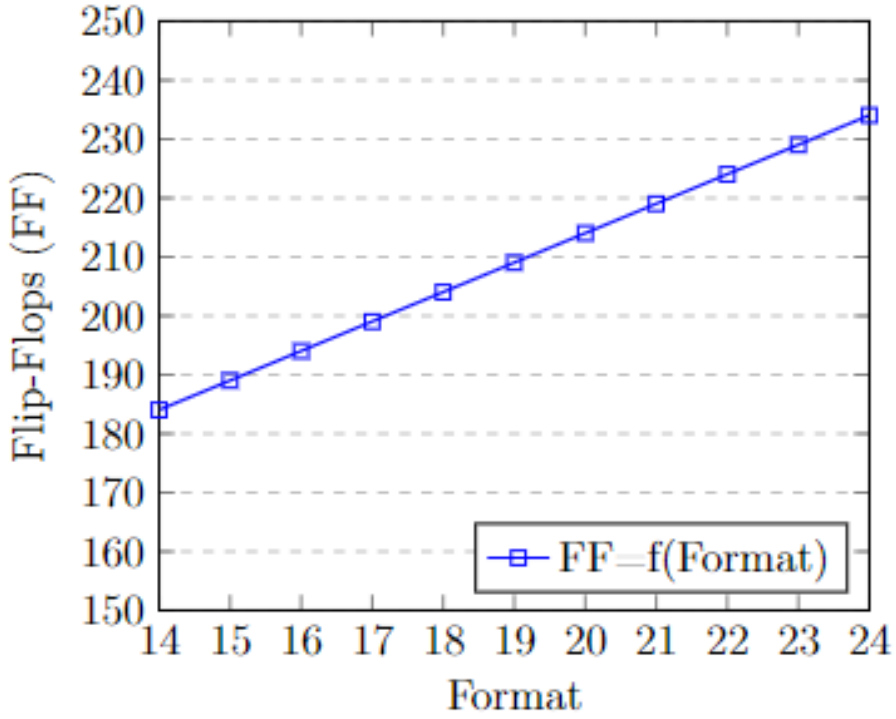


Figure 4.41: Évolution du nombre de flip-flops du module de convolution en fonction de la taille du format en virgule fixe.

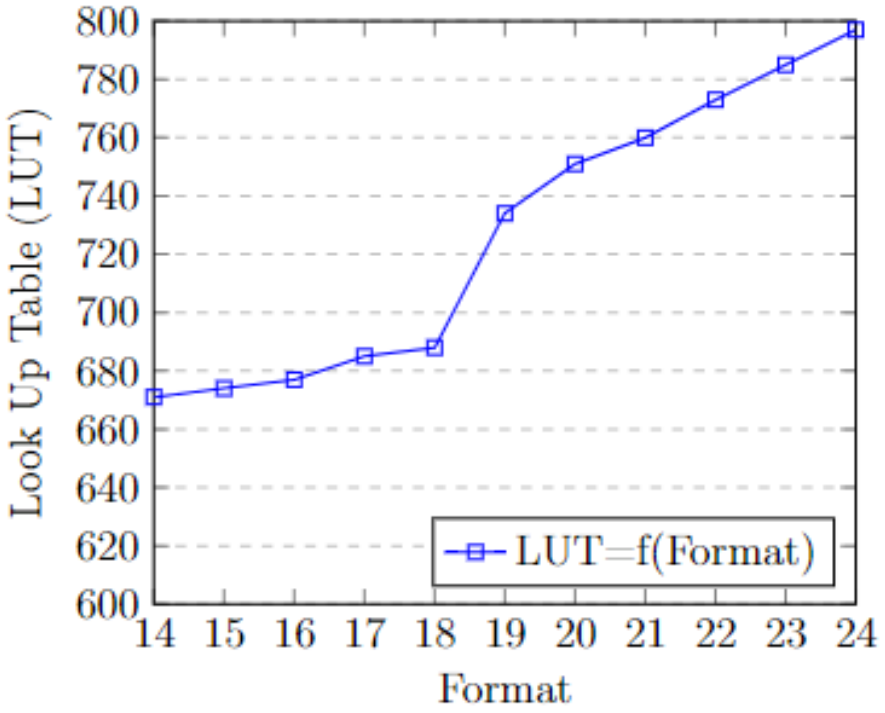


Figure 4.42: Évolution du nombre de Look Up Table du module de convolution en fonction de la taille du format en virgule fixe.

4.5.1.3 Module de mise en commun

La mise en œuvre matérielle du module de mise en commun, explicitement décrite dans la section 2.6.3, suit la méthode du maximum. En d'autres termes, pour chaque fenêtre carrée des cartes de caractéristiques d'entrée de dimension 4x4, la couche de mise en commun retient la valeur maximale. Cette approche garantit que, pendant le processus de regroupement, seule la caractéristique la plus significative au sein de chaque fenêtre est conservée, contribuant ainsi à la préservation des traits les plus Importants des cartes de caractéristiques d'entrée. D'autre part, en ce qui concerne la représentation des données, nous avons pris en compte le format de la virgule fixe optimisée qui a été utilisé dans l'implémentation matérielle du module de convolution, à savoir le format 19.

La validation du module de mise en commun a été réalisée en utilisant des données empiriques issues de la phase d'apprentissage. Ces données ont été intégrées dans le fichier de "Test-bench" requis pour la co-simulation. Cette approche a permis de tester le module dans des conditions réalistes, en utilisant les mêmes données que celles rencontrées lors de l'entraînement du réseau neuronal. Ainsi, la capacité du module à effectuer la mise en commun de manière efficace et précise a été rigoureusement évaluée.

Ainsi, les résultats de la Cosimulation, comme illustré dans la figure 4.43, démontrent une corroboration significative entre les données issues de l'opération de mise en commun et celles prédites lors de la phase d'apprentissage. Ces résultats confirment la fiabilité de l'implémentation du module de mise en commun et confirment également sa capacité à généraliser efficacement les données.

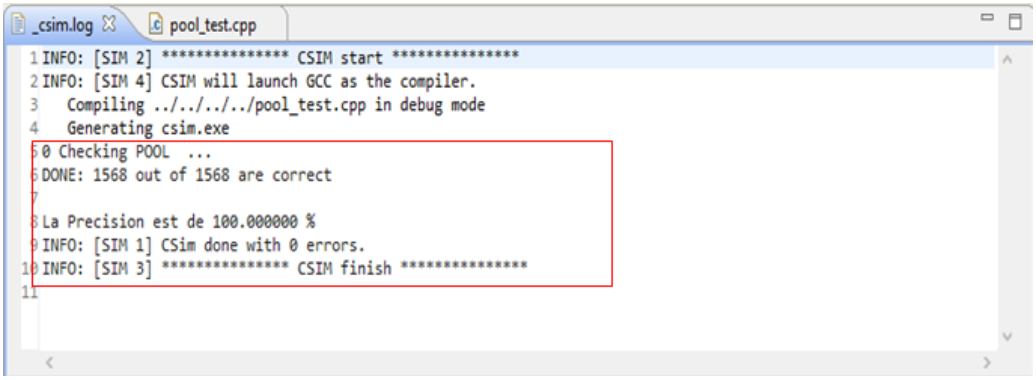


Figure 4.43: Résultat de la Cosimulation du module de mise en commun.

Les métriques synthétiques ont été utilisées pour générer l'interface du module de mise en commun. La Figure 4.44 illustre cette interface ainsi que ses différents signaux de contrôle et paramètres internes.

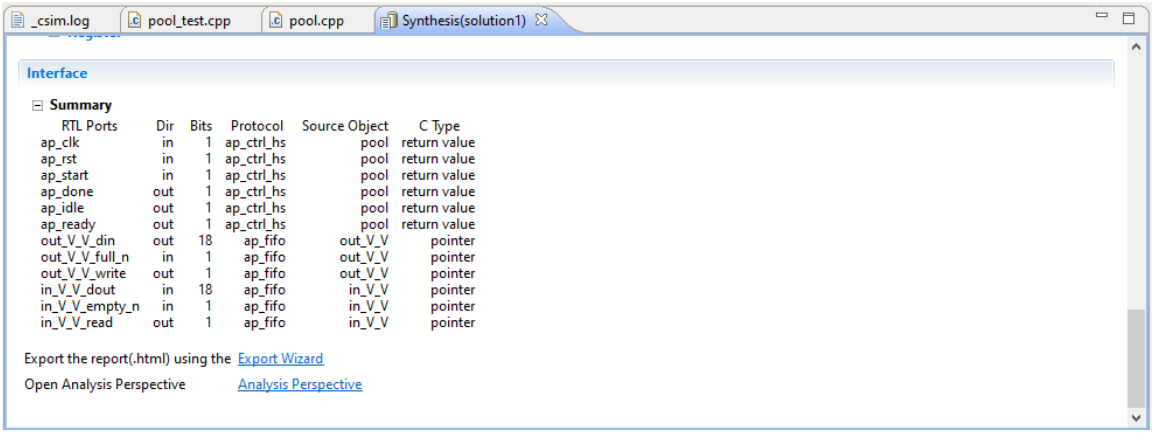
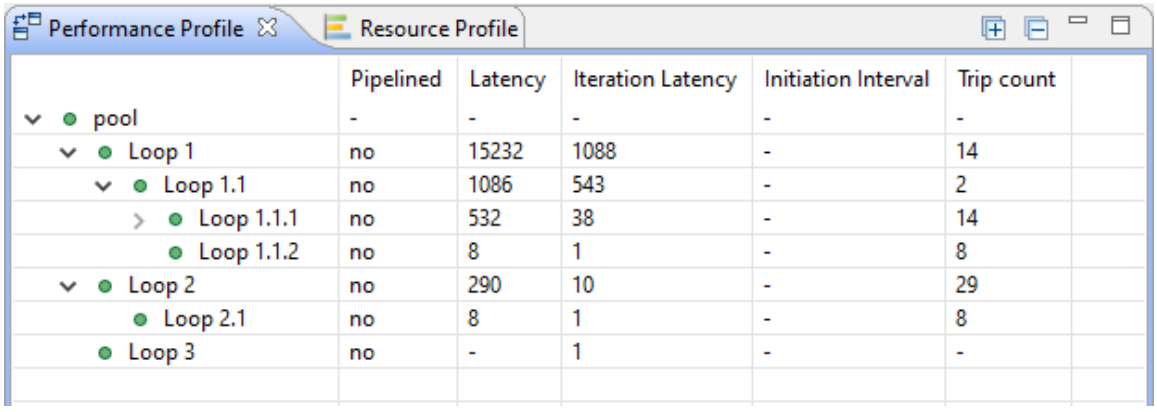


Figure 4.44: Interface du module mise en commun implémenté.

La figure 4.45 illustre les performances du module de mise en commun, en examinant la vitesse d'exécution ainsi que le nombre d'itérations nécessaires pour achever une tâche. Parallèlement, la figure 4.46 offre un aperçu détaillé de l'utilisation des ressources matérielles pendant le processus.



	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
pool	-	-	-	-	-
Loop 1	no	15232	1088	-	14
Loop 1.1	no	1086	543	-	2
Loop 1.1.1	no	532	38	-	14
Loop 1.1.2	no	8	1	-	8
Loop 2	no	290	10	-	29
Loop 2.1	no	8	1	-	8
Loop 3	no	-	1	-	-

Figure 4.45: Profil de Performances du module mise en commun.

Performance Profile | Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Words
pool	0	0	4108	1095					
I/O Ports(2)					36				
Instances(1)	0	0	0	561					
Memories(112)	0		4032	112	2016			112	112
Expressions(29)	0	0	0	279	94	110	36		
Registers(21)			76		79				
Channels(0)	0		0	0	0			0	0
Multiplexers(11)	0		0	143	32			0	
DSP(1)		0							

Figure 4.46: Profil de ressources du module mise en commun.

4.5.1.4 Module entièrement connecté

Le module de la couche entièrement connectée est conçu de manière à ce que chaque neurone de cette couche soit connecté à toutes les unités neuronales de la couche immédiatement précédente. La Figure 4.47 illustre l'interface du module entièrement connecté, laquelle a été produite en utilisant les métriques de synthèse de l'environnement Vivado HLS suite à la validation de la fonctionnalité. En complément, les Figures 4.48 et 4.49 illustrent respectivement les profils de performance (latence) et de ressources.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fc	return value
ap_rst	in	1	ap_ctrl_hs	fc	return value
ap_start	in	1	ap_ctrl_hs	fc	return value
ap_done	out	1	ap_ctrl_hs	fc	return value
ap_idle	out	1	ap_ctrl_hs	fc	return value
ap_ready	out	1	ap_ctrl_hs	fc	return value
out_V_din	out	32	ap_fifo	out_V	pointer
out_V_full_n	in	1	ap_fifo	out_V	pointer
out_V_write	out	1	ap_fifo	out_V	pointer
in_V_dout	in	32	ap_fifo	in_V	pointer
in_V_empty_n	in	1	ap_fifo	in_V	pointer
in_V_read	out	1	ap_fifo	in_V	pointer
weight_address0	out	17	ap_memory	weight	array
weight_ce0	out	1	ap_memory	weight	array
weight_q0	in	32	ap_memory	weight	array
bias_address0	out	7	ap_memory	bias	array
bias_ce0	out	1	ap_memory	bias	array
bias_q0	in	32	ap_memory	bias	array

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

Figure 4.47: Interface du module entièrement connectés implémenté.

La figure 4.45 met en évidence les performances du module entièrement connectés

en termes de vitesse d'exécution, nombre d'itérations, tandis que la figure 4.46 donne un aperçu de la manière dont les ressources matérielles sont utilisées.

Module Hierarchy

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
fc	-	1	5	709	854	624194	624195	none

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
fc	-	624194	-	624195	-
memset_output	no	119	1	-	120
Loop 2	no	623232	1082	-	576
Loop 2.1	no	1080	9	-	120
Loop 3	no	840	7	-	120

Figure 4.48: Profil de performance(latence) du module entièrement connectées.

Module Hierarchy

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
fc	-	1	5	709	854	624194	624195	none

Performance Profile | Resource Profile

	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	Banks/Depth	Wor
fc	1	5	709	854					
I/O Ports(4)					128				
Instances(3)	0	5	421	421					
Memories(1)	1		0	0	32			1	120
Expressions(16)	0	0	0	225	133	103	1		
Registers(16)			288		291				
Channels(0)	0		0	0	0			0	0
Multiplexers(10)	0		0	208	105			0	
DSP(3)		0							

Figure 4.49: Profil de ressources du module entièrement connectées.

4.5.1.5 Implementation du Modèle CNN LeNet5

Dans un modèle CNN, l'architecture est conçue comme une séquence de couches, chacune ayant un rôle spécifique dans le traitement des données. Ainsi, le choix de la structure sous-jacente qui assure la liaison entre ces différentes couches revêt une importance primordiale. A cet effet, la Figure 4.50 présente la configuration architecturale adoptée pour la mise en œuvre concrète du modèle CNN. Cette configuration, dénommée structure en flux de données, repose sur une séquence

organisée de tâches successives dans le but de créer une architecture propice au traitement parallèle. Cette approche vise à optimiser le débit tout en minimisant la latence.

Dans cette structure architecturale du modèle CNN, les tampons de données en sortie de chaque couche, également appelés buffers, génèrent une quantité suffisante de données pour permettre un traitement immédiat par la couche suivante, éliminant ainsi le besoin d'attente. À mesure que les données sont extraites du tampon, elles sont instantanément remplacées par de nouvelles données. Il est cependant essentiel de noter que cette synchronisation entre les couches est spécifiquement applicable aux couches de convolution et de mise en commun. Les couches entièrement connectées ne peuvent pas bénéficier de cette synchronisation, car elles nécessitent le traitement total de l'ensemble des données provenant des couches précédentes. Cependant, il convient de souligner que cette contrainte n'affecte nullement la latence, car ces couches ne contribuent qu'à une fraction minime, représentant moins de 1%, de la charge de calcul totale.

Comme toute interface implémentée sur FPGA, celle du modèle CNN LeNet5 nécessité des signaux également de commande tant à l'entrée qu'à la sortie du modèle CNN, ainsi que les signaux de communication "Handshake" entre les couches, pour assurer une synchronisation efficace des transferts de données inter-couches.

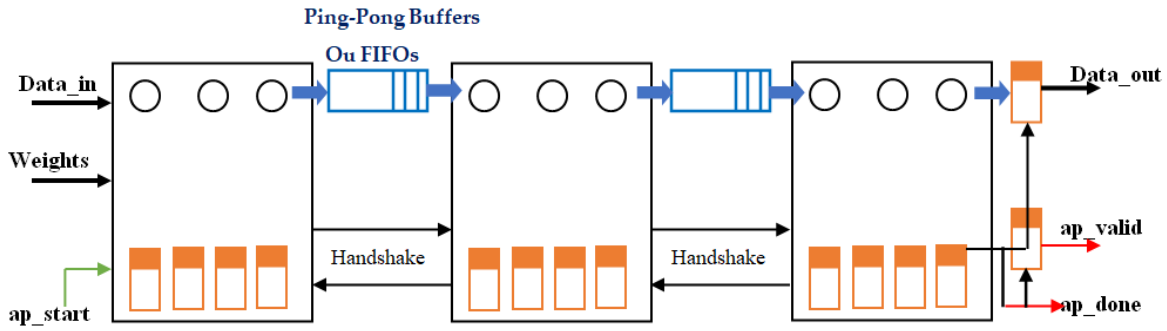


Figure 4.50: Architecture matérielle à base de flux de données.

Pour mettre en avant l'avantage de l'architecture en flux de données adoptée dans cette approche par rapport à une architecture séquentielle, nous avons mis en œuvre les deux architectures sans apporter aucune optimisation. Par la suite, en ayant recours aux outils d'analyse et de synthèse du HSL, nous avons généré des métriques de performance pour les deux architectures. Les Figures 4.51 et 4.52 présentent

respectivement les métriques de l'architecture séquentielle et de l'architecture en flux de données. L'architecture en flux de données est caractérisée par la mention ***dataflow*** sur la colonne métrique ***Pipeline type***. nous observons également sur la colonne métrique ***Latency*** une latence plus élevée dans l'architecture séquentielle en raison du traitement séquentiel des tâches. En revanche, l'architecture en flux de données permet de réduire considérablement la latence grâce à un traitement parallèle, où les tâches peuvent être effectuées simultanément. Cette démarche nous a permis de confronter les performances des deux architectures et de renforcer notre décision initiale.

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
lenet5	-	144	9	37826	17523	1555731~1570211	1555732 ~ 1570212	none
conv2D_layer3	-	45	1	15044	3589	415186~415826	415186 ~ 415826	none
conv2D_layer2	-	3	1	11929	3470	581854~587854	581854 ~ 587854	none
conv2D_layer1	-	1	1	4390	3072	299028~306868	299028 ~ 306868	none
pool_layer1	-	0	0	2761	2949	12274	12274	none
pool_layer2	-	0	0	3123	2841	5077	5077	none
fc_layer1	-	81	2	176	363	208753	208753	none
fc_layer2	-	12	2	170	361	30649	30649	none
fc_layer3	-	1	2	145	330	1869	1869	none

Figure 4.51: Métriques de performance de l'architecture séquentielle.

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
lenet5	-	144	9	35589	28596	586548~592548	581855 ~ 587855	dataflow
pool_layer2	-	0	0	2006	8911	2306	2306	none
pool_layer1	-	0	0	1651	8087	5107	5107	none
conv2D_layer3	-	45	1	15045	3600	415186~415826	415186 ~ 415826	none
conv2D_layer2	-	3	1	11928	3480	581854~587854	581854 ~ 587854	none
conv2D_layer1	-	1	1	4391	3083	299028~306868	299028 ~ 306868	none
fc_layer1	-	81	2	177	372	208753	208753	none
fc_layer2	-	12	2	171	370	30649	30649	none
fc_layer3	-	1	2	146	339	1869	1869	none
Loop_lenet5_label0_p	-	1	0	18	104	1026	1026	none

Figure 4.52: Métriques de performance de l'architecture en flux de données.

Pour le reste de cette de l'implémentation, l'architecture en flux de données est adoptée comme base sur laquelle viennent s'appliquer différentes options d'optimisation. Ces dernières concernes le partitionnement des matrices, le pipelining des fonctions ainsi que l'enroulement *(Unrolling)* des boucles et l'efficace configuration du stockage et de l'accès aux données à travers une hiérarchie de niveaux de mémoire, comprenant notamment la mémoire externe (DRAM) *(Dynamic Random*

Access Memory), la mémoire interne (BRAM) *(Block Random Access Memory)* et les registres locaux.

4.5.1.6 Quantification des données

La représentation des données revêt une importance capitale dans les applications d'implémentation matérielle sur FPGA, en particulier en raison de l'impact direct de la taille des données sur les ressources disponibles et les performances du système. La manière dont les données sont encodées et stockées peut avoir des répercussions significatives sur l'efficacité de l'utilisation des ressources matérielles, y compris la capacité mémoire et les (DSP) *(Digital Signal Processor)*, la bande passante mémoire, ainsi que la vitesse d'exécution des opérations.

De nombreux travaux d'implémentation des CNN se sont concentrés sur le choix approprié de la quantification des données pour représenter les images, les poids et les biais. Par exemple, Chun Yan Lo et Chiu-Wing Sham [84] ont démontré qu'une implémentation du modèle LeNet5 en virgule fixe sur 8 bits présente une précision de 99,55%, tandis qu'une implémentation améliorée en virgule fixe sur 4 bits atteint une précision de 99,28%, avec moins de 0,3% de perte d'exactitude. Cependant, des réserves peuvent être émises concernant ces résultats, car selon notre exploration de la quantification menée dans la section 4.5.1, une représentation en 19 bits maintient une précision de 98%.

4.5.1.7 Optimisation de l'implémentation du modèle CNN

Durant cette étape d'implémentation et d'amélioration, l'objectif principal est de mettre en œuvre diverses techniques d'optimisation, parmi lesquelles le partitionnement, le pipelining et le parallélisme. Il convient de noter que la plupart de ces optimisations sont traditionnellement ciblées sur la couche de convolution, considérée comme étant à la fois centrale et intensément calculatoire dans les architectures de réseaux neuronaux profonds.

L'algorithme 4.1 expose les emplacements de chaque directive d'optimisation au sein des différentes boucles et fonctions imbriquées. Cependant, afin d'évaluer l'impact de chaque directive sur les performances de l'implémentation, nous avons choisi d'adopter une approche progressive en les appliquant graduellement,voir le Tableau 4.1. En utilisant les métriques d'analyse et de synthèse de Vivado HLS,

nous effectuons une analyse approfondie à partir de laquelle nous déciderons quelles directives maintenir, c'est-à-dire celles qui offrent un équilibre optimal entre la performance et les ressources disponibles.

Algorithme 4.1 : Directives d'optimisation appliquées sur le processus de convolution

```
Directive de partitionnement des matrices sur les données et les paramètres
 #pragma HLS array_partition
Data : lecture ← #Data_Weights_Bias
Directive de pipelining de la fonction des boucles imbriquées
#pragma HLS pipeline
while no ≤ NoF do
    Directive d'enroulement pour la "Boucle 4"
    #pragma HLS unroll
    while y ≤ Noy do
        while x ≤ Nox do
            Directive d'enroulement pour la "Boucle 3"
            #pragma HLS unroll
            while ni ≤ NiF do
                Directive d'enroulement pour la "Boucle 2"
                #pragma HLS unroll
                while j ≤ Ky do
                    while i ≤ Kx do
                        Directive d'enroulement pour la "Boucle 1"
                        #pragma HLS unroll
                    end
                end
            end
        end
    end
end
```

Les étapes d'optimisation séquentielle ajoutées au modèle CNN LeNet5 sont présentées dans le tableau 4.1. Ce tableau offre une définition claire de toutes les options d'optimisations architecturales adoptées. Il est également important de souligner que l'élaboration de cette implémentation repose sur le concept de **Dataflow**, qui adopte une architecture en flux de données. Ensuite, pour chaque nouveau design, des améliorations supplémentaires ont été apportées.

Tableau 4.1: Optimisation Séquentielle : Un Cheminement à Travers les Améliorations Successives.

	Enroulement	Pipeline	Enroulement	Pipeline	Enroulement	Pipeline	Enroulement	Pipeline	Partition Matrice	Structure Séquentielle	Structure Dataflow
	Boucle$_1$		**Boucle$_2$**		**Boucle$_3$**		**Boucle$_4$**				
Baseline										✓	
Dataflow											✓
Design$_1$									✓		✓
Design$_2$	✓								✓		✓
Design$_3$	✓	✓							✓		✓
Design$_4$	✓	✓	✓	✓					✓		✓
Design$_5$	✓	✓	✓	✓	✓	✓			✓		✓
Design$_6$	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓

(1) **Design_1 :** Dans le cadre de notre première démarche d'optimisation séquentielle, nous avons initié le processus dans cette option de design en procédant au partitionnement des matrices concernées.

(2) **Design_2 :** Lors de la phase d'implémentation de ce design, une directive de parallélisation a été intégrée à la boucle_1 de la couche de convolution. Parallèlement, la directive de partitionnement des matrices a été conservée à l'ensemble des données entrantes dans le modèle CNN.

(3) **Design_3 :** Pour cette option, on a pris le design_2 de base et on y a ajouté une deuxième méthode de pipelining, centrée sur la Boucle_1. Cette orientation découle d'une évaluation approfondie de cette boucle.

(4) **Design_4 :** Pour cette option architecturale, nous avons conservé les optimisations du design_3, auxquelles nous avons ajouté une seconde les directives de parallélisation et de pipelining sur la Boucle_2.

(5) **Design_5 :** Pour cette option architecturale, nous avons également conservé les optimisations du design_4, auxquelles nous avons ajouté une seconde les directives de parallélisation et de pipelining sur la Boucle_3.

(6) **Design_6 :** Pour cette dernière option, nous avons suivi la même approche

que précédemment, en conservant les directives antérieures et en y ajoutant celles de parallélisation et de pipelining spécifiquement sur la Boucle_4.

4.6 Résultats et interprétation

L'utilisation des metriques d'analyse, de synthese et de copmaraison de solutions disponibles dans l'environnement Vivado HLS nous a permis de générer une gamme de rapports, incluant des analyses, des synthèses et des comparaisons. Ces rapports ont joué un rôle essentiel dans l'acquisition d'une compréhension approfondie de chaque option de conception, nous permettant ainsi de prendre des décisions éclairées en tenant compte de leurs avantages et inconvénients respectifs.

Ainsi et à la suite de l'analyse de ces rapports, une conclusion a été tirée en se basant sur une analyse statistique, comme illustré dans les Figures 4.53, 4.54, et 4.55 . Les données présentées révèlent que les conceptions 4, 5 et 6 surpassent les autres en termes de latence et d'utilisation des ressources matérielles. Bien que les conceptions 6 et 7 présentent une latence supérieure, elles requièrent néanmoins davantage de ressources par rapport à la conception 4. Par conséquent, lors du choix d'une option de conception, il est crucial de considérer l'équilibre entre la latence et l'utilisation des ressources. Malgré les améliorations en termes de latence pour les conceptions 5 et 6, la conception 4 demeure la plus efficace globalement pour répondre à nos exigences.

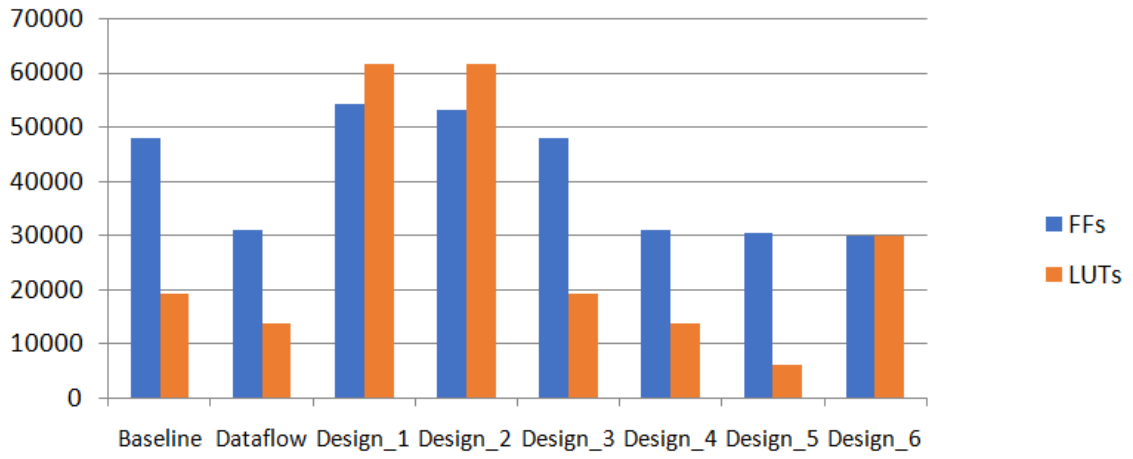


Figure 4.53: Estimations d'utilisation des bascules (Flip Flops) et des LUTs (Look Up Tables).

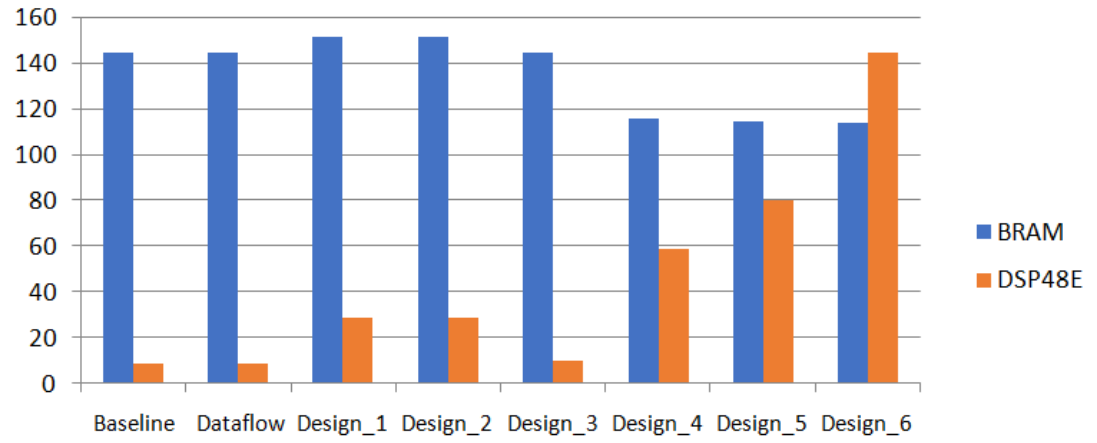


Figure 4.54: Estimations d'utilisation des BRAMS (Bloc RAMS) et des DSP48E.

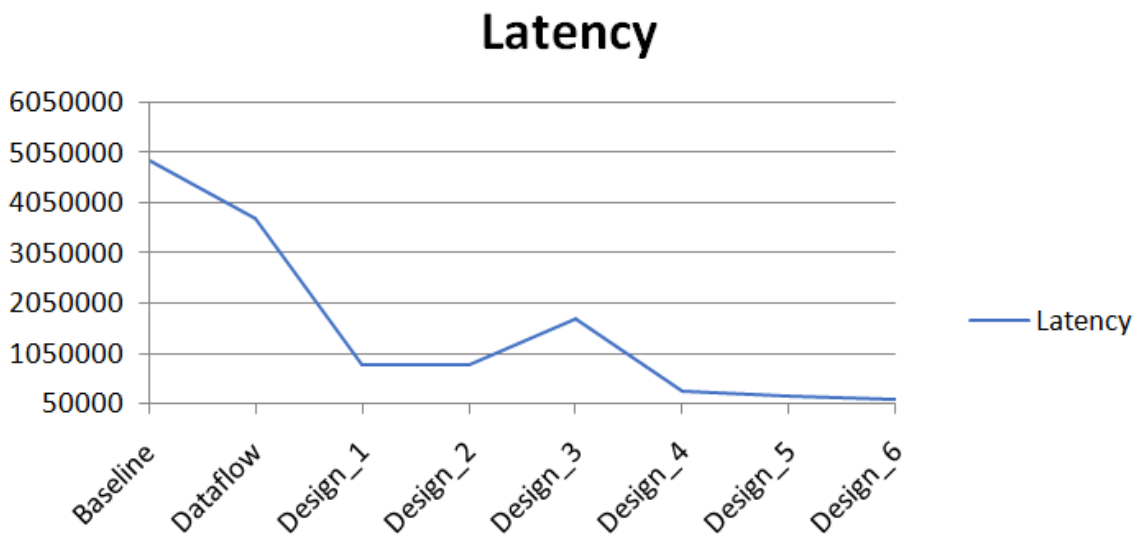


Figure 4.55: Analyse de l'évolution de la latence selon les options architecturales adoptées.

4.7 Étude comparative et Validation

Une étude comparative approfondie a été menée tableau 4.2, analysant trois travaux antérieurs [85] , [86] , [87], ainsi que notre propre approche, en se fondant sur cinq paramètres fondamentaux : le modèle CNN sélectionné, la plateforme de développement utilisée, le schéma de quantification, la fréquence de fonctionnement (en MHz) et la latence. Tous ces travaux se concentrent sur l'architecture reconnue du modèle CNN LeNet 5 et opèrent à une fréquence de 100 MHz. Les divergences entre ces études résident principalement dans les choix respectifs de plateforme et

de format de quantification, ainsi que dans les performances de latence obtenues. Par exemple, le travail [85] a exploité la plateforme ZC706, avec une quantification Fixed25, affichant une latence de 3 millisecondes. En contraste, le travail [86] a préféré la plateforme ZC709, avec une quantification Fixed16, et a atteint une latence de 1.318 millisecondes. Notamment, le travail [87], basé sur la plateforme ZC706 et une quantification Fixed16, a réussi à réduire considérablement la latence à 157.7 microsecondes. En comparaison, dans notre propre recherche, la plateforme ZC707 et une quantification Fixed19 ont été utilisées, résultant en une latence légèrement plus élevée de 299.9 microsecondes. Cette différence de latence entre le travail [87] et le notre peut principalement être attribuée au choix de la quantification. En effet, une quantification plus précise, telle que celle employée dans notre étude avec Fixed19, engendre une représentation plus détaillée des données, impactant ainsi la latence par le biais d'une manipulation plus complexe des informations. À l'inverse, une quantification moins précise, comme celle adoptée dans le travail [87] avec Fixed16, peut offrir des calculs plus rapides, bien que moins précis. En conclusion, le processus de quantification joue un rôle crucial dans les performances de latence des implémentations sur FPGA, en balançant le compromis entre précision et rapidité de traitement.

Tableau 4.2: Étude comparative avec des travaux antérieures

	[85]	[86]	[87]	[Notre]
Modèle CNN	LeNet5	LeNet5	LeNet5	LeNet5
Plateforme FPGA	ZC706	ZC709	ZC706	ZC707
Quantification (virgule Fixe)	25	16	16	19
Fréquence (MHz)	100	100	100	100
Latence	3ms	1.318ms	175.7μs	299.3μs

4.8 Conclusion

L'intégration de l'inférence des CNN sur des plateformes matérielles a suscité un intérêt croissant au sein de la communauté scientifique, se manifestant par une multitude de méthodologies et d'approches d'optimisation. Ces initiatives visent à accélérer les processus d'inférence tout en minimisant les coûts énergétiques associés.

Dans le présent chapitre, une exploration méticuleuse de l'implémentation du mo-

dèle Convolutional Neural Network (CNN) LeNet5 a été entreprise. Notre approche a principalement reposé sur l'utilisation de l'environnement de développement Vivado HLS, lequel a joué un rôle fondamental dans la conceptualisation et la réalisation de cette implémentation. Les outils sophistiqués de développement disponibles dans Vivado HLS, incluant les directives d'optimisation, ainsi que les métriques analytiques, de synthèse intégrées et de comparaison de rapports, ont joué un rôle décisif dans la structuration méthodique du processus d'optimisation. Chaque étape d'optimisation a été soigneusement décomposée et évaluée, permettant ainsi de formuler des conclusions significatives sur les orientations stratégiques à envisager pour la prochaine étape du processus d'optimisation.

Cette approche rigoureuse a non seulement facilité la compréhension profonde de la performance de l'implémentation, mais a également jeté les bases d'améliorations continues et ciblées, essentielles à l'évolution et à l'optimisation continues du modèle LeNet5 dans le contexte de notre projet de recherche.

Le chapitre à suivre représente une progression naturelle de notre recherche antérieure, mettant en lumière une méthode novatrice basée sur les matrices systoliques. Cette approche révolutionnaire ambitionne d'améliorer encore davantage les performances du modèle CNN en capitalisant sur les atouts intrinsèques des architectures matricielles systoliques, en particulier en ce qui concerne le parallélisme et la réutilisation des données.

Chapitre 5

Implémentation utilisant le registre de transfert logique : approche et méthodologie.

5.1 Introduction

Dans ce chapitre consacré à une seconde partie expérimentale, nous entreprenons une analyse approfondie de deux aspects interdépendants. Le premier aspect porte sur l'optimisation des opérations de convolution et la réutilisation des données en adoptant l'approche basée sur les matrices systoliques discutées dans la section 3.3.2.3. Cette méthode vise à améliorer l'efficacité des calculs en exploitant le parallélisme et en réduisant les transferts de données, ce qui favorise une utilisation plus efficiente des ressources matérielles disponibles. Le second aspect se concentre sur l'utilisation du RTL pour mettre en œuvre cette approche. Cette démarche implique une conception au niveau des transferts de registres, offrant ainsi une granularité plus fine dans la description matérielle et permettant une optimisation supplémentaire des performances.

Au cours de cette partie expérimentale, notre focalisation est spécifiquement dirigée vers la couche de convolution, celle-ci jouant un rôle prépondérant, constituant plus de 90 % des opérations au sein des modèles CNNs.

Les efforts d'optimisation des couches de convolution, basés sur les matrices systoliques, décrits dans la littérature [17, 88–90] , poursuivent un double objectif.

D'une part, ils cherchent à paralléliser les opérations de multiplication, d'addition et d'accumulation au sein des couches de convolution, ce qui entraîne une accélération significative du traitement. D'autre part, ils s'efforcent de maximiser la réutilisation des données afin de minimiser les accès à la mémoire, réduisant ainsi la consommation énergétique. Ces travaux visent ainsi à améliorer l'efficacité des opérations de convolution en tirant parti du parallélisme et de l'optimisation des accès mémoire, ouvrant ainsi la voie à des solutions plus rapides et plus efficaces pour le traitement des données dans divers domaines d'application.

5.2 Méthodologie d'implémentation

La Figure 5.1 montre comment fonctionne l'approche avec les matrices systoliques, où chaque bloc de base est appelé un PE.

La façon dont ce PE est construit à l'intérieur dépend du type de systolique utilisé. Dans notre exemple, nous avons décrit deux types : les poids stationnaires et les sorties stationnaires. De plus, nous avons inclus un pseudo-code simple en Verilog à titre illustratif pour l'architecture des sorties stationnaires du PE. Il est important de noter que c'est cette architecture que nous avons utilisée dans nos travaux.

Dans notre étude, nous avons choisi d'utiliser une convolution simple avec un noyau de taille 3x3, comme celle couramment utilisée dans la plupart des modèles CNNs. Cela signifie que nous aurons besoin de neuf (9) éléments de processus (PEs) et de neuf (9) tampons de ligne ou mémoires tampons. Ces tampons seront répartis en trois groupes : trois pour stocker les poids du noyau de convolution, trois pour stocker les données d'entrée ou les caractéristiques d'entrée (Input Feature Maps), et trois pour stocker les sorties. Il est important de noter que les tampons de lignes en entrée servent à introduire les poids et les données d'entrée de manière systolique, c'est-à-dire de façon synchronisée et continue, ce qui facilite le traitement parallèle et efficace des informations dans la matrice systolique. En ce qui concerne le module PE, il est principalement conçu pour effectuer trois opérations importantes. Tout d'abord, il multiplie la valeur du noyau (kernel) par celle des données d'entrée de la fenêtre sur laquelle la convolution est appliquée. Ensuite, il ajoute ce résultat à d'autres calculs. Enfin, il accumule ce résultat avec celui provenant du PE voisin.

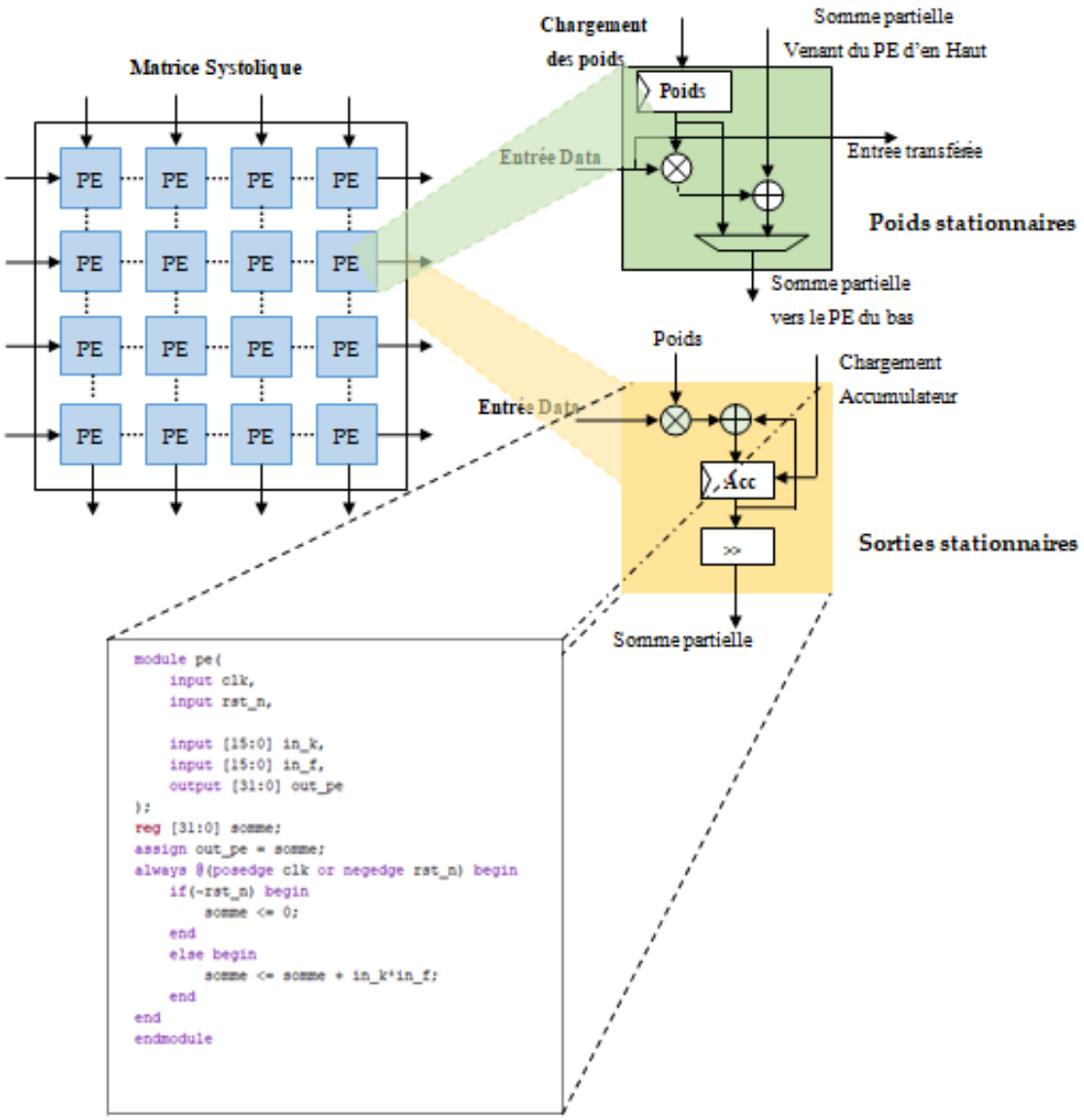


Figure 5.1: Architecture de matrice systolique basée sur les types de poids et de sortie stationnaires.

5.2.1 Résultats et interprétations

Une fois le développement du module de convolution terminé, il a été soumis à plusieurs phases de test. Ces étapes comprenaient la simulation, où le fonctionnement du module était évalué virtuellement ; l'analyse, où les résultats de la simulation étaient examinés en détail ; la synthèse, qui impliquait la génération des structures matérielles correspondantes ; et enfin, l'implémentation, où le module était intégré dans le système final. Chaque phase était essentielle pour assurer le bon fonction-

nement du module de convolution et son adéquation aux besoins spécifiques de l'application.

La Figure 5.2 offre un aperçu de la hiérarchie des différents composants, tels que les éléments de processus (PEs) et les tampons de ligne, que nous avons conçus pour réaliser l'opération de convolution 3x3. En outre, elle synthétise les diverses ressources utilisées par chaque module, offrant ainsi une vision claire de la répartition des ressources au sein de notre architecture. Par ailleurs, la Figure 5.3 fournit un résumé des ressources consommées par chaque module ainsi que le pourcentage d'utilisation par rapport à celles disponibles sur la plate-forme FPGA utilisée. Ce qui ressort de ce rapport, c'est le taux élevé des Entrées/Sorties, en raison des nombreuses lignes de contrôle nécessaires à la gestion des opérations et au cheminement des données vers et depuis le module de convolution.

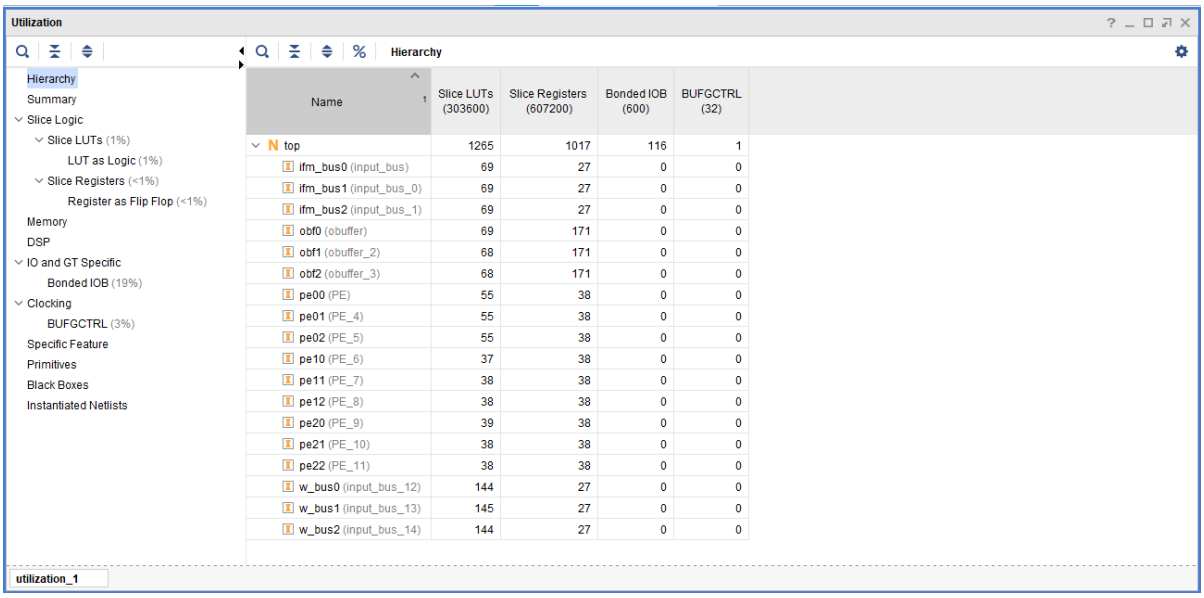


Figure 5.2: Hiérarchie des modules et les ressources internes qui composent la convolution 3x3.

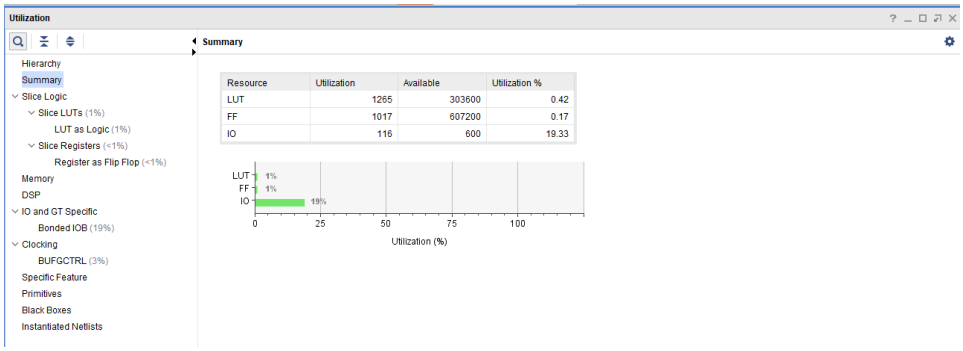


Figure 5.3: Sommaire des ressources utilisées.

La phase de simulation du module de convolution, mise en œuvre selon l'approche systolique, a conduit à l'élaboration d'un schéma logigramme complet décrivant en détail les différents processus de transfert de données, les opérations de convolution telles que la multiplication et l'accumulation, ainsi que le stockage des résultats au niveau des buffers de sortie. Afin d'assurer une compréhension claire et une lisibilité optimale du schéma, celui-ci a été organisé en deux parties distinctes.

La première partie du schéma logigramme, présentée dans la figure 5.4, se concentre sur le transfert systolique des données d'entrée (input feature maps) ainsi que sur les poids des filtres. Cette section offre une analyse exhaustive du flux de données et des opérations réalisées à chaque étape du processus de convolution, offrant ainsi une vision détaillée du fonctionnement de l'algorithme systolique.

Lorsque les lignes d'activation relatives aux caractéristiques d'entrée ainsi qu'aux poids sont déclenchées, un processus de transfert systolique est initié. Ce mode de transfert, caractérisé par sa nature séquentielle, guide les caractéristiques d'entrée de la première ligne, ainsi que leurs poids correspondants, vers les bus d'entrée appropriés. Cette phase initiale est ensuite suivie par le transfert des caractéristiques d'entrée de la deuxième ligne, avec un léger décalage temporel d'un cycle d'horloge. Enfin, le processus se termine par le transfert des caractéristiques d'entrée de la troisième ligne, également avec un décalage d'un cycle d'horloge par rapport à la deuxième ligne.

La seconde et la troisième partie du logigramme, comme illustrée dans les figure 5.5 et 5.6 mettent en lumière la phase de récupération des résultats finaux issus de l'opération de convolution. Ces résultats sont ensuite stockés dans des buffers de sortie spécifiquement conçus à cet effet. Tant que les processeurs élémentaires (PEs) n'ont pas achevé leurs opérations et que les résultats n'ont pas été acheminés vers les buffers de sortie, la ligne de contrôle **"empty"** demeure active tandis que celle de **"full"** demeure inactive. Une fois les opérations terminées au niveau des PEs, la ligne de contrôle **"full"** devient active, signalant ainsi que les buffers de sortie sont pleins. Ces buffers peuvent alors transférer les résultats vers la couche suivante.

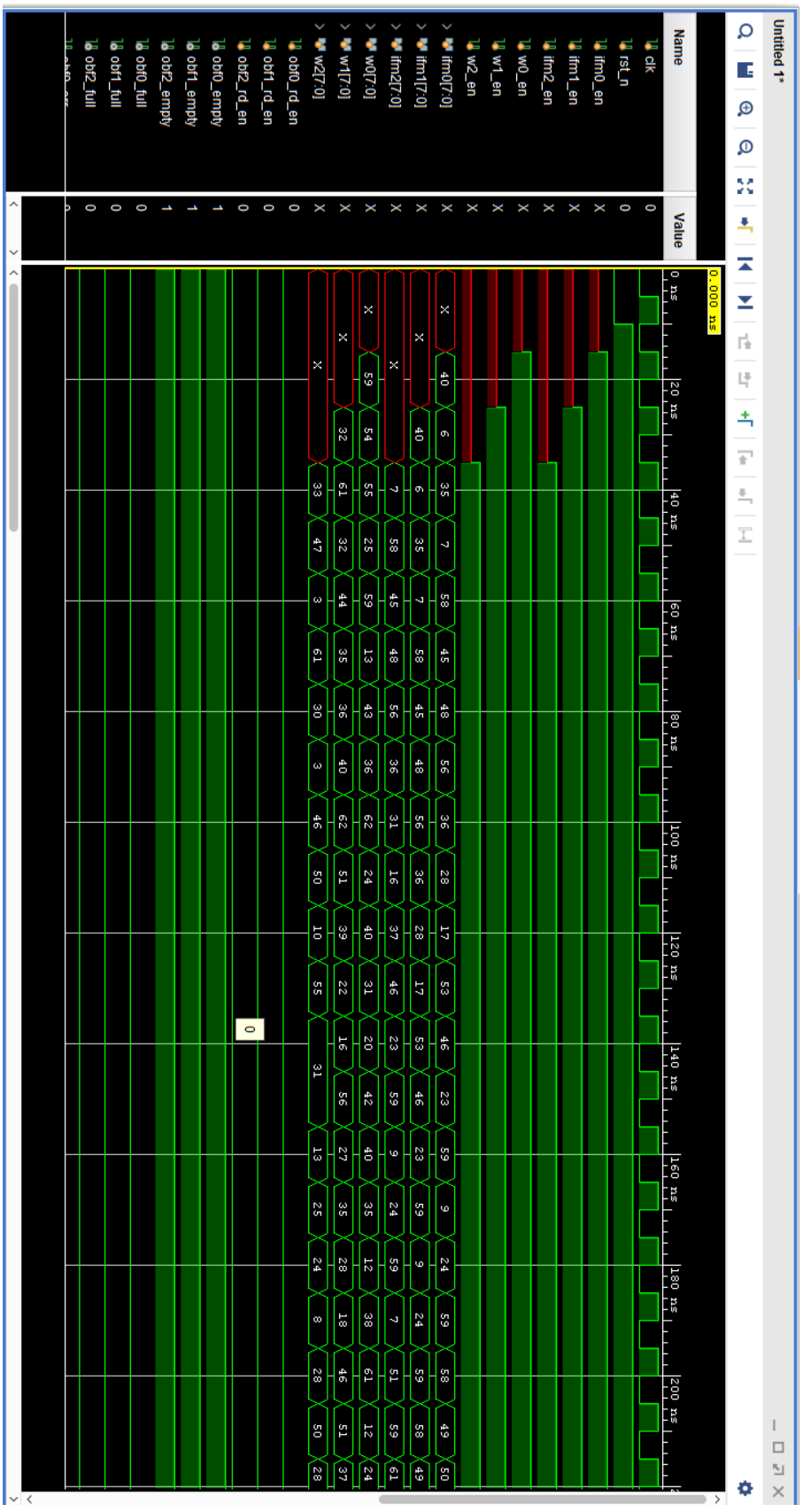


Figure 5.4: Logigramme de l'implémentation du module de convolution dans un contexte de traitement systolique (Partie 1).

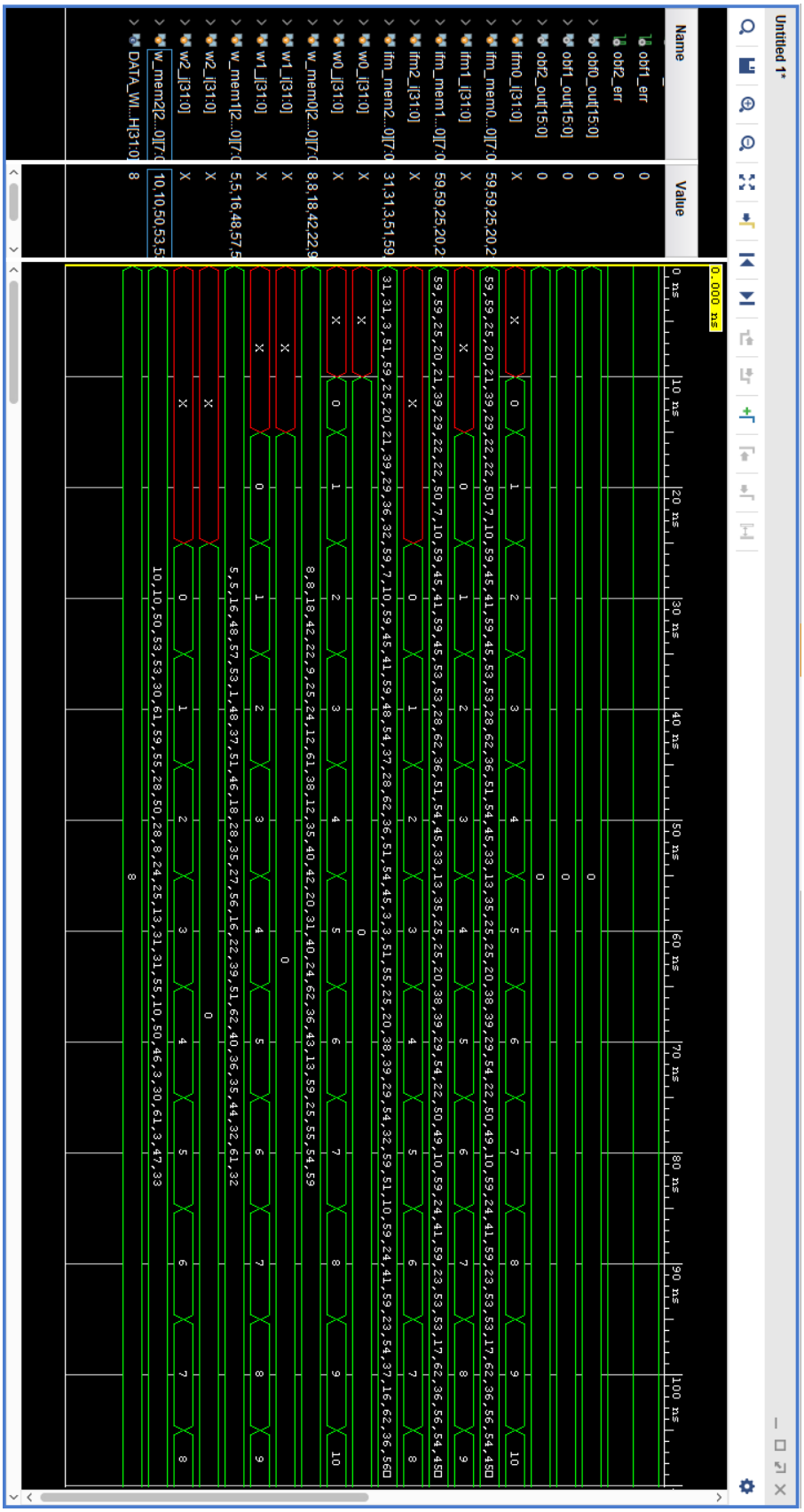


Figure 5.5: Logigramme de l'implémentation du module de convolution dans un contexte de traitement systolique (Partie 2).

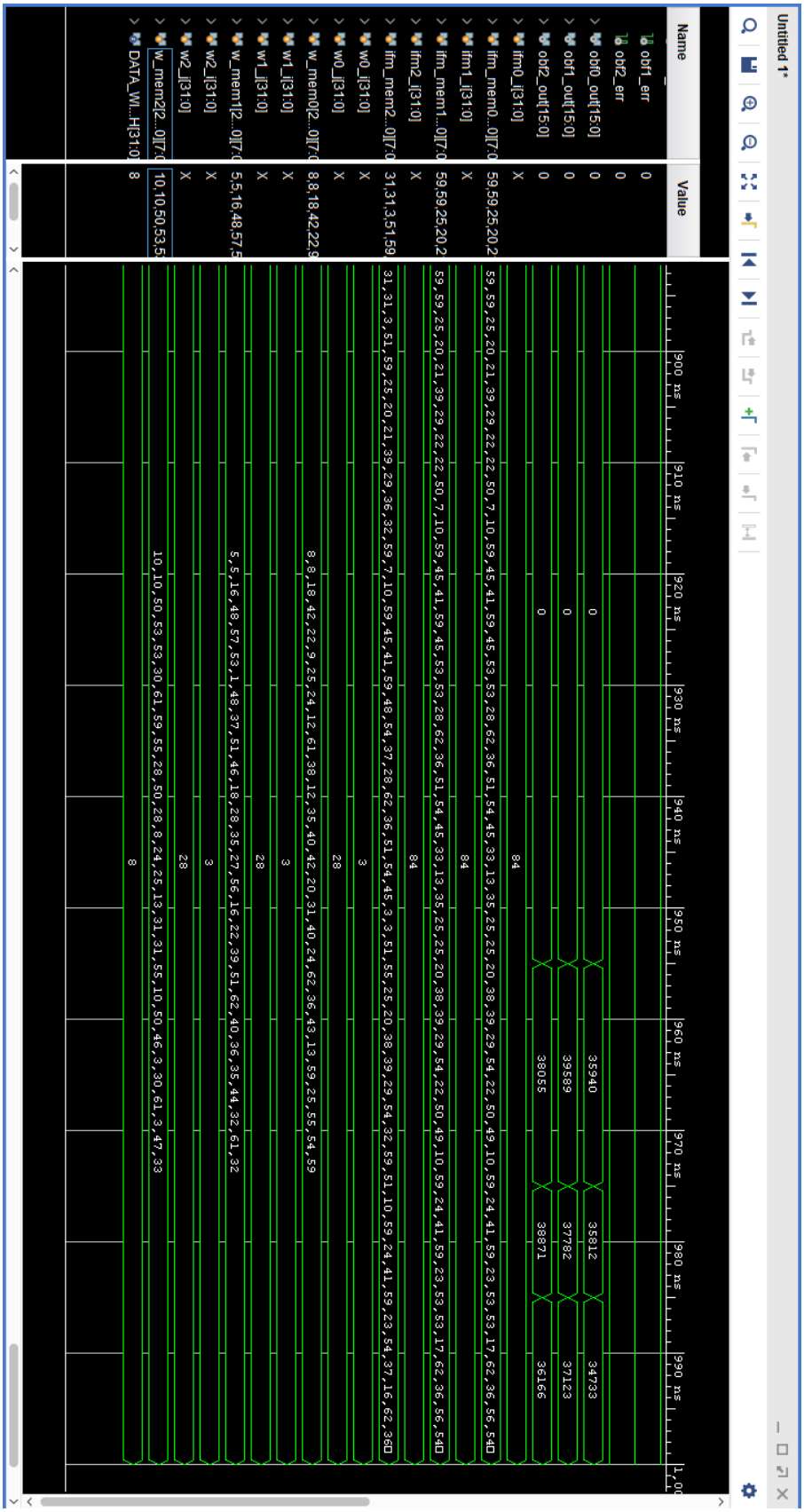


Figure 5.6: Logigramme de l'implémentation du module de convolution dans un contexte de traitement systolique(Partie 3).

Le schéma de câblage du module de convolution, qui met en lumière la connectivité entre les différents blocs, notamment les neuf buffers d'entrée, les neuf PEs, ainsi que les trois buffers de sortie, est présenté de manière détaillée dans la figure 5.7.

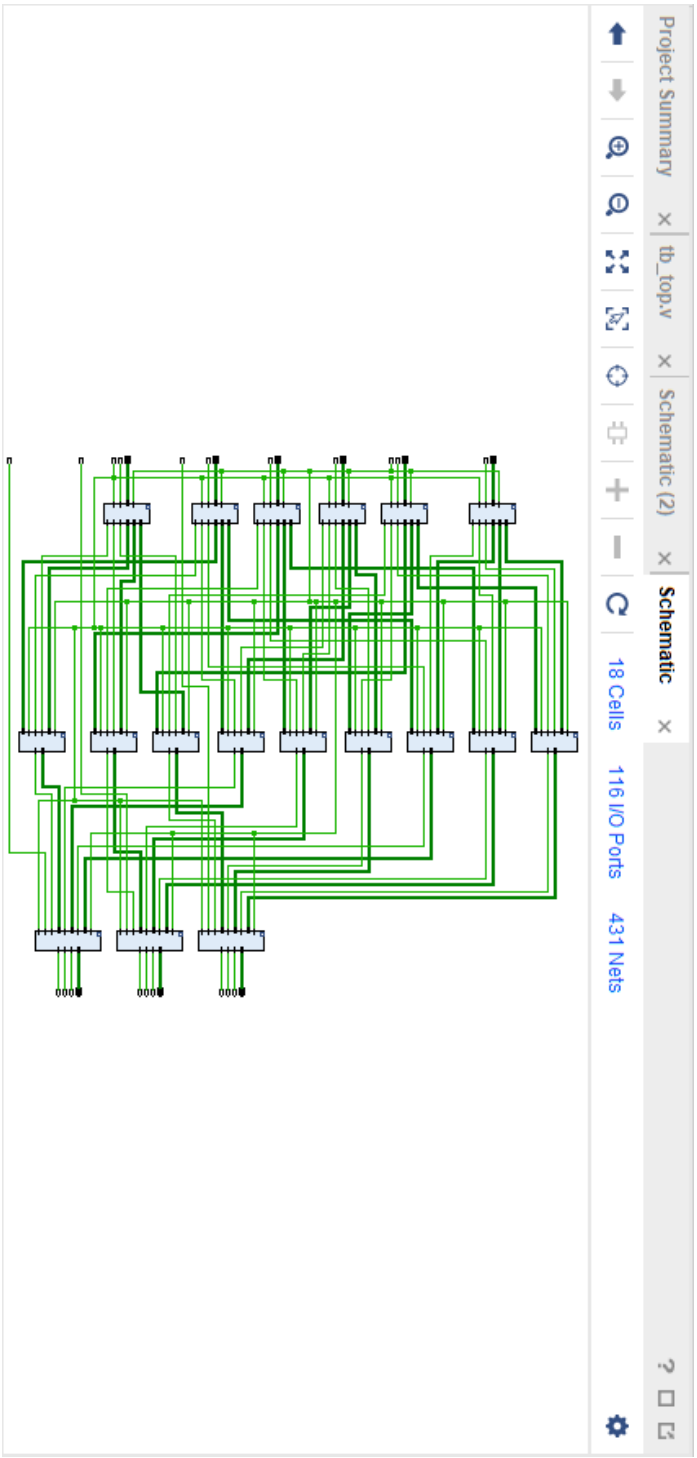


Figure 5.7: Schéma de câblage des différents blocs.

La Figure 5.8 présente de manière graphique le cheminement des caractéristiques d'entrée, ainsi que les poids associés à leurs bus respectifs, vers les PEs correspondants. Cette représentation visuelle permet de visualiser clairement le flux des données et l'interaction entre les caractéristiques d'entrée et les PEs.

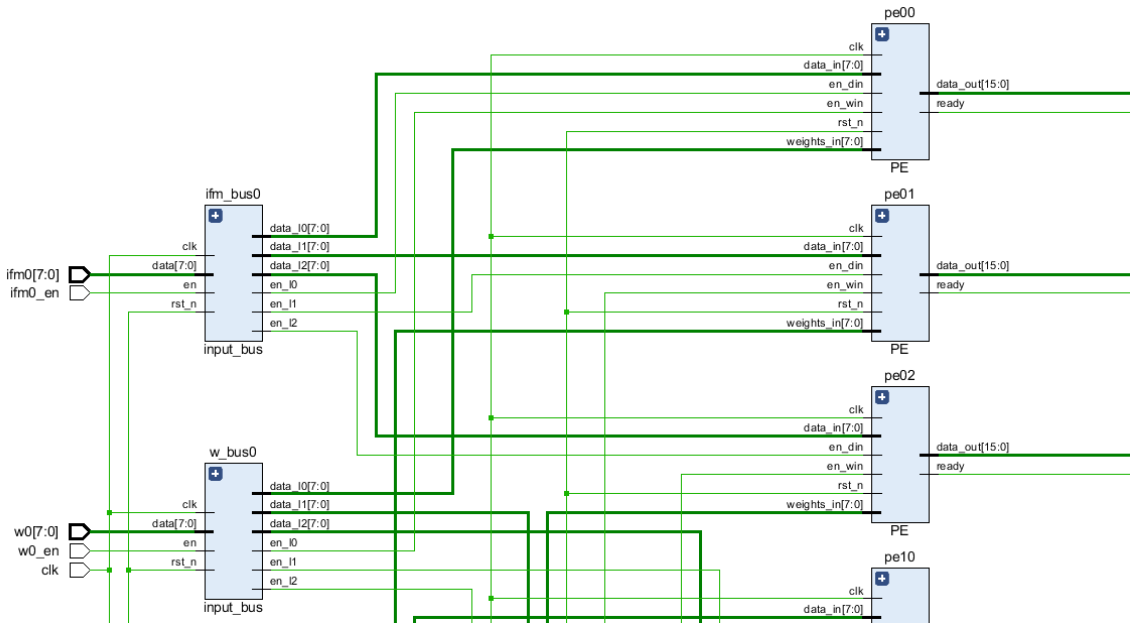


Figure 5.8: Flux des caractéristiques d'entrée et poids vers les PEs.

Le schéma présente dans la figure 5.9 expose de manière didactique le schéma de câblage ainsi que les ressources internes qui composent le bus d'entrée. Cette visualisation offre une analyse minutieuse de la structure et de la configuration des composants essentiels du bus. Fondamentalement, celui-ci est composé de trois buffers distincts, chacun dédié au stockage d'une caractéristique d'entrée spécifique avant de la diriger vers le PE correspondant. Il convient également de noter la connexion en cascade des buffers, ce qui permet un acheminement systolique des données.

Dans l'architecture en matrice systolique, un élément capital réside dans la notion du PE. Ce dernier représente l'unité de base, fondamentale pour le fonctionnement et la performance globale du système. L'architecture de chaque PE ainsi que ses ressources internes revêtent une importance capitale, car elles déterminent en grande partie les capacités de traitement et les performances de l'ensemble du système.

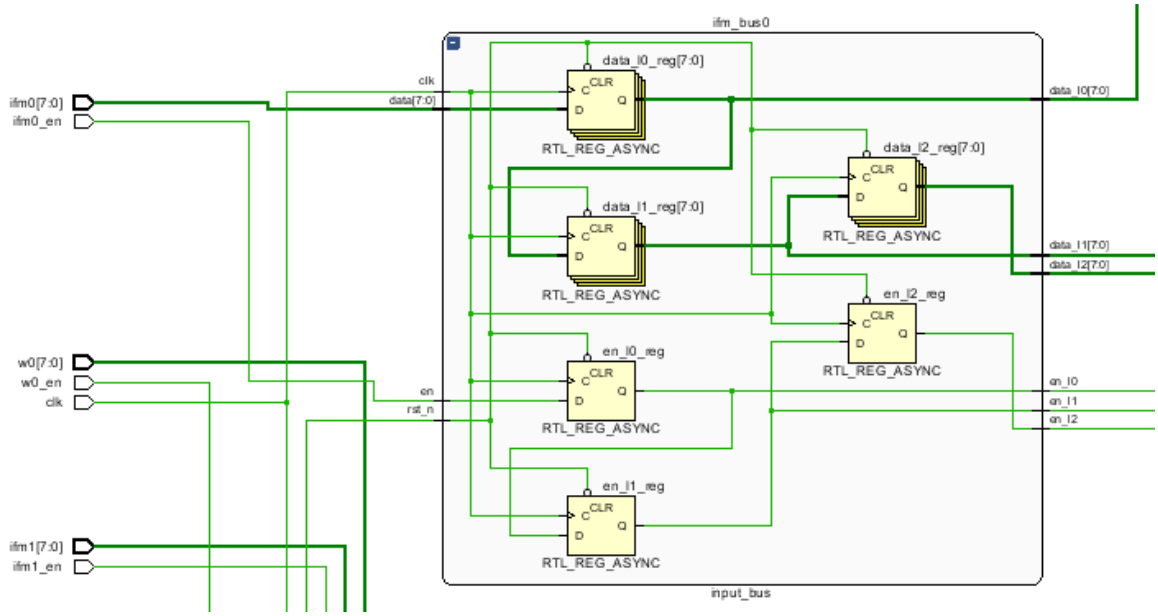


Figure 5.9: Câblage et composants internes du bus de transmission des caractéristiques d'entrée vers les PEs.

Le câblage et les caractéristiques internes des PEs sont précisément définis par le schéma illustré dans la figure 5.10. Ce schéma fournit un cadre structuré et détaillé décrivant la connectivité, les interconnexions, ainsi que la configuration des composants internes essentiels au fonctionnement du PE.

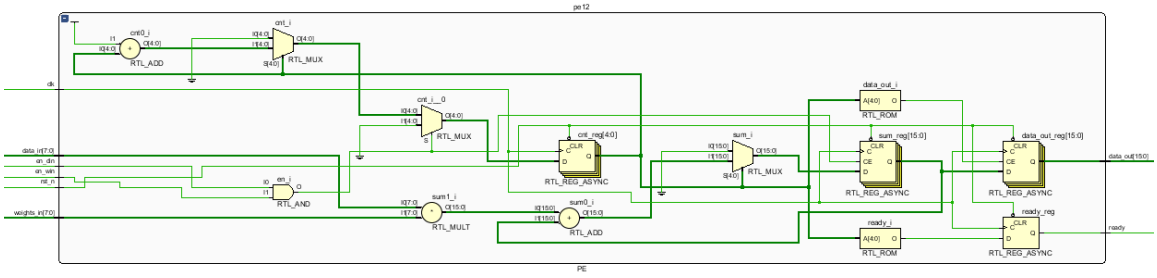


Figure 5.10: Connectivité et Configuration des Composants Constituant les PEs .

L'analyse de la consommation énergétique du module de convolution est présentée dans la Figure 5.11, où l'on observe une consommation totale de 0.243 W, répartie à hauteur de 61% entre les entrées et les sorties.

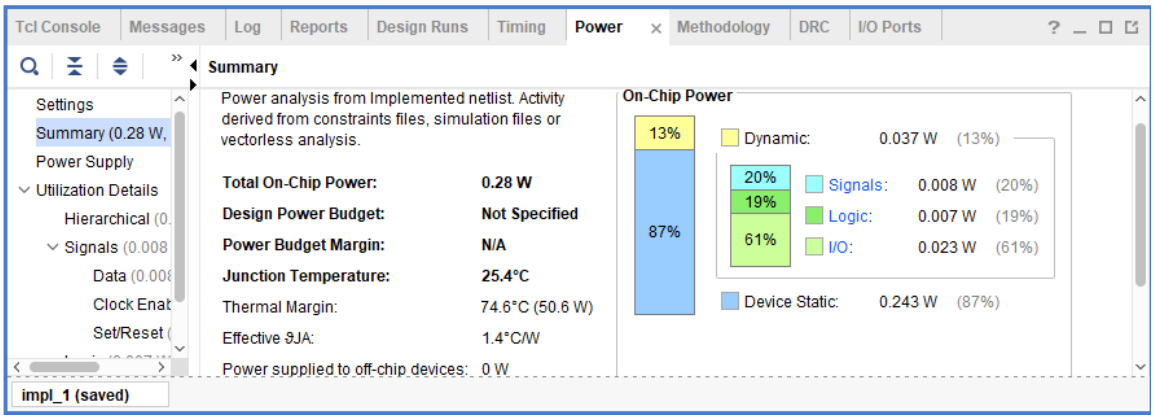


Figure 5.11: Rapport détaillé de la puissance consommée par le module de convolution.

La représentation graphique de l'occupation matérielle du circuit sur la plateforme FPGA (Les tâches colorées en vert) est présentée dans la Figure 5.12.

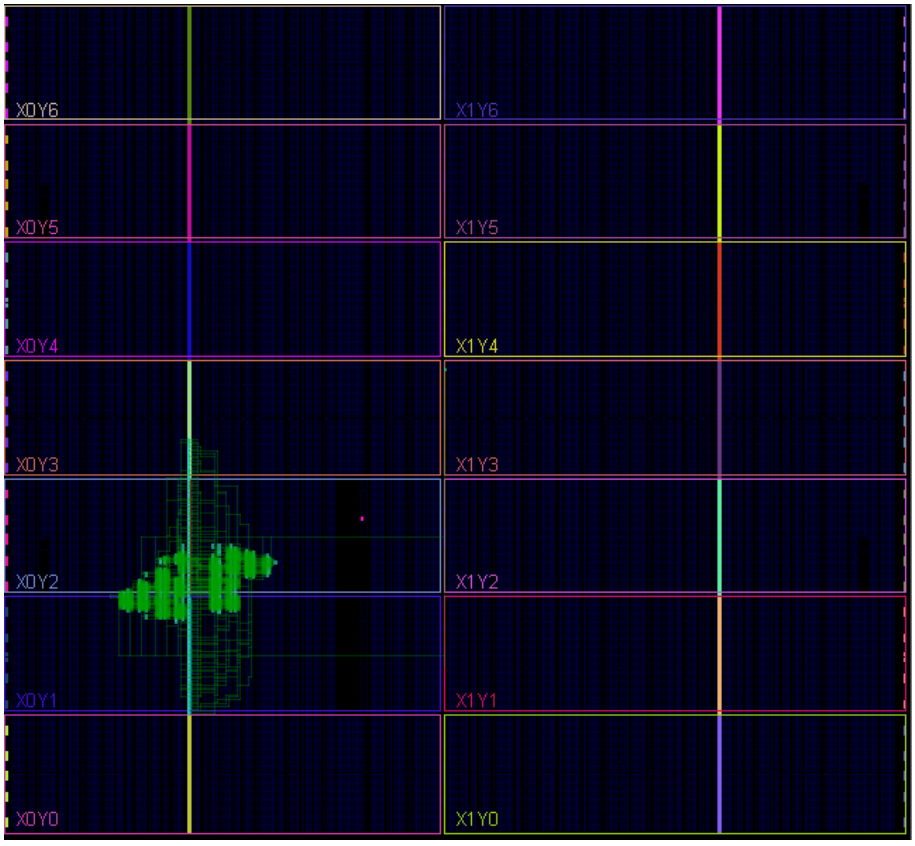


Figure 5.12: Occupation matérielle du circuit sur la plateforme FPGA.

5.3 Conclusion

Dans ce chapitre, on a examiné en détail l'implémentation matérielle de la couche de convolution à l'aide de matrices systoliques sur une plateforme FPGA, en mettant l'accent sur le RTL, notamment le Verilog. L'analyse approfondie des principes fondamentaux de cette approche a révélé les défis techniques associés à sa mise en œuvre. De plus, une exploration détaillée des différentes phases du processus de convolution basé sur les matrices systoliques a été menée, couvrant le cheminement systolique des données d'entrée, le parallélisme des opérations et le cheminement des résultats vers les couches suivantes du modèle CNN. Les outils de simulation, d'analyse de synthèse et d'implémentation ont joué un rôle crucial en fournissant des rapports sur les ressources utilisées et les performances, facilitant ainsi la prise de décision pour les choix architecturaux et les optimisations.

Conclusions et travaux futures

Conclusions

Le thème central de notre thèse porte sur l'implémentation efficace des réseaux de neurones convolutifs (CNN) sur des circuits FPGA (Field-Programmable Gate Array). Notre objectif principal est de contribuer au développement de méthodologies avancées d'implémentation et d'optimisation de ces réseaux sur des plates-formes matérielles dans le but d'améliorer considérablement leurs performances.

Au cours de notre recherche, nous avons exploré deux approches essentielles pour la programmation et l'implémentation. Tout d'abord, nous avons initié notre démarche en nous concentrant sur les méthodes d'implémentation exploitant le "High-Level Synthesis" (HLS), disponible sur la plateforme Xilinx Vivado. Cet outil de programmation offre la possibilité de traduire du code de haut niveau, tel que du code C ou C++, en code matériel, ce qui permet une exécution plus rapide et plus efficace sur des FPGA. Le HLS simplifie considérablement le processus de conception matérielle en permettant de décrire les opérations en termes de comportement et d'algorithmes, pour ensuite les convertir en logique matérielle.

En outre, cet environnement présente un ensemble de directives d'optimisation architecturale, permettant ainsi d'améliorer la performance et l'efficacité des conceptions matérielles. De plus, il offre des métriques d'analyse, de synthèse et de comparaison de solutions, facilitant ainsi l'évaluation et l'optimisation continues des implémentations matérielles. Cette démarche, en combinant la flexibilité du langage de haut niveau avec la puissance de l'exécution matérielle, ouvre la voie à des solutions innovantes et efficaces dans le domaine de la conception FPGA

Au cours de cette démarche, une exploration exhaustive du modèle CNN LeNet5 a été entreprise. Dans une première étape, le modèle a été soumis à un processus d'apprentissage en utilisant le jeu de données MNIST, et à la fin de cette phase, une

remarquable précision de 97% a été atteinte. Les paramètres ainsi que les biais des filtres issus de ce phase d'apprentissage ont été intégrés dans la phase d'inférence, laquelle a été réalisée au sein de l'environnement HLS de Vivado. L'utilisation des outils métriques d'analyse, de synthèse et de comparaison de Vivado a permis d'évaluer avec précision et de comparer les performances des différentes conceptions Explorées. Ceci a permis de quantifier les avantages et les limitations de chaque conceptions. Au bout de cette approche une latence de 299.3 μs a été réalisée.

Nous avons exploré le langage de description matérielle Verilog, soulignant son rôle crucial dans la conception détaillée du matériel FPGA et offrant un contrôle précis sur son architecture. Cette approche permet la création de conceptions personnalisées en spécifiant chaque élément matériel, ce qui est avantageux pour les projets nécessitant un contrôle fin sur le FPGA. À travers l'utilisation de Verilog, nous avons proposé une approche de parallélisme pour le module de convolution, principalement basée sur les matrices systoliques.

En explorant ces deux approches, notre objectif était de mettre en œuvre les différentes optimisations ainsi que les divers outils de développement permettant une implémentation efficace des Convolutional Neural Networks (CNNs) sur les plateformes FPGA. Cela impliquait de considérer une gamme d'optimisations telles que le pipelining des fonctions, le parallelisme des boucles et le partitionnement des matrices, ainsi que l'utilisation d'outils spécifiques de développement tels que le HLS et le Verilog.

Travaux futures

Dans le cadre de cette thèse, nos travaux de recherche ont ouvert des perspectives intéressantes pour des améliorations et des extensions futures. Ainsi, plusieurs pistes de recherche peuvent être envisagées pour des travaux futurs. Tout d'abord, des efforts supplémentaires peuvent être consacrés à l'exploration de nouvelles approches méthodologiques visant à optimiser les performances des CNNs. Cela pourrait inclure des études approfondies sur les techniques d'entraînement et de régularisation, ainsi que sur les méthodes d'adaptation aux données et aux tâches spécifiques.

Par ailleurs, une attention particulière peut être portée aux aspects applicatifs

des CNNs, ainsi que des intelligences artificielles en général. Il serait intéressant d'explorer davantage les domaines où les CNNs peuvent apporter une valeur ajoutée significative, tels que la vision par ordinateur, le traitement du langage naturel, la médecine, l'ingénierie, et bien d'autres. Des études empiriques approfondies pour évaluer l'efficacité des CNNs dans des applications réelles pourraient également être entreprises, afin de mieux comprendre leurs avantages, leurs limites et leurs potentiels d'amélioration.

Bibliographie

[1] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5 :115–133, 1943.

[2] Walter Pitts and Warren S McCulloch. How we know universals the perception of auditory and visual forms. *The Bulletin of mathematical biophysics*, 9 :127–147, 1947.

[3] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 52 :99–115, 1990.

[4] Jerome Y Lettvin, Humberto R Maturana, Warren S McCulloch, and Walter H Pitts. What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11) :1940–1951, 1959.

[5] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss : An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1) :127–138, 2016.

[6] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

[7] Long Cheng, Yan Gu, Qingzhi Liu, Lei Yang, Cheng Liu, and Ying Wang. Advancements in accelerating deep neural network inference on aiot devices : A survey. *IEEE Transactions on Sustainable Computing*, 2024.

[8] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, and Jason Cong. Caffeine : Towards uniformed representation and acceleration for deep convo-

lutional neural networks. In *Proceedings of the ACM Turing Award Celebration Conference-China 2023*, pages 47–48, 2023.

[9] McCulloch Warren S. and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophysics*, 5 :115–133, 1943.

[10] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[12] Guoming Li, Yanbo Huang, Zhiqian Chen, Gary D Chesser, Joseph L Purswell, John Linhoss, and Yang Zhao. Practices and applications of convolutional neural network-based computer vision systems in animal farming : A review. *Sensors*, 21(4) :1492, 2021.

[13] Insaf Adjabi, Abdeldjalil Ouahabi, Amir Benzaoui, and Abdelmalik Taleb-Ahmed. Past, present, and future of face recognition : A review. *Electronics*, 9(8) :1188, 2020.

[14] Hoel Kervadec, Jose Dolz, Meng Tang, Eric Granger, Yuri Boykov, and Ismail Ben Ayed. Constrained-cnn losses for weakly supervised segmentation. *Medical image analysis*, 54 :88–99, 2019.

[15] Teja Kattenborn, Jens Leitloff, Felix Schiefer, and Stefan Hinz. Review on convolutional neural networks (cnn) in vegetation remote sensing. *ISPRS journal of photogrammetry and remote sensing*, 173 :24–49, 2021.

[16] Qasem Abu Al-Haija and Saleh Zein-Sabatto. An efficient deep-learning-based detection and classification system for cyber-attacks in iot communication networks. *Electronics*, 9(12) :2152, 2020.

[17] Shaohua Wan and Sotirios Goudos. Faster r-cnn for multi-class fruit detection using a robotic vision system. *Computer Networks*, 168 :107036, 2020.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets : Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv :1704.04861*, 2017.

[23] Mingxing Tan and Quoc Le. Efficientnet : Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[24] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[25] François Chollet. Xception : Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[26] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet : Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv :1602.07360*, 2016.

[27] Pablo Barros, Sven Magg, Cornelius Weber, and Stefan Wermter. A multichannel convolutional neural network for hand posture recognition. In *Artificial Neural Networks and Machine Learning–ICANN 2014 : 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014. Proceedings 24*, pages 403–410. Springer, 2014.

[28] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[29] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10) :1533–1545, 2014.

[30] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[31] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv :1510.03820*, 2015.

[32] Ruhi Sarikaya, Geoffrey E Hinton, and Anoop Deoras. Application of deep belief networks for natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(4) :778–784, 2014.

[33] Shengxin Zha, Florian Luisier, Walter Andrews, Nitish Srivastava, and Ruslan Salakhutdinov. Exploiting image-trained cnn architectures for unconstrained video classification. *arXiv preprint arXiv :1503.04144*, 2015.

[34] Kelathodi Kumaran Santhosh, Debi Prosad Dogra, Partha Pratim Roy, and Adway Mitra. Vehicular trajectory classification and traffic anomaly detection in videos using a hybrid cnn-vae architecture. *IEEE Transactions on Intelligent Transportation Systems*, 23(8) :11891–11902, 2021.

[35] Mahbub Hussain, Jordan J Bird, and Diego R Faria. A study on cnn transfer learning for image classification. In *Advances in Computational Intelligence Systems : Contributions Presented at the 18th UK Workshop on Computational Intelligence, September 5-7, 2018, Nottingham, UK*, pages 191–202. Springer, 2019.

[36] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th international conference on control automation robotics & vision (ICARCV)*, pages 844–848. IEEE, 2014.

[37] Sai Yeshwanth Chaganti, Ipseeta Nanda, Koteswara Rao Pandi, Tavva GNRSN Prudhvith, and Niraj Kumar. Image classification using svm and cnn. In *2020 International conference on computer science, engineering and applications (ICCSEA)*, pages 1–5. IEEE, 2020.

[38] Shubham Kumar Singh, Revant Kumar Thakur, Satish Kumar, and Rohit Anand. Deep learning and machine learning based facial emotion detection using cnn. In *2022 9th international conference on computing for sustainable global development (INDIACom)*, pages 530–535. IEEE, 2022.

[39] RKMS K Karunanayake, WG Malaka Dananjaya, MS Y Peiris, BRIS Gunatileka, Shashika Lokuliyana, and Anuththara Kuruppu. Cureto : skin diseases detection using image processing and cnn. In *2020 14th international conference on Innovations in Information Technology (IIT)*, pages 1–6. IEEE, 2020.

[40] Jiaxing Li, Dexiang Zhang, Jingjing Zhang, Jun Zhang, Teng Li, Yi Xia, Qing Yan, and Lina Xun. Facial expression recognition with faster r-cnn. *Procedia Computer Science*, 107 :135–140, 2017.

[41] Yan Zhao, Shuxiang Guo, Yuxin Wang, Jinxin Cui, Youchun Ma, Yuwen Zeng, Xinke Liu, Yuhua Jiang, Youxinag Li, Liwei Shi, et al. A cnn-based prototype method of unstructured surgical state perception and navigation for an endovascular surgery robot. *Medical & Biological Engineering & Computing*, 57 :1875–1887, 2019.

[42] Tsun-Hsuan Wang, Hung-Jui Huang, Juan-Ting Lin, Chan-Wei Hu, Kuo-Hao Zeng, and Min Sun. Omnidirectional cnn for visual place recognition and navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2341–2348. IEEE, 2018.

[43] Alexandros Kouris and Christos-Savvas Bouganis. Learning to fly by myself : A self-supervised cnn-based approach for autonomous navigation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[44] François Bardet, Thierry Chateau, and Datta Ramadasan. Suivi et catégorisation visuels en temps réel d'un nombre variable d'objets : application au suivi de véhicules. In *ORASIS'09-Congrès des jeunes chercheurs en vision par ordinateur*, 2009.

[45] Dr M Durairaj and BH Krishna Mohan. A convolutional neural network based approach to financial time series prediction. *Neural Computing and Applications*, 34(16) :13319–13337, 2022.

[46] Pedram Khatamino, İsmail Cantürk, and Lale Özyılmaz. A deep learning-cnn based system for medical diagnosis : An application on parkinson's disease handwriting drawings. In *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pages 1–6. IEEE, 2018.

[47] Paolo Arena, Adriano Basile, Maide Bucolo, and Luigi Fortuna. Image processing for medical diagnosis using cnn. *Nuclear Instruments and Methods in Physics Research Section A : Accelerators, Spectrometers, Detectors and Associated Equipment*, 497(1) :174–178, 2003.

[48] Ahmad Waleed Salehi, Shakir Khan, Gaurav Gupta, Bayan Ibrahimm Alabduallah, Abrar Almjally, Hadeel Alsolai, Tamanna Siddiqui, and Adel Mellit. A study of cnn and transfer learning in medical imaging : Advantages, challenges, future scope. *Sustainability*, 15(7) :5930, 2023.

[49] Jingyu Yang, Jianhua Guo, Huanjing Yue, Zhiheng Liu, Haofeng Hu, and Kun Li. Cdnet : Cnn-based cloud detection for remote sensing imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 57(8) :6195–6211, 2019.

[50] Wei Zhang, Ping Tang, and Lijun Zhao. Remote sensing image scene classification using cnn-capsnet. *Remote Sensing*, 11(5) :494, 2019.

[51] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 477–484. IEEE, 2016.

[52] Yixing Li, Zichuan Liu, Kai Xu, Hao Yu, and Fengbo Ren. A gpu-outperforming fpga accelerator architecture for binary convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 14(2) :1–16, 2018.

[53] Raju Machupalli, Masum Hossain, and Mrinal Mandal. Review of asic accelerators for deep neural network. *Microprocessors and Microsystems*, 89 :104441, 2022.

[54] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-Sun Seo. Performance modeling for cnn inference accelerators on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(4) :843–856, 2019.

[55] Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, and Jun Yang. Dracc : A dram based accelerator for accurate cnn inference. In *Proceedings of the 55th annual design automation conference*, pages 1–6, 2018.

[56] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.

[57] Sparsh Mittal. A survey of fpga-based accelerators for convolutional neural networks. *Neural computing and applications*, 32(4) :1109–1139, 2020.

[58] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 26–35, 2016.

[59] Hardik Sharma, Jongse Park, Emmanuel Amaro, Bradley Thwaites, Praneetha Kotha, Anmol Gupta, Joon Kyung Kim, Asit Mishra, and Hadi Esmaeilzadeh. Dnnweaver : From high-level deep network models to fpga acceleration. In *the Workshop on Cognitive Architectures*, 2016.

[60] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. Accelerating deep convolutional networks using low-precision and sparsity. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2861–2865. IEEE, 2017.

[61] Yu Xing, Shuang Liang, Lingzhi Sui, Xijie Jia, Jiantao Qiu, Xin Liu, Yushun Wang, Yi Shan, and Yu Wang. Dnnvm : End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10) :2668–2681, 2019.

[62] Dong Wang, Jianjing An, and Ke Xu. Pipecnn : An opencl-based fpga accelerator for large-scale convolution neuron networks. *arXiv preprint arXiv :1611.02450*, 2016.

[63] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[64] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization : Towards lossless cnns with low-precision weights. *arXiv preprint arXiv :1702.03044*, 2017.

[65] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv :1612.01064*, 2016.

[66] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net : Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[67] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect : Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[68] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[69] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858. PMLR, 2016.

[70] Diederik P Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

[71] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv :1904.09237*, 2019.

[72] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks : Tricks of the Trade : Second Edition*, pages 421–436. Springer, 2012.

[73] Tijmen Tieleman and Geoffrey Hinton. Rmsprop : Divide the gradient by a running average of its recent magnitude. coursera : Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn*, 17, 2012.

[74] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[75] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

[76] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv :1412.6553*, 2014.

[77] Xingyu Liu, Jeff Pool, Song Han, and William J Dally. Efficient sparse-winograd convolutional neural networks. *arXiv preprint arXiv :1802.06367*, 2018.

[78] Chihiro Ito, Xin Cao, Masaki Shuzo, and Eisaku Maeda. Application of cnn for human activity recognition with fft spectrogram of acceleration and gyro sensors. In *Proceedings of the 2018 ACM international joint conference and 2018 international symposium on pervasive and ubiquitous computing and wearable computers*, pages 1503–1510, 2018.

[79] Kamran Chitsaz, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahram Shirani. Acceleration of convolutional neural network using fft-based split convolutions. *arXiv preprint arXiv :2003.12621*, 2020.

[80] Aravind Ravi, Nargess Heydari, and Ning Jiang. User-independent ssvep bci using complex fft features and cnn classification. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 4175–4180. IEEE, 2019.

[81] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2 : A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2) :292–308, 2019.

[82] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2018.

[83] Jung Hwan Kim and Sudhakar M. Reddy. On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement. *IEEE Transactions on Computers*, 38(4) :515–525, 1989.

[84] Chun Yan Lo and Chiu-Wing Sham. Energy efficient fixed-point inference system of convolutional neural network. In *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 403–406. IEEE, 2020.

[85] Sina Ghaffari and Saeed Sharifian. Fpga-based convolutional neural network accelerator design using high level synthesize. In *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*, pages 1–6. IEEE, 2016.

[86] Zhiqiang Liu, Yong Dou, Jingfei Jiang, and Jinwei Xu. Automatic code generation of convolutional neural networks in fpga implementation. In *2016 International conference on field-programmable technology (FPT)*, pages 61–68. IEEE, 2016.

[87] Xinheng Liu, Dae Hee Kim, Chang Wu, and Deming Chen. Resource and data optimization for hardware implementation of deep neural networks targeting fpga-based edge devices. In *Proceedings of the 20th System Level Interconnect Prediction Workshop*, pages 1–8, 2018.

[88] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[89] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert Ou, Max Banister, et al. Gemmini : An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv preprint arXiv :1911.09925*, 3 :25, 2019.

[90] Surya Selvam, Vinod Ganesan, and Pratyush Kumar. Fuseconv : Fully separable convolutions for fast inference on systolic arrays. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 651–656. IEEE, 2021.

[91] AMD. Amd virtex 7 fpga vc707 evaluation kit.

Annexe A

Données Annexes

A.1 Format HDF5

HDF5 est un modèle de données, une bibliothèque et un format de fichier pour stocker et gérer des données. Il prend en charge une variété illimitée de types de données et est conçu pour des opérations d'entrée/sortie flexibles et efficaces, ainsi que pour la gestion de données volumineuses et complexes. HDF5 est portable et extensible, permettant aux applications d'évoluer dans leur utilisation de HDF5. La suite de technologies HDF5 comprend des outils et des applications pour la gestion, la manipulation, la visualisation et l'analyse des données au format HDF5.

A.2 Ensemble de données MNIST

La base de données MNIST, un sous-ensemble des bases de données spéciales du Modified National Institute of Standards and Technology (NIST), est une collection de chiffres manuscrits comprenant 60 000 exemples d'entraînement et 10 000 exemples de test. Ces chiffres ont été normalisés en termes de taille et centrés dans des images de 20x20 pixels, tout en préservant leur aspect d'origine en noir et blanc. Cette normalisation a entraîné des niveaux de gris dans les images en raison de l'anti-aliasing utilisé. Enfin, les images ont été centrées dans une image de 28x28 pixels en calculant le centre de masse des pixels, positionnant ainsi ce point au centre de chaque image.La Figure A.1 montre un échantillon de chiffres

de la base de données MNIST affiché lors de la phase d'apprentissage pour vérifier son téléchargement.

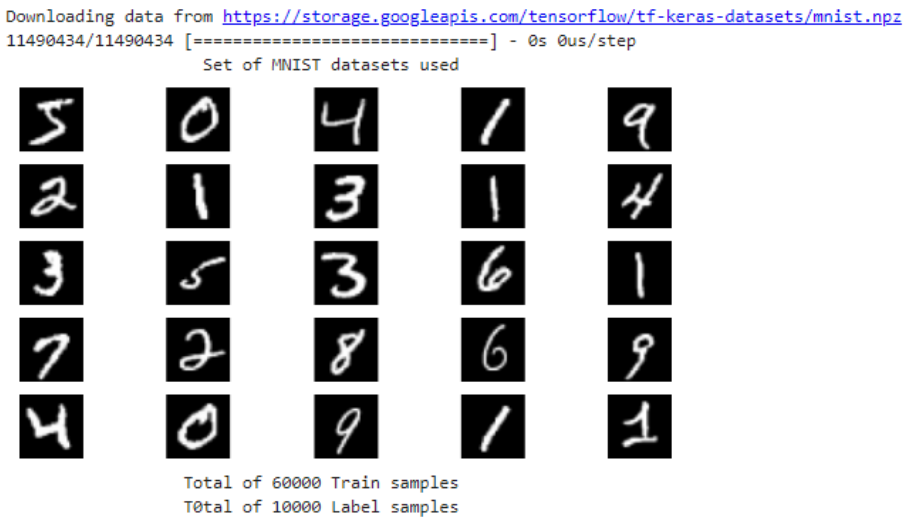


Figure A.1: Exemples de chiffres issus de l'ensemble de données MNIST importé.

Annexe B

Schémas bloc et caractéristiques.

B.1 Kit FPGA Vertex VC707

Le Kit d'évaluation Virtex^{TM} 7 FPGA VC707 est une plateforme complète, hautement flexible et à haut débit, utilisant la puce Virtex 7 XC7VX485T-2FFG1761C. Elle comprend des composants matériels de base, des outils de conception, des propriétés intellectuelles (IP) et des conceptions de référence pré-vérifiées pour des systèmes exigeant une connectivité série à haute performance et une interface mémoire avancée. Les conceptions de référence pré-vérifiées incluses et les connecteurs FPGA Mezzanine Connectors (FMC) conformes aux normes de l'industrie permettent une mise à l'échelle et une personnalisation avec des cartes filles [91].

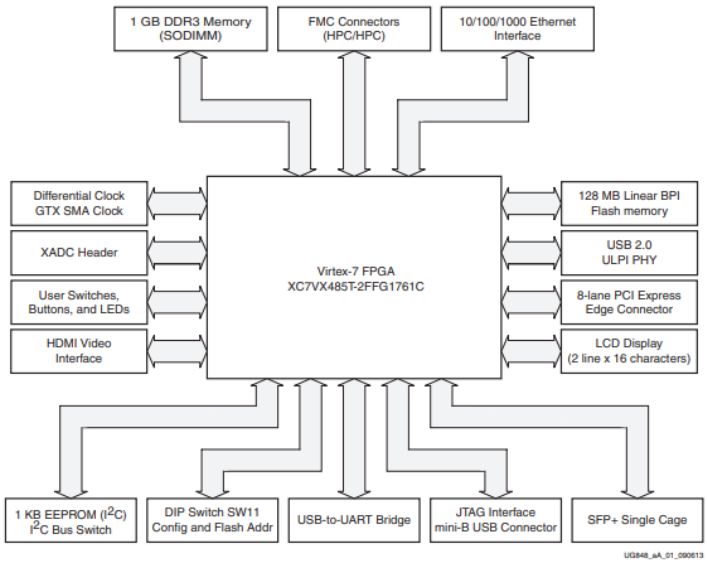


Figure B.1: Schéma bloc de la carte FPGA VC707 [91].

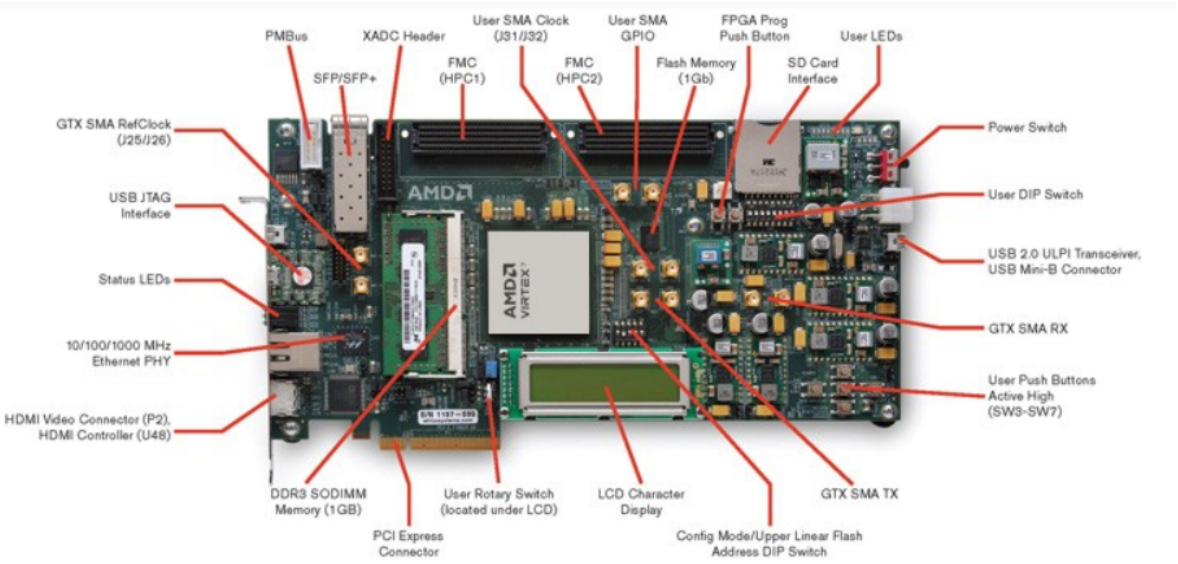


Figure B.2: Emplacements des composants de la carte VC707 [91].

B.1.1 Caractéristiques clés et avantages

— Plateforme de connectivité 40 Gb/s pour des applications à large bande passante et à haute performance utilisant les FPGA Virtex 7 VX485T
— Matériel, outils de conception, propriétés intellectuelles (IP) et conceptions de référence pré-vérifiées
— Prise en charge du traitement intégré avec MicroBlaze, RISC 32 bits logiciel
— Activation de la connectivité série avec PCIe Gen2x8, SFP+ et paires SMA, UART, IIC
— Interface mémoire avancée avec 1 Go de mémoire DDR3 SODIM jusqu'à 800 MHz / 1600 Mbps
— Développement d'applications réseau avec Ethernet 10-100-1000 Mbps (GMII, RGMII et SGMII)
— Expansion des entrées/sorties avec l'interface de carte fille FPGA Mezzanine Card (FMC)

B.1.2 Ressources matérielles

— Logic Cells	485,760
— DSP Slices	2,800
— Memory (Kb)	37,080

— GTX 12.5 GB/s Transceivers 56
— I/O Pins 700

B.1.3 Mémoire

— 1GB DDR3 SODIMM 800MHz / 1600Mbps
— 128MB (1024Mb) Linear BPI Flash for PCIe Configuration
— SD Card Slot
— 8Kb IIC EEPROM

B.2 High-Level Synthesis

L'outil de synthèse de haut niveau (HLS) Xilinx®ivado®ransforme une spécification en langage C en une implémentation de niveau de transfert de registre (RTL) qui peut être synthétisée dans une matrice logique programmable (FPGA) de Xilinx.On peut rédiger des spécifications en langage C, C++ ou SystemC, et le FPGA offre une architecture massivement parallèle présentant des avantages en termes de performances, de coûts et de consommation énergétique par rapport aux processeurs traditionnels.