وزارة التعليم العالي و البحث العلمي

BADJIMOKHTAR-ANNABAUNIVERSITY

UNIVERSITE BADJI MOKHTAR ANNABA

جامعة باجي مختار-عنابة

Année : 2020

Faculté: Sciences de L'Ingéniorat

Département : Electronique

MEMOIRE

Présenté en vue de l'obtention du diplôme de : Master

Étude et implémentation d'un réseau de neurones convolutif CNN sur FPGA.

Domaine : Sciences et Technologies

Filière : Electronique

Spécialité : Electronique des systèmes embarqués

Par: Braikia Oussama et Bouchouicha Nadir

DEVANT Le JURY

Ali Bouchaala	MCB	U.ANNABA	Président
Amira YAHI	MCB	U.ANNABA	Encadrant
Hamza Attoui	MCB	U.ANNABA	Examinateur

Résumé :

Le terme Deep Learning ou Deep Neural Network fait référence aux réseaux de neurones artificiels (ANN) à plusieurs couches. Au cours des dernières décennies, il a été considéré comme l'un des outils les plus puissants et est devenu très populaire dans la littérature car il est capable de gérer une énorme quantité de données. L'intérêt d'avoir des couches cachées plus profondes a récemment commencé à surpasser les performances des méthodes classiques dans différents domaines; en particulier dans la reconnaissance de formes. L'un des réseaux neuronaux profonds les plus populaires est le réseau neuronal convolutif (CNN). Il tire ce nom d'une opération linéaire mathématique entre des matrices appelée convolution. Le CNN a une excellente performance dans les problèmes d'apprentissage automatique. En particulier, les applications qui traitent des données d'image, telles que le plus grand ensemble de données de classification d'images (Image Net), la vision par ordinateur et le traitement du langage naturel (NLP). Les exigences en matière de bande passante et de puissance de traitement ont mis les chercheurs au défi de trouver des structures permettant aux réseaux neuronaux convolutionnels modernes d'être intégrés dans les ASIC et les FPGA, cependant, une énorme quantité d'architectures matérielles dédiées a été proposée pour fournir des performances élevées à faible coût d'énergie et de surface. Dans ce mémoire, nous expliquerons et définirons tous les éléments et questions importantes liés à un réseau CNN, et comment ces éléments fonctionnent. En outre, nous essayons de proposer en VHDL une implémentation matérielle à base d'FPGA pour quelques modules du réseau CNN. L'objectif de cette étude est d'évaluer la faisabilité d'une telle implémentation et prouver son efficacité par rapport à une exécution à base d'un GPU.

Abstract:

The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers. Over the last few decades, it has been considered to be one of the most powerful tools, and has become very popular in the literature as it is able to handle a huge amount of data. The interest in having deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition. One of the most popular deep neural networks is the Convolutional Neural Network (CNN). It take this name from mathematical linear operation between matrixes called convolution. CNN have multiple layers; including convolutional layer, non-linearity layer, pooling layer and fully-connected layer. The convolutional and fully-connected layers have parameters but pooling and non-linearity layers don't have parameters. The CNN has an excellent performance in machine learning problems. Especially the applications that deal with image data, such as largest image classification data set (Image Net), computer vision, and in natural language processing (NLP).requirements in bandwidth and processing power have challenged architects to find structures that allow modern Convolutional Neural Networks to be embedded into ASICs and FPGAs. Thus, a huge amount of dedicated hardware architectures have been proposed to provide high performance at low energy and area costs. In this master thesis, we will explain and define all the elements and important issues related to CNN, and how these elements work. In addition, we propose a VHDL based hardware implementation of FPGA for a few modules for the CNN network. The objective of this study is to evaluate the feasibility of such implementation and prove its effectiveness compared to performances based on a GPU execution.

REMERCIEMENT

Nous souhaitons adresser nos remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire. A savoir nos amis, nos parents respectifs.

Nous tenons à remercier chaleureusement Mme. Amira YAHI pour son encadrement pédagogique, sa disponibilité, sa générosité et ses conseils qui nous avons bien aidé et guidé dans le bon chemin. Sans oublier d'adresser nos vifs remerciements aux membres de Jury M. BOUCHAALA et M. ATTOUI qui nous ont honoré de juger notre travail

Sommaire

Liste des figures

Liste des tableaux

ALU: arithmetic logic unit

ASIC: application-specific integrated circuit

CLB:configurable Logic Block

DSP:Digital Signal Processor

FSM:Finite state machine

HSL: High Level Synthesis

IA:Intelligence artificial

ILSVRC: ImageNet Large Scale Visual Recognition Challenge

MAC : Multiply and Accumulate

RAM:Random Access Memory

RGB: Rouge, vert, bleu, abrégé en RVB ou en RGB

RTL:Register Transfer Level

SRAM: Static Random Access Memory

SVM:Support vector machine

VHDL:Very High Speed Integrated Circuit Hardware Description Language

ANN: Artificial Neural Networks

CNN:Convolutional neural network

DL: Deep Learning

FPGA: Field Programmable Gate Arrays

GPU:Graphics processing unit

ISE: Integrated Synthesis Environment

LUT :Look UpTable

SIPO: Serial In - Parallel Out

SISO: Serial In - Serial Out

uC :Microcontrôleur

Chapitre 1 : Introduction

1. Introduction générale :

Les réseaux neuronaux convolutifs pour la reconnaissance de l'écriture manuscrite Les premiers travaux sur les réseaux neuronaux convolutifs (CNN) modernes ont eu lieu dans les années 1990, inspirés par le néocognitron. Yann LeCun et al, dans leur article "Gradient-Based Learning Applied to Document Recognition" (aujourd'hui cité 17 588 fois) ont démontré qu'un modèle CNN qui regroupe des caractéristiques simples en caractéristiques progressivement plus complexes peut être utilisé avec succès pour la reconnaissance de caractères manuscrits. **[1]**

Plus précisément, LeCun et al. A formé un CNN en utilisant la base de données du MNIST de chiffres manuscrits (MNIST se prononce "EM-nisst"). Le MNIST est un ensemble de données désormais célèbre qui comprend des images de chiffres manuscrits appariés avec leur véritable étiquette de 0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9. Un modèle CNN est formé sur le MNIST en lui donnant une image d'exemple, en lui demandant de prédire quel chiffre est montré dans l'image, puis en mettant à jour les paramètres du modèle selon qu'il a prédit l'identité du chiffre correctement ou non. Les modèles CNN de pointe peuvent aujourd'hui atteindre une précision quasi parfaite sur la classification numérique MNIST. [W1]

Figure 1.1 : **Techniques de reconnaissance automatique de l'écriture manuscrite**

Une conséquence directe de ce travail est que votre courrier est désormais trié par des machines, utilisant des techniques de reconnaissance automatique de l'écriture pour lire l'adresse. Des réseaux neuronaux convolutifs pour tout voir Tout au long des années 1990 et au début des années 2000, les chercheurs ont poursuivi leurs travaux sur le modèle CNN.

Vers 2012, les CNN ont connu une énorme poussée de popularité (qui se poursuit encore aujourd'hui) après qu'une CNN appelée AlexNet ait obtenu des images d'étiquetage à la pointe de la technologie dans le cadre du défi ImageNet. Alex Krizhevsky et al. a publié le document "ImageNet Classification withDeepConvolutional Neural Networks" décrivant le modèle AlexNet gagnant ; ce document a depuis été cité 38 007 fois.

Tout comme le MNIST, ImageNet est un ensemble de données publiques et gratuites sur les images et les véritables étiquettes correspondantes. Au lieu de se concentrer sur les chiffres manuscrits de 0 à 9, ImageNet se concentre sur les "images naturelles", ou les images du monde, étiquetées avec une variété de descripteurs dont "amphibien", "meuble" et "personne". Les étiquettes ont été acquises grâce à un effort humain massif (c'est-à-dire un étiquetage manuel - en demandant à quelqu'un d'écrire "de quoi s'agit-il ?" pour chaque image). ImageNet comprend actuellement 14 197 122 images.

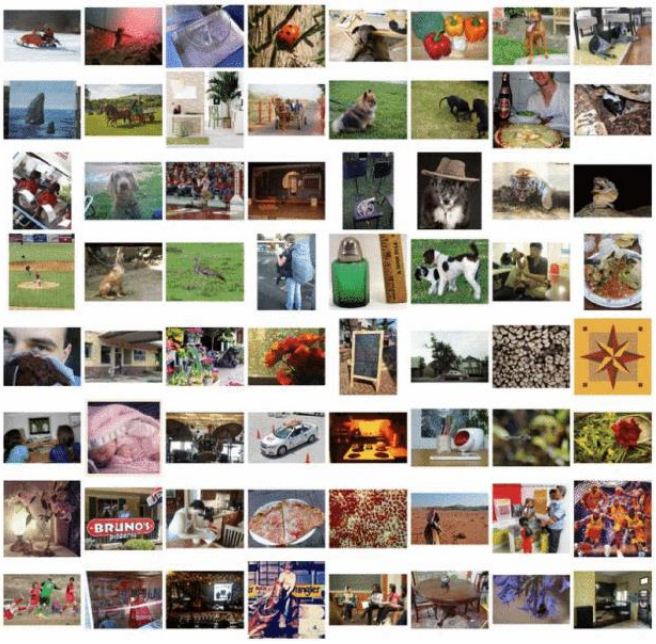


Figure 1.2 : **Image data base**

CNNs et vision humaine ? La presse populaire parle souvent de la façon dont les modèles de réseaux neuronaux sont "directement inspirés du cerveau humain". Dans un certain sens, c'est vrai, car les CNN et le système visuel humain suivent une structure hiérarchique "simple à complexe". Cependant, la mise en œuvre réelle est totalement différente ; les cerveaux sont construits à l'aide de cellules, et les réseaux neuronaux sont construits à l'aide d'opérations mathématiques. Conclusion La vision par ordinateur a fait beaucoup de chemin au cours des

dernières décennies. Il est passionnant d'imaginer les nouveaux développements qui transformeront le domaine à l'avenir et qui donneront un coup de fouet à des technologies telles que l'interprétation automatisée des images radiologiques et les voitures à conduite automatique.

Afin de diminuer l'utilisation accrue de la mémoire due aux massifs calculs nécessaires pour les récents CNN, des architectures matérielles ont été proposées par l'industrie et les communautés de recherche. Par exemple, Microsoft a créé Project Catapult, développant des accélérateurs FPGA pour ses serveurs cloud dans [Ovtcharov et al., 2015, Putnam et al., 2016, Caulfield et al., 2016]. Google a investi dans le développement d'une série d'ASIC pour le traitement du deeplearning connue sous le nom de TensorProcessing Unit [Jouppi et al., 2017]. Movidius, une société Intel, a intégré un outil de prototypage DNN (Deep Neural Network) dans une clé USB pour un développement rapide [Intel Corp., 2015]. De plus, de nombreuses startups sont apparues. D'un autre côté, les réseaux de neurones nécessitent également une quantité considérable de traitement. Par exemple, une convolution fait glisser une fenêtre sur les entrées, calculant plusieurs fois une somme de produits. En outre, le deuxième type de couche le plus courant dans les CNN, la couche entièrement connectée (fullyconnected), peut être considérée comme une séquence de multiplications matricielles, ce qui peut également être difficile en termes de calcul pour toute implémentation physique. A titre d'exemple, Alexnet a cinq couches convolutives et trois couches entièrement connectées tandis que VGGNet se compose de 16 couches convolutives et de trois couches entièrement connectées. Cette quantité d'opérations pose un problème même pour les architectures qui sont composées de nombreuses unités de traitement graphique (GPU) disposées sur un ordinateur. Plus précisément, lorsqu'il est proposé d'intégrer ces réseaux sur des FPGA ou des ASIC dédiés, ce volume d'opérations nécessite des schémas bien pensés de partage des ressources arithmétiques et, par conséquent, de gain de surface.

Il existe trois chemins de conception principaux que l'on peut suivre lors du développement d'architectures matérielles pour les algorithmes d'apprentissage en profondeur. La première consiste à adopter l'utilisation des GPU, où les développeurs implémentent un réseau en les programmant et en tirant parti du haut niveau de parallélisme fourni. En ce qui concerne la phase d'apprentissage, les GPU sont le choix numéro un des scientifiques en apprentissage automatique, car ils sont principalement axés sur la réalisation de performances optimales tant que la formation ne sera pas exécutée régulièrement. En outre, pendant cette phase, des

opérations en virgule flottante de haute précision (également fournies par les GPU) sont requises par des algorithmes tels que la rétro propagation. Les deux autres chemins possibles incluent les FPGA ou les ASIC. Le premier est l'un des choix les plus flexibles et le second atteint les performances les plus élevées [Kuon et Rose, 2007]. Le flux de conception pour un ASIC est coûteux et implique de nombreuses étapes de la spécification initiale à la sortie de bande finale et à la fabrication, contrairement aux FPGA qui peuvent fournir une mise sur le marché rapide et d'éventuelles mises à niveau de l'architecture après la livraison. Fondamentalement, FPGA et ASIC ont attiré de nombreux concepteurs pour mettre en œuvre des algorithmes d'inférence, en raison de la possibilité d'utiliser des formats numériques de toute précision [Jiao et al., 2017, Courbariaux et al., 2015]; la possibilité d'économies d'énergie par rapport aux GPU [Nurvitadhi et al., 2017, Gupta et al., 2015]; et les performances globales ainsi que le potentiel de mise au point de l'architecture [Hailesellasie et al., 2018].

2. Contribution du mémoire :

L'objectif de ce mémoire est de mettre en œuvre une architecture matérielle capable de s'exécuter sur une plateforme FPGA d'un réseau de neurone convolutif CNN, pour cela, une étude a été faite en décrivant le fonctionnement des modules concernés, nous les détaillons puis nous proposons une architecture matérielle avec des schéma RTL pour chacun de ces modules en utilisant le logiciel ISE (Xilinx). L'objectif principal est de montrer l'efficacité d'une telle réalisation par rapport à une exécution à base d'un GPU.

Ce document est organisé comme suit : le premier chapitre présente une introduction générale sur les réseaux de neurones convolutifs CNN, dans lequel nous présentons les questions liées au développement matériel des réseaux CNN, avec quelques exemples de l'industrie. Dans le deuxième chapitre, une étude est faite sur les concepts d'apprentissage automatique et les réseaux de neurones. Le chapitre 3 évoque les réseaux CNN de façon précise et détaillée, puis la plateforme de prototypage FPGA avec ses performances par rapport aux GPU. Finalement, dans le chapitre 4, nous proposons une architecture matérielle pour quelques modules d'un réseau CNN, que nous définissons en détails.

Chapitre 2 : Les réseaux de neurones

1. Introduction :

Le monde est rempli d'objets que nous pouvons presque tous comprendre et qui nous font réagir sans trop réfléchir. Par exemple, un panneau "Stop" en partie recouvert de neige reste un panneau "Stop", et une chaise cinq fois plus grande qu'une chaise classique reste un endroit où s'asseoir. Toutefois, pour les ordinateurs standards, ce type de logique intuitive est hors de portée. Aujourd'hui, l'apprentissage automatique est capable d'offrir cet avantage aux ordinateurs grâce à sa technologie avancée.

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle (IA). En général, l'objectif de l'apprentissage automatique est de comprendre la structure des données et de les intégrer dans des modèles qui peuvent être compris et utilisés par le monde. [W2]

Bien que l'apprentissage automatique soit un domaine de l'informatique, il diffère des approches informatiques traditionnelles. En effet dans cette dernière, les algorithmes sont des ensembles d'instructions explicitement programmées utilisées par les ordinateurs pour calculer ou résoudre des problèmes. Les algorithmes d'apprentissage automatique permettent aux ordinateurs de s'entraîner sur les entrées de données et utilisent l'analyse statistique pour produire des valeurs qui se situent dans une plage spécifique. Pour cette raison, l'apprentissage automatique facilite l'utilisation des ordinateurs dans la construction de modèles à partir de données d'échantillonnage afin d'automatiser les processus de prise de décision en fonction des données saisies **[2]**

Dans l'apprentissage automatique, les tâches sont généralement classées en grandes catégories. Ces catégories sont basées sur la façon dont l'apprentissage est reçu ou comment le feedback sur l'apprentissage est donné au système développé. Deux des méthodes d'apprentissage automatique les plus largement adoptées sont l'apprentissage supervisé et l'apprentissage non supervisé.

2. L'apprentissage automatique :

L'apprentissage automatique est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d' « apprendre » à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des

tâches sans être explicitement programmés pour chacune. Plus largement, il concerne la conception, l'analyse, l'optimisation, le développement et l'implémentation de telles méthodes.

Le concept du Machine Learning date du milieu du 20ème siècle. Dans les années 1950, le mathématicien britannique Alan Turing imagine une machine capable d'apprendre, une « Learning Machine ». Au cours des décennies suivantes**, différentes techniques de Machine Learning** ont été développées pour créer des algorithmes capables d'apprendre et de s'améliorer de manière autonome. **[3]**

2.1 L'apprentissage supervisé :

Dans l'apprentissage supervisé, l'ordinateur est fourni avec des exemples d'entrées qui sont étiquetés avec les sorties souhaitées. Le but de cette méthode est que l'algorithme puisse «apprendre» en comparant sa sortie réelle avec les sorties «enseignées» pour trouver des erreurs et modifier le modèle en conséquence. L'apprentissage supervisé utilise donc des modèles pour prédire les valeurs d'étiquettes sur des données non étiquetées supplémentaires.

2.2 L'apprentissage non supervisé :

Dans l'apprentissage non supervisé, les données sont non étiquetées, de sorte que l'algorithme d'apprentissage trouve tout seul des points communs parmi ses données d'entrée. Les données non étiquetées étant plus abondantes que les données étiquetées, les méthodes d'apprentissage automatique qui facilitent l'apprentissage non supervisé sont particulièrement utiles.**[3].**

2.3 L'apprentissage profond :

Deep Learning (en Français, la traduction est : apprentissage profond) est une forme d'intelligence artificielle, dérivée du Machine Learning (apprentissage automatique). .L'apprentissage profond est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires. Ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur, du traitement automatisé du langage. Dans les années 2000, ces progrès ont suscité des investissements privés, universitaires et publics importants, notamment de la part des GAFAM (Google, Apple, Facebook, Amazon, Microsoft).

3. Le réseau neuronal :

La pratique, de tous les algorithmes de DL(Deep Learning) sont des réseaux neuronaux. **[4]** Les réseaux neuronaux, aussi appelés ANN(Artificial Neural Networks), sont des modèles de traitement de l'information qui simulent le fonctionnement d'un système nerveux biologique. C'est similaire à la façon dont le cerveau manipule l'information au niveau du fonctionnement. Tous les réseaux neuronaux sont constitués de neurones inter connectés qui sont organisés en couches.

3.1 Le neurone :

Ce qui forme les réseaux de neurones, ce sont les neurones artificiels inspirés du vrai neurone qui existe dans notre cerveau. Les 2 figures suivantes (*figure 2.1 et figure 2.2*) montrent une représentation d'un neurone réel et d'un neurone artificiel.

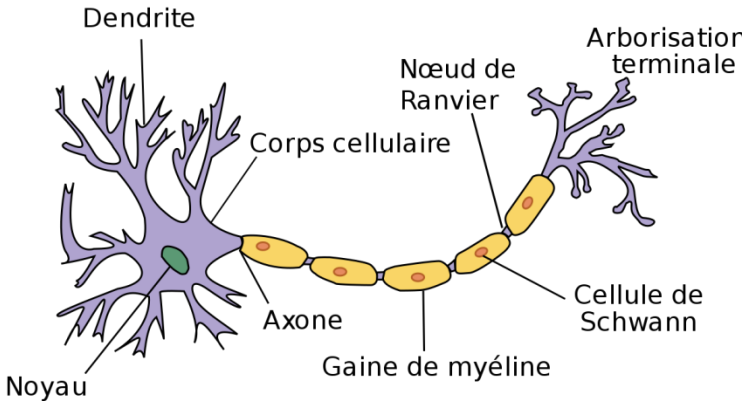


Figure 2.1 : **Représentation d'un réseau de neurone**

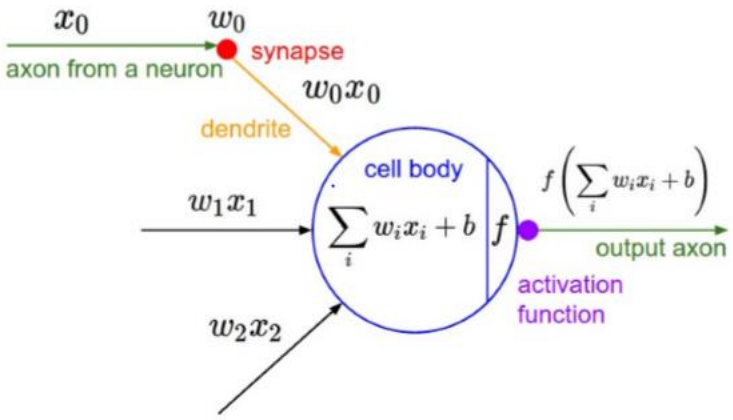


Figure 2.2 : **Le modèle de neurone**

Les x_i sont des valeurs numériques qui représentent soit les données d'entrée, soit les valeurs sorties d'autres neurones. Les poids w_i sont des valeurs numériques qui représentent soit la valeur de puissance des entrées, soit la valeur de puissance des connexions entre les neurones. Il existe des opérations qui se passent au niveau du neurone artificiel. Le neurone artificiel fera un produit entre le poids (w) et la valeur d'entrée (x), puis ajouter un biais (b), le résultat est transmis à une fonction d'activation (f) qui ajoutera une certaine non-linéarité.

4. Les fonctions d'activation :

Après que le neurone a effectué le produit entre ses entrées et ses poids, il applique également une non-linéarité sur ce résultat. **[5]** Cette fonction non linéaire s'appelle la fonction d'activation. La fonction d'activation est une composante essentielle du réseau neuronal. Ce que cette fonction a décidé est si le neurone est activé ou non. Il calcule la somme pondérée des entrées et ajoute le biais. C'est une transformation non linéaire de la valeur d'entrée.

Après la transformation, cette sortie est envoyée à la couche suivante. La non-linéarité est si importante dans les réseaux de neurones, sans la fonction d'activation, un réseau de neurones est devenu simplement un modèle linéaire.

5. Conclusion :

Dans ce chapitre, nous avons abordé les notions d'apprentissage général, ainsi que l'apprentissage approfondie (DL) qui font partie de l'intelligence artificielle, nous avons défini le réseau de neurone qui est l'élément de base du CNN, le chapitre suivant abordera les CNN en détails ainsi que leur topologie afin de proposer une implémentation à base d'FPGA.

Chapitre 3 : CNN et FPGA

1. Introduction :

L'une des capacités fondamentales de l'humain est celle d'analyser son environnement. Dans la majorité des cas, cela passe par la reconnaissance des éléments de notre champ de vision : trouver les autres personnes, identifier les voitures, les animaux…

Jusqu'à l'émergence des réseaux de neurones convolutifs, en 2012 avec *Alex Krizhevsky*, la tâche était difficile pour un ordinateur. **[6]** Heureusement, l'approche de ces réseaux inspirés de notre œil (plus particulièrement du fait que certains neurones de notre aire visuelle ne réagissent qu'aux bordures verticales et d'autres aux horizontales/diagonales) a ouvert de nombreuses applications, que ce soit en imagerie médicale, véhicules autonomes, reconnaissance faciale, et même analyse de textes.

Un réseau de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN ou ConvNet pour Convolutional Neural Networks) est un type de réseau de neurones artificiels acycliques (feed-forward), dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

2. Les différentes couches du CNN :

Un réseau de neurones à convolution peut avoir plusieurs dizaines, voire plusieurs centaines de couches, qui apprennent chacune à identifier différentes caractéristiques d'une image. **[7]** Des filtres sont appliqués à chaque image utilisée pour l'apprentissage à différentes résolutions, et la sortie de chaque image convoluée est utilisée comme entrée de la couche suivante. Les premiers filtres peuvent être des caractéristiques très simples, comme par exemple la luminosité ou les bords, puis passer à des caractéristiques plus complexes qui définissent l'objet de façon unique. A l'instar des autres réseaux de neurones, les CNN se composent d'une couche d'entrée, d'une couche de sortie et de nombreuses couches cachées entre ces deux couches (figure 3.1). **[7]**

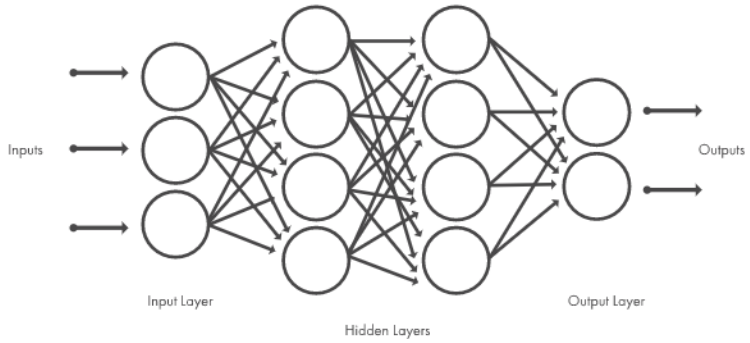


Figure 3.1 **: vue générale des couches CNN**

Toutes ces couches effectuent des opérations qui modifient les données dans l'objectif d'apprendre les caractéristiques spécifiques à ces données. Les trois types de couches les plus répandus sont les suivants : convolution, activation ou ReLU et pooling.

2.1. Convolution :

La convolution est la partie la plus importante des CNN, elle va opérer sur une image en 3 dimensions, qui a donc une longueur, largeur et profondeur, la profondeur peut être égale 0 dans le cas où l'image est en noir et blanc, sinon égale à 3 si elle est en couleur (R G B), en d'autres termes notre image est soit une matrice 2 dimensions si elle est en noir et blanc, soit un bloc a 3 dimensions si elle est en couleurs.

La convolution consiste à faire dérouler un filtre de taille donnée sur cette image, la question qui se pose maintenant est : qu'est-ce qu'un filtre ? Si on prend un exemple d'une image en couleurs (profondeur =3), alors un filtre est lui aussi un bloc de largeur et hauteur inférieures de celle de l'image, mais de même profondeur, ce filtre, on va le faire passer sur le bloc de départ (image).

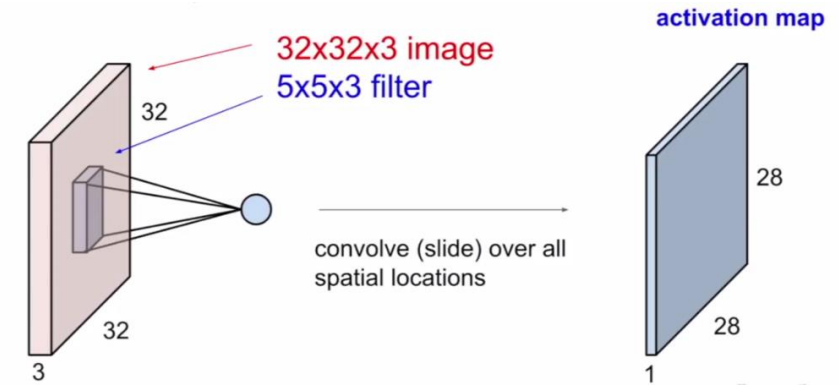


Figure 3.2 : **L'utilisation des filtres pour obtenir des cartes d'activation**

La formule pour calculer tenseur de la sortie du feature maps de la couche est donnée par :

$$f^{(l)}[n,i,j] = b^{(l)}[n] + \sum_{c=1}^{C}\sum_{p=1}^{K}\sum_{q=1}^{K} \Phi^{(l)}[c, i+p, j+q].w^{(l)}[n,c,p,q]$$

∀l = 1 : L (nombres de couches de convolution)

∀n = 1 : N (nombres des sorties des featuremaps)

∀i =1 : Ix (ligne des featuremaps)

∀j = 1: Iy (colonnes des featuremaps)

• **f** **(l)** le tenseur de la sortie du featuremaps de la couche (l)

• **b(l)[n]** le biais appliqué au feature n

• **Φ(l)** le tenseur d'entré du featuremaps de la couche (l)

• **w(l)** le tenseur du filtre pré-appris

2.1.1. Le padding :

Parfois, il peut être intéressant de conserver une certaine dimension dans les tailles des images en sortie des convolutions. Le **padding** consiste simplement à ajouter des 0 tout autour d'une matrice (image) pour en augmenter la taille.

Par exemple, un **padding** de 2 sur une matrice de taille 32×32 ajoutera des 0 à gauche sur 2 colonnes, à droite sur 2 colonnes, en haut sur 2 lignes et en bas sur 2 lignes !

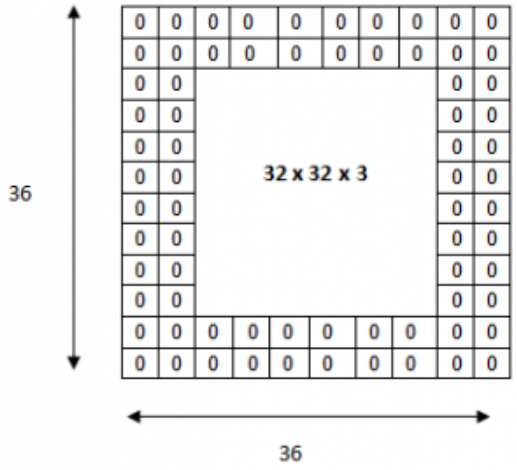


Figure 3.3 : Un padding de 2 sur une matrice de taille 32×32

2.1.2. Le pas (stride) :

Il contrôle le chevauchement des champs récepteurs. Plus le pas est petit, plus les champs récepteurs se chevauchent et plus le volume de sortie sera grand.

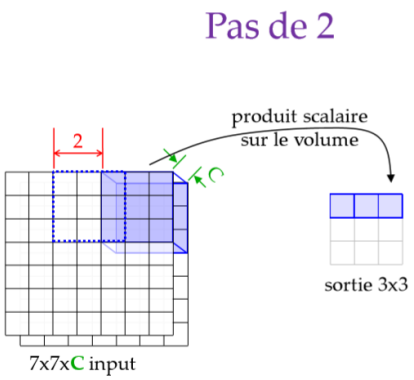


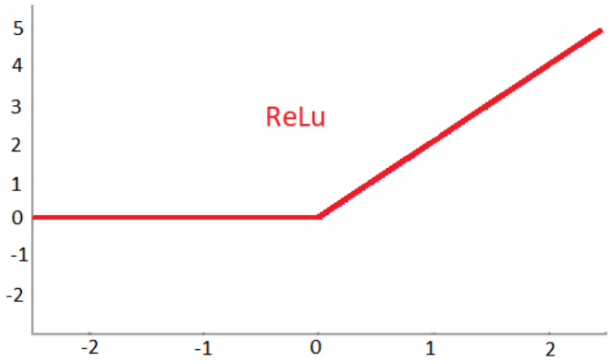
Figure 3.4 : **Un exemple de pas de 2**

La formule pour calculer le nombre de neurones du volume de sortie est :

$$w_0 = \frac{w_i - k + 2P}{s} + 1$$

2.2. La fonction ReLu :

ReLU est une opération élément par élément (appliquée par pixel) et remplace toutes les valeurs de pixels négatives dans la carte d'entités par zéro. Le but de ReLU est d'introduire la non-linéarité dans notre ConvNet, car la plupart des données du monde réel que nous voudrions que notre ConvNet apprenne serait non linéaire (la convolution est une opération linéaire - multiplication et addition matricielles par élément, donc nous tenir compte de la non-linéarité en introduisant une fonction non linéaire comme ReLU). **[8]**



La formule à appliquer est donnée par :

$$f(x) = \max(0, x)$$

2.3. Le pooling :

Le pooling est une spécificité des réseaux de neurones convolutionnels. Les deux méthodes les plus utilisées pour appliquer cette opération sont les suivantes, soit on fait la moyenne des valeurs de la zone (poolingaverage), soit on extrait uniquement la valeur la plus élevée (pooling max). **[9]** Sa fonction est de réduire progressivement la taille spatiale de la représentation pour réduire le nombre de paramètres et de calculs dans le réseau sans perdre les informations les plus importantes, et donc de contrôler également l'overfitting. Le principal avantage du poolingaverage est qu'il est efficace lorsque l'on souhaite détecter des signaux faibles comme pour le cas de la stéganalyse. **[9]** Le pooling max est quant à lui efficace lorsque l'on veut détecter des signaux forts, comme des objets par exemple, de plus il permet au modèle d'être invariant aux translations. Après cette étape de sous-échantillonnage nous obtenons une featuremap qui est définie comme :

- I (l) k = pool(s (l) k),
- avec I (l) k une featuremap de la couche l
- pool () l'opération de pooling
- s (l) k la valeur de sortie du neurone k de la couche l.

2.3.1 Exemple de max pooling :

Un exemple d'opération max pooling est donné par la figure suivante :

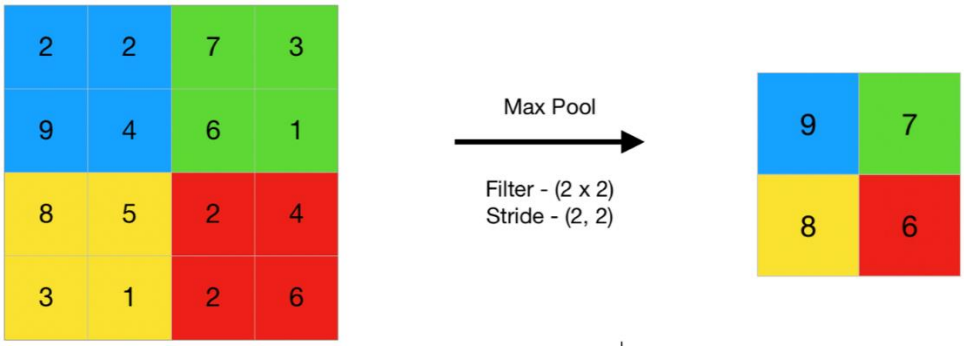


Figure 3.5 : **exemple Maxpooling**

- ∀l = 1 : L (nombre de couches de pooling)
- ∀n = 1 : N (nombres de sorties featuremaps)

- ∀i =1 : Ix (ligne du featuremap)
- ∀j = 1 : Iy (colonne du featuremap)

La fonction pour le calcul est donnée par :

$$f^{(l)}[n,i,j] = \max_{p,q\in[1:K]} \left(\Phi^{(l)}[n,i+p,j+q]\right)$$

2.3.2 Exemple de averagepooling :

Un deuxième exemple d'opération averagepooling est donné par la figure suivante :

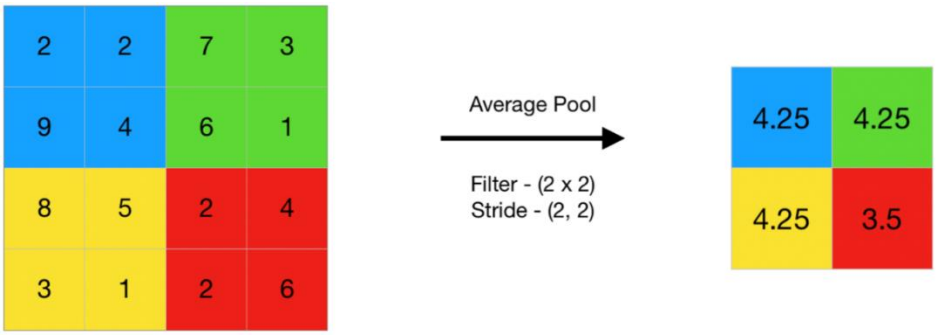


Figure 3.6 : **exemple Average pooling**

2.4. Couche entièrement connectée (fully connected) :

La couche entièrement connectée est un Perceptron multicouche traditionnel qui utilise une fonction d'activation softmax dans la couche de sortie (d'autres classificateurs comme SVM peuvent également être utilisés, mais s'en tenir à softmax dans ce post). Le terme « entièrement connecté » implique que chaque neurone de la couche précédente est connecté à chaque neurone de la couche suivante. La sortie des couches convolutionnelle et de mise en commun représente des caractéristiques de haut niveau de l'image d'entrée. Le but de la couche entièrement connectée est d'utiliser ces fonctionnalités pour classer l'image d'entrée dans différentes classes en fonction de l'ensemble de données d'apprentissage.

La fonction pour le calcul est donnée par :

$$f^{(l)}[n] = \text{act}\left[b^{(l)}[n] + \sum_{c=1}^{C^{(l)}} < \phi^{(l)}[c], w^{(l)}[n,c] >\right]$$

2.4.1. La fonction softmax :

Softmax est une fonction d'activation spécialisée pour les réseaux de classification encodés en one-of-N. Elle réalise une exponentielle normalisée (c'est-à-dire que la somme

des sorties totalise 1). En combinaison avec la fonction d'erreur d'entropie croisée, elle permet de modifier les réseaux **perceptrons multicouches** pour l'estimation des probabilités de classes.

La fonction est définie par :

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ pour tout } j \in \{1, \dots, K\}.$$

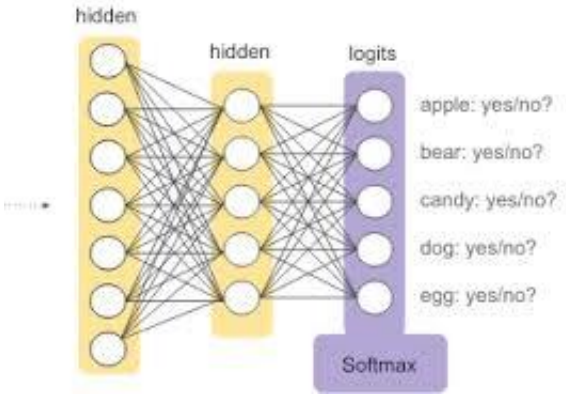


Figure 3.7 : **La fonction softmax.**

3. Les topologies des CNN :

Malheureusement, tous les aspects des CNNs ne sont pas aussi intuitifs à apprendre et à comprendre. Ainsi, il y a toujours une longue liste de paramètres qui doivent être définis manuellement pour permettre au CNN d'avoir de meilleurs résultats, par exemple combien de caractéristiques doit on choisir et combien de pixels doit-on considérer dans chaque caractéristique etc.

En plus de ces paramètres, il y a aussi d'autres éléments architecturaux de plus haut niveau. Certains modèles d'apprentissage profond peuvent avoir plus d'une centaine de couches, ce qui rend le nombre de possibilités extrêmement important. Dans ce principe, au cours des deux dernières années, de nombreuses topologies CNN ont été introduites pour leur précision et leurs performances. La classification d'images est une tâche intéressante et considérablement la plus difficile en vision par ordinateur, où le solveur doit soit étiqueter des images, identifier des objets dans des images ou regrouper des images de mêmes caractéristiques dans des groupes similaires. Cette tâche semble être résolue efficacement en utilisant les réseaux de neurones convolutifs. Un concours de classification d'images appelé le Défi de reconnaissance visuelle à grande échelle ImageNet (ILSVRC) est organisé chaque

année, où les participants sont en concurrence avec leurs algorithmes CNN développés pour classer les images de la base de données ImageNet. La base de données ImageNet comprend plus de 14 millions d'images, chacune étant étiquetée avec une classe correspondante. L'ensemble de formation de l'ILSVRC comprend environ 1,2 million d'images dans 1000 classes différentes.

3.1. Exemples topologies des CNN :

La section suivante résume les topologies CNN les plus utilisées.

3.1.1. LeNet-5 (1998)

LeNet-5, un réseau convolutionnel à 7 niveaux pionnier de LeCun et al en 1998, qui classe les chiffres, a été appliqué par plusieurs banques pour reconnaître les nombres manuscrits sur les chèques (chèques) numérisés en images d'entrée en niveaux de gris de 32 x 32 pixels. La capacité de traiter des images à plus haute résolution nécessite des couches plus grandes et plus convolutives, de sorte que cette technique est limitée par la disponibilité des ressources informatiques. **[10]**

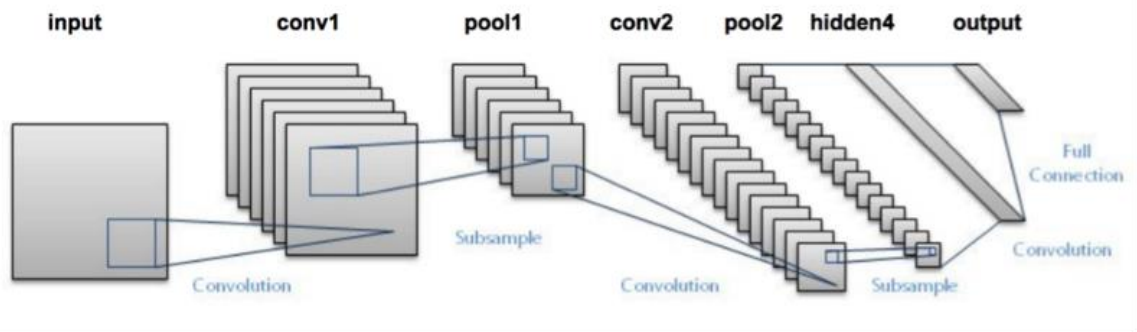


Figure 3.8 : **Architecture LeNet 5**

3.1.2. AlexNet (2012) :

AlexNet est le nom de l'architecture d'un réseau profond de neurones à convolutions qui a remporté l'épreuve ImageNet en 2012. Conçu par une équipe de l'Université de Toronto dirigée par Geoffrey Hinton, dont faisaient partie Alex Krizhevsky, architecte principal qui lui a donné son nom, et Ilya Sutskever, AlexNet a marqué un point tournant dans l'emploi des réseaux profonds de neurones.**[11].**

L'architecture de AlexNet est détaillée dans la figure 3.10.

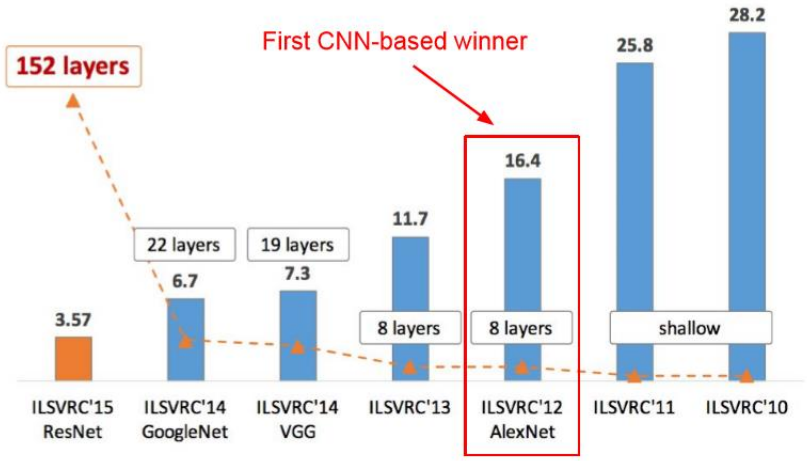


Figure 3.9 : **Classement d'architecture.**

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

Tableau 1 : **Paramètres de l'architecture AlexNet**

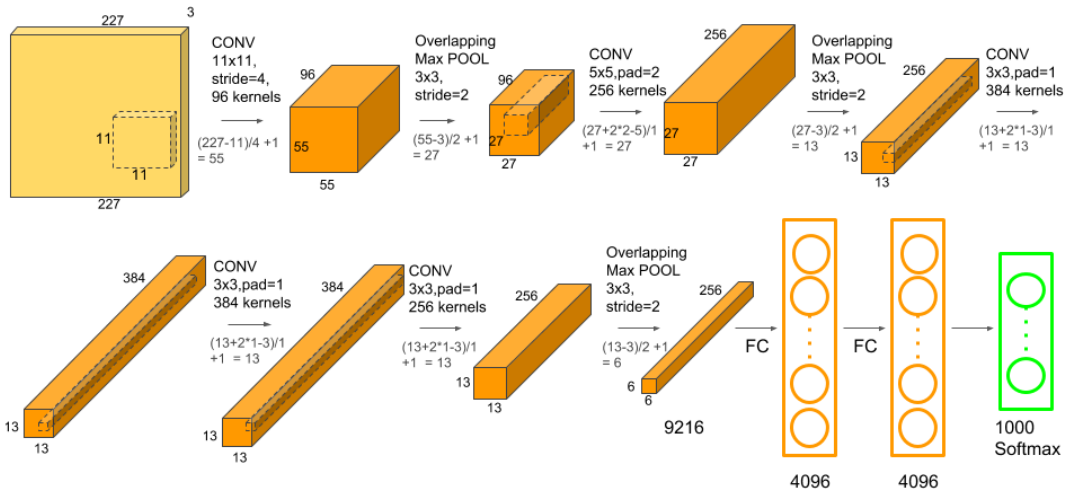


Figure 3.10 : **Architecture AlexNet**

3.1.3. VGGNet (2014)

Le finaliste du concours ILSVRC 2014 est surnommé VGGNet par la communauté et a été développé par Simonyan et Zisserman. VGGNet se compose de 16 couches convolutives et est très attrayant en raison de son architecture très uniforme. Semblable à AlexNet, seulement 3x3 convolutions, mais beaucoup de filtres. Formé sur 4 GPU pendant 2 à 3 semaines. C'est actuellement le choix le plus préféré dans la communauté pour extraire des fonctionnalités d'images. La configuration de poids du VGGNet est accessible au public et a été utilisée dans de nombreuses autres applications et défis comme extracteur de fonctionnalités de base. Cependant, VGGNet se compose de 138 millions de paramètres, ce qui peut être un peu difficile à gérer.

Figure 3.11 : **Architecture VGGNet**

3.1.4. ResNet (2015) :

Enfin, lors de l'ILSVRC 2015, le soi-disant Residual Neural Network (ResNet) par Kaiming He et al a introduit une architecture anovel avec «skip connections» et présente une forte normalisation par lots. De telles connexions de saut sont également connues sous le nom

d'unités fermées ou d'unités récurrentes fermées et ont une forte similitude avec les éléments réussis récents appliqués dans les RNN. Grâce à cette technique, ils ont pu former un NN avec 152 couches tout en ayant une complexité inférieure à VGGNet. Il atteint un taux d'erreur parmi les 5 premiers de 3,57%, ce qui bat les performances au niveau humain sur cet ensemble de données. **[12].**

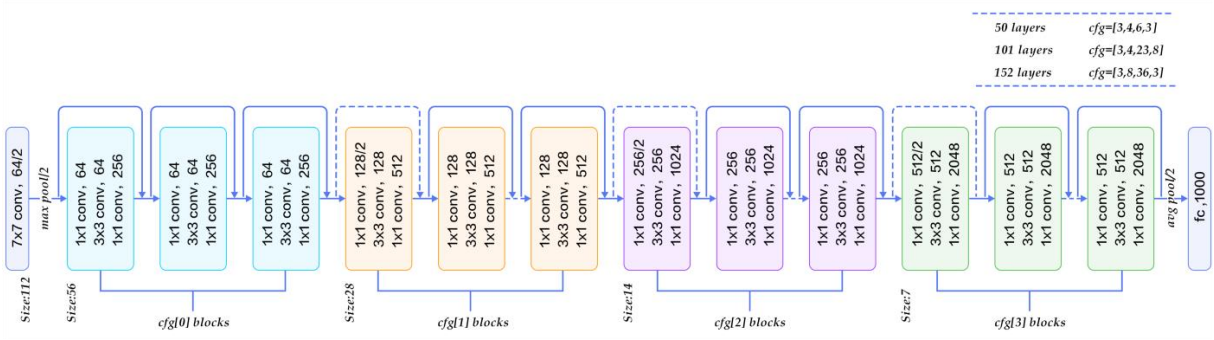
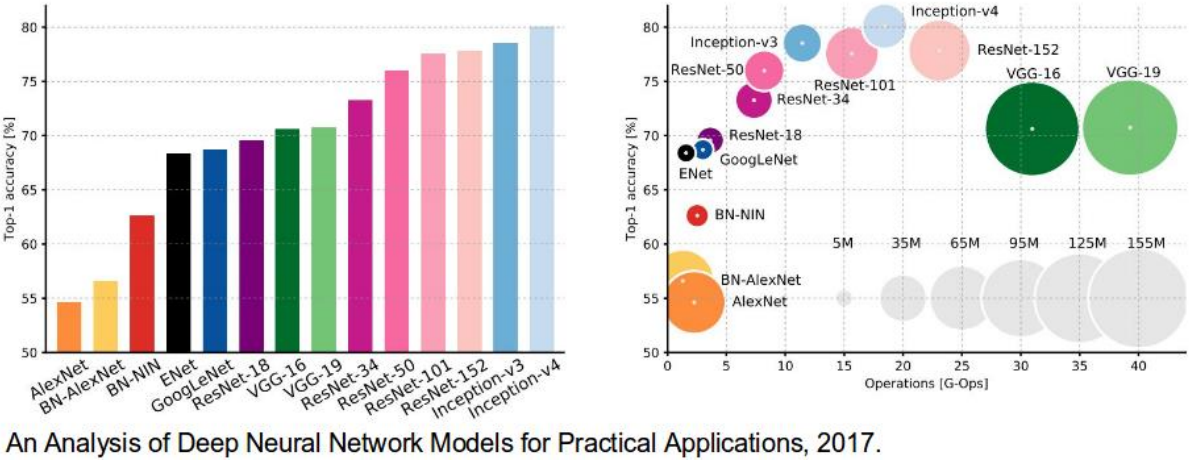


Figure 3.12 : **Architecture ResNet**



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figure 3.13 : **Analyse des modèles des réseaux de neurones profonds pour une application pratique 2017**

Le tableau 2 résume les topologies cités.

Nous avons bien défini les différentes couches d'un réseau CNN, ainsi que les topologies existantes, dans notre travail d'implémentation d'un réseau CNN, il est important de définir les éléments d'un FPGA essentiels pour réaliser cette implémentation.

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Tableau 2 :**Liste des différentes topologies CNN**

4. L'architecture d'une carte FPGA :

Chaque fournisseur FPGA a sa propre architecture FPGA, mais en termes généraux, ils sont tous une variation de celle illustrée sur la figure 1. L'architecture se compose de blocs logiques configurables, de blocs d'E / S configurables et d'une interconnexion programmable. En outre, il y aura des circuits d'horloge pour piloter les signaux d'horloge vers chaque bloc logique. Des ressources logiques supplémentaires telles que des ALU, de la mémoire et des décodeurs peuvent également être disponibles.

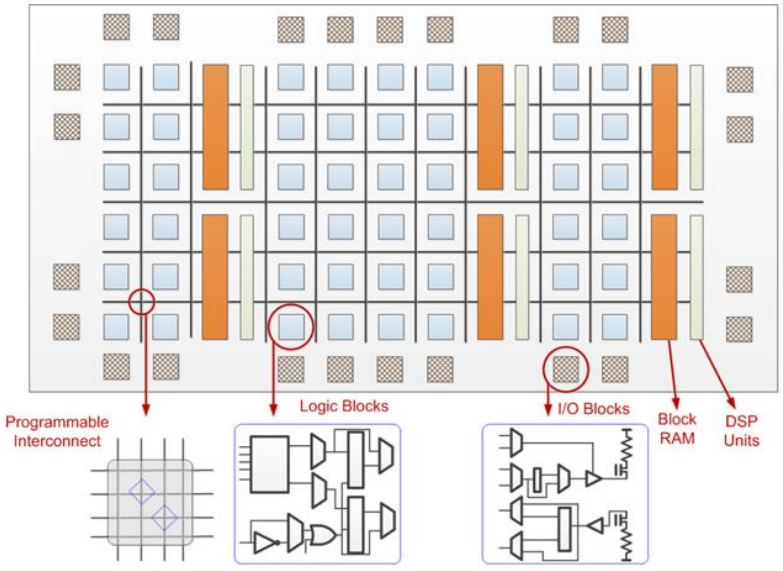


Figure 3.14 : **Architecture FPGA générique**

4.1. Blocs logiques configurables (CLB):

Ces blocs contiennent la logique du FPGA. Dans l'architecture à gros grain utilisée par tous les fournisseurs de FPGA aujourd'hui, ces CLB contiennent suffisamment de logique pour créer une petite machine à états comme illustré sur la figure 3.15. Le bloc contient de la RAM pour créer des fonctions logiques combinatoires arbitraires, également appelées tables de recherche (LUT).**[13]** Il contient également des bascules pour les éléments de stockage cadencés, ainsi que des multiplexeurs afin d'acheminer la logique à l'intérieur du bloc et vers et depuis les ressources externes. Les multiplexeurs permettent également de sélectionner et de réinitialiser la polarité et d'effacer la sélection d'entrée.

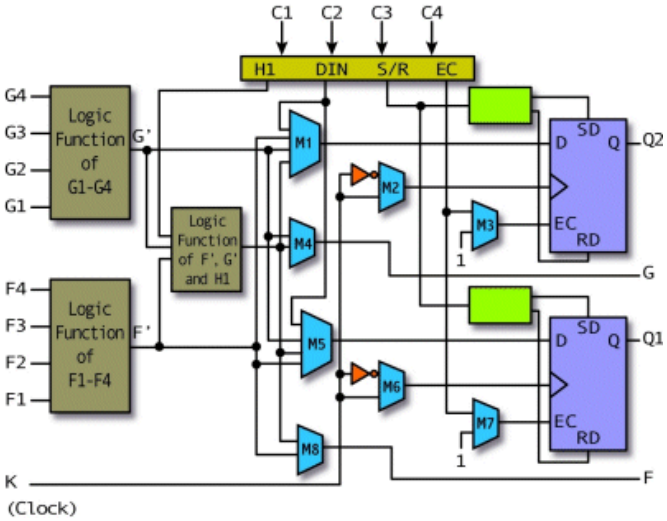


***Figure 3.15:*FPGA Configurable logic block (CLB) (courtesy of Xilinx).**

4.2. LUT (lookup table):

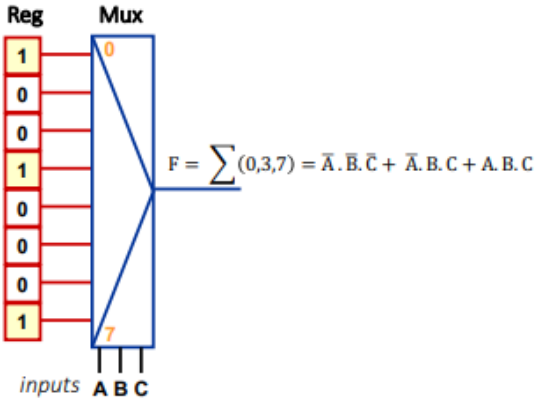


***Figure 3.16:* FPGA LookUp Table.**

Une **LUT**, qui signifie **LookUpTable**, est en termes généraux une table qui détermine la sortie pour n'importe quelle entrée. Dans le contexte de la logique combinatoire, c'est la **table de vérité**. Cette table de vérité définit efficacement le comportement de la logique combinatoire.**[14].**

4.3. Blocs d'E / S configurables:

Un bloc d'entrée / sortie (E / S) configurable, comme illustré sur la figure 3.17 , est utilisé pour acheminer les signaux sur la puce et les renvoyer. Il se compose d'un tampon d'entrée et d'un tampon de sortie avec des commandes de sortie à trois états et à collecteur ouvert. Généralement, il existe des résistances pull up sur les sorties et parfois des résistances pull down qui peuvent être utilisées pour terminer les signaux et les bus sans nécessiter de résistances discrètes externes à la puce.

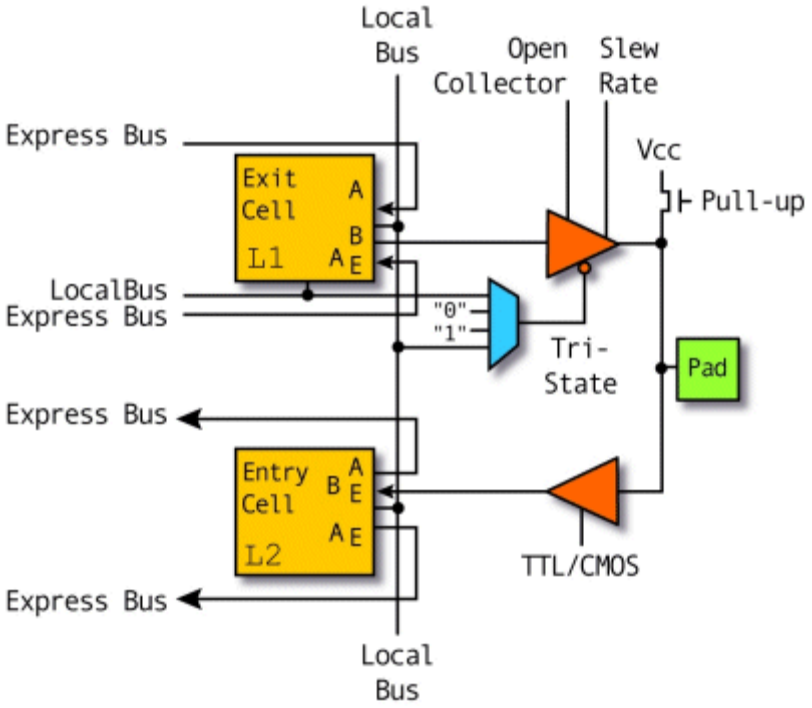


***Figure 3.17 :*d'E / S configurable FPGA (gracieuseté de Xilinx).**

4.4. Interconnexion programmable:

Dans *la figure 3.18*, une hiérarchie des ressources d'interconnexion peut être vue. Il existe de longues lignes qui peuvent être utilisées pour connecter des CLB critiques qui sont

physiquement éloignés les uns des autres sur la puce sans induire beaucoup de retard. Ces longues lignes peuvent également être utilisées comme bus dans la puce.

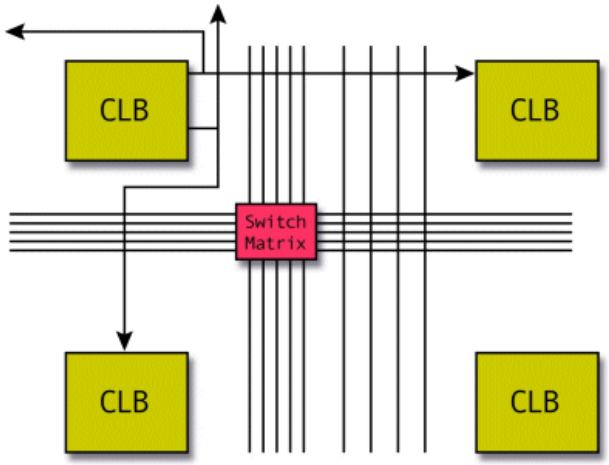


Figure 3.18 :**Interconnexion programmable FPGA (gracieuseté de Xilinx).**

4.5. Circuit d'horloge :

Des blocs d'E / S spéciaux avec des tampons d'horloge à entraînement élevé spéciaux, appelés pilotes d'horloge, sont répartis autour de la puce. Ces tampons se connectent aux pads d'entrée d'horloge et entraînent les signaux d'horloge sur les lignes d'horloge globales. Ces lignes d'horloge sont conçues pour des temps d'inclinaison faibles et des temps de propagation rapides. Notez que la conception synchrone est une obligation avec les FPGA, car le décalage absolu et le retard ne peuvent être garantis nulle part ailleurs que sur les lignes d'horloge globales.**[15]**

4.6. SRAM vs Antifuse vs. Flash :

Il existe trois technologies concurrentes pour la programmation des FPGA. **[16]** La programmation SRAM implique un petit bit RAM statique pour chaque élément de programmation. L'écriture du bit avec un zéro désactive un interrupteur, tandis que l'écriture avec un active un interrupteur. Une autre méthode implique un antifusible constitué d'une structure microscopique qui, contrairement à un fusible ordinaire, ne fait normalement aucune connexion. Une grande quantité de courant pendant la programmation de l'appareil provoque la connexion des deux côtés de l'antifusible. Une troisième méthode, relativement nouvelle, utilise des bits EPROM flash pour chaque élément de programmation.

4.7. Technologies émergentes Noyaux:

Lorsqu'on parle d'un «noyau», on fait simplement référence à une grande fonction autonome. Il existe deux types de base de cœurs. Le noyau souple, appelé noyau IP, est une fonction qui est décrite par sa fonction logique plutôt que par n'importe quelle implémentation physique. Les noyaux logiciels sont généralement constitués de code HDL (Hardware Description Language). Les noyaux durs, en revanche, consistent en des implémentations physiques d'une fonction. En ce qui concerne les FPGA, ces noyaux durs sont appelés noyaux intégrés car ils sont physiquement intégrés dans la puce et entourés d'une logique programmable.

4.8. Cœurs de processeur:

Les cœurs de processeur sont l'un des types de cœurs couramment disponibles en tant que cœurs IP(IntellectualProperty) ou noyaux intégrés. Ces processeurs sont généralement ceux qui sont conçus pour les systèmes embarqués car, presque par définition, les appareils programmables sont des systèmes embarqués.

4.9. Cœurs DSP:

Les processeurs de signaux numériques (DSP) sont un autre type commun de cœur qui est proposé sous forme de cœur IP ou de cœur intégré. Ce sont essentiellement des processeurs spécialisés qui sont utilisés pour manipuler des signaux analogiques. Ils sont couramment utilisés pour le filtrage et la compression des signaux vidéo ou audio.Suivant le travail réalisé, il est important de définir le type de DSP utilisé lors de la conception du réseau CNN à base d'FPGA.

4.9.1. DSP48E :

LeDSP48E est un élément logique de traitement du signal numérique inclus dans certaines familles de périphériques FPGA, comme le Xilinx Virtex-5. Nous avons utilisé cet élément pour effectuer différents types d'opérations arithmétiques, notamment un multiplicateur-accumulateur, un multiplicateur-additionneur et un compteur à une ou *n* étapes.**[17]** Nous pouvons également utiliser ce DSP pour effectuer différents types d'opérations logiques, telles que les opérations AND, OR et XOR. Il est possible de mettre en cascade plusieurs éléments DSP48E pour implémenter des fonctions plus complexes, y compris des multiplicateurs complexes et des filtres FIR n-tap, sans utiliser de ressources de matrice FPGA supplémentaires.**[17]**

4.9.2. DSP48E1 :

LeDSP48E1 est un élément de traitement du signal numérique disponible uniquement sur certaines familles de dispositifs FPGA Xilinx Virtex-6 et plus. Le DSP48E1 comprend un pré-additionneur avant le multiplicateur et d'autres fonctionnalités qui étendent les fonctionnalités d'un DSP48E. Ce pré-additionneur peut être utile pour implémenter certains algorithmes, tels que les filtres FIR symétriques. **[17]**

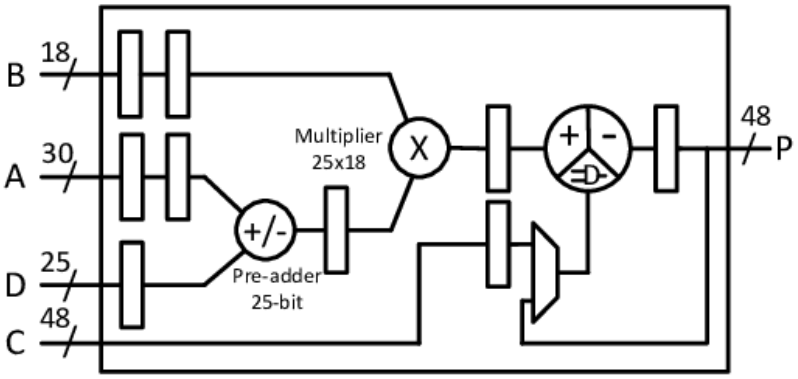


Figure 3.19 : **Architecture interne du DSP48E1**

5. Comparaison FPGA vs GPU :

5.1. Architecture :

Les GPU et FPGA ont une architecture complètement différente. Les GPU sont essentiellement un dispositif informatique extrêmement rapide et efficace composé de nombreux processeurs parallèles. Les GPU sont conçus pour des calculs parallèles (de nombreux ALU parallèles) et un accès rapide à la mémoire. Les FPGA sont constitués d'un ensemble de portes logiques qui peuvent effectuer n'importe quelle implémentation numérique souhaitée par le développeur. Le FPGA peut être un commutateur réseau, un processeur ou un mineur de bitcoin. FPGA offre la flexibilité maximale possible à l'ingénieur numérique.

5.2. Efficacité énergétique :

Les FPGA et les GPU ne sont pas considérés comme des appareils à faible consommation. Mais par rapport aux GPU, les FPGA sont considérés comme une solution plus économe en énergie car les FPGA sont constitués uniquement de fonctions matérielles tandis que les GPU ont tendance à être très énergivores car ils en ont besoin pour faciliter la

programmabilité logicielle. En plus de cela, les FPGA sont venus en plusieurs tailles afin que le concepteur puisse choisir une taille de périphérique qui s'adapte parfaitement à l'application.

5.3. Puissance de calcul :

Les FPGA sont des puces programmables qui peuvent être conçues pour implémenter toute fonction numérique ou tout calcul par matériel. Cela signifie que les FPGA peuvent être programmés pour implémenter des GPU. Bien que cela soit correct, il est important de noter que les fonctionnalités de programmabilité du FPGA ajoutent des retards à toutes les portes logiques et au routage interne. Cela signifie que dans certains cas, un GPU pourrait être plus rapide et devenir une machine de traitement plus puissante qu'un FPGA.

5.4. Programmation :

Les FPGA peuvent être programmés à l'aide du langage de description du matériel ou HDL tels que VHDL et Verilog. Les GPU peuvent être configurés à l'aide de langages de programmation de logiciels à usage général, y compris C, C ++, Java, Python, etc.

5.5. La flexibilité :

Les FPGA ont tendance à avoir une architecture plus flexible que les GPU. Avec les GPU, vos capacités de reprogrammation sont limitées car le flux de données dans ces structures est déterminé par le logiciel et est dirigé par une hiérarchie complexe de la mémoire interne du GPU.

5.6. Ajouts périphériques :

Les FPGA peuvent être connectés à presque toutes les puces numériques ou analogiques telles que les interfaces d'émetteur-récepteur, les convertisseurs, les interfaces sans fil, ainsi que plusieurs autres périphériques pour faciliter son utilisation. Les GPU ne prennent généralement pas en charge une multitude d'ajouts périphériques et sont largement limités aux goûts de la mémoire cache.

5.7. Commodité :

Les GPU, comparativement, sont plus faciles à utiliser que les FPGA, car le processus de développement de ces derniers a tendance à être beaucoup plus étendu et compliqué que pour

les premiers, ce qui explique pourquoi les GPU sont maintenant appliqués dans une multitude de domaines de nos jours.

5.8. Prototypage ASIC :

En raison de leur reprogrammabilité et de leur configurabilité, les FPGA sont le choix optimal à des fins de prototypage. Les GPU ne peuvent pas être utilisés pour cette raison. Les ASIC sont permanents et coûteux à fabriquer, c'est pourquoi la conception électronique doit d'abord être testée via un FPGA afin que toutes les erreurs puissent être triées et que des modifications puissent être apportées au lieu d'avoir à réorganiser un ASIC.

5.9. Applications typiques :

Les FPGA sont vénérés pour leurs performances optimales et rentables en plus du fait qu'ils peuvent être configurés plusieurs fois et remplir diverses fonctions tout au long de leur cycle de vie appliqué. En ce qui concerne les GPU, cependant, ils sont mieux adaptés aux applications cibles telles que le traitement vidéo, les calculs à virgule flottante, l'analyse d'images car ils sont chargés avec une puissance de traitement élevée.

	x86	Gpu	Dsp	Fpga	µC
Embed	Maybe	Hard	Easy	Easy	Easy
Low power	Unusual	Nope	Sometimes	Sometimes	Yes
Float op	Good	Excellent	Excellent	Possible	Nope
Int op	Excellent	Excellent	Excellent	Excellent	Mediocore
Control flow	Excellent	Challenging	Fair	Challenging	Excellent
IO	Mediocore	Nope	Ok	Ginormous	Ok
Pipelining	Nope	Nope	Nope	Yes	Nope
Programmability	Easy	Medium	Medium	Challenging	Easy
Timing control	Medium	What?	Fair	Excellent	Fair

Tableau 3 : **Comparaison entre CPU, FPGA, GPU et uC**

6. Conclusion:

Dans ce chapitre, nous avons fait l'étude des différentes couches qui constituent un réseau CNN afin de proposer une implémentation matérielle pour chaque couche, dans une deuxième

partie, nous définissons les topologies CNN existantes ainsi que leur architecture, il est important de choisir l'une des topologies dans le travail d'implémentation proposé. Dans une dernière partie de ce chapitre, nous définissons l'FPGA avec ses éléments de base, les DSP qui sont des éléments de calculs essentiels dans la conception d'un réseau CNN.

Chapitre 4 : Accélération matérielle et implémentation CNN

1. Introduction

Les réseaux de neurones à convolution profonde (CNN) sont les systèmes de pointe pour la classification d'images en raison de leur grande précision, mais d'un autre côté, leur complexité de calcul élevée est très coûteuse. L'accélération est aujourd'hui la cible dans ce domaine pour utiliser ces systèmes dans des applications temps réel. Les unités de traitement graphique sont la solution, mais sa consommation d'énergie élevée empêche son utilisation dans les équipements utilisés quotidiennement. De plus, le FPGA a une faible consommation d'énergie et une architecture flexible qui convient davantage aux implémentations CNN. Ce travail aborde ce problème et fournit une solution qui pourrait remplacer l'exécution par GPU, et qui fait montrer les avantages d'un FPGA par rapport à la vitesse du CNN et la maniabilité de la reconfiguration convenable du FPGA. Cette solution repose sur deux techniques principales d'accélération: le parallélisme des ressources des couches et le pipelining à l'intérieur de certaines couches.. La mise en œuvre du travail d'implémentation d'un réseau de neurones CNN a été initié par le logiciel ISE Xilinx, où nous avons utilisé vivado HLS (High LevelSynthesis) pour le synthèse et l'estimation des ressources occupées par l'FPGA. Les résultats d'implémentation sont résumés en fin de chapitre. D'autre part, Dans l'implémentation proposée, il est important de définir les différents modules du CNN proposés ainsi que leur spécification matérielle, dans la partie qui suit, nous détaillons les différents éléments du CNN proposé.

2. Implémentation matérielle :

La section suivante présente les différents modules CNN réalisés en VHDL en but d'implémentation matérielle sur FPGA. L'architecture générale est montrée sur la figure 4.1.

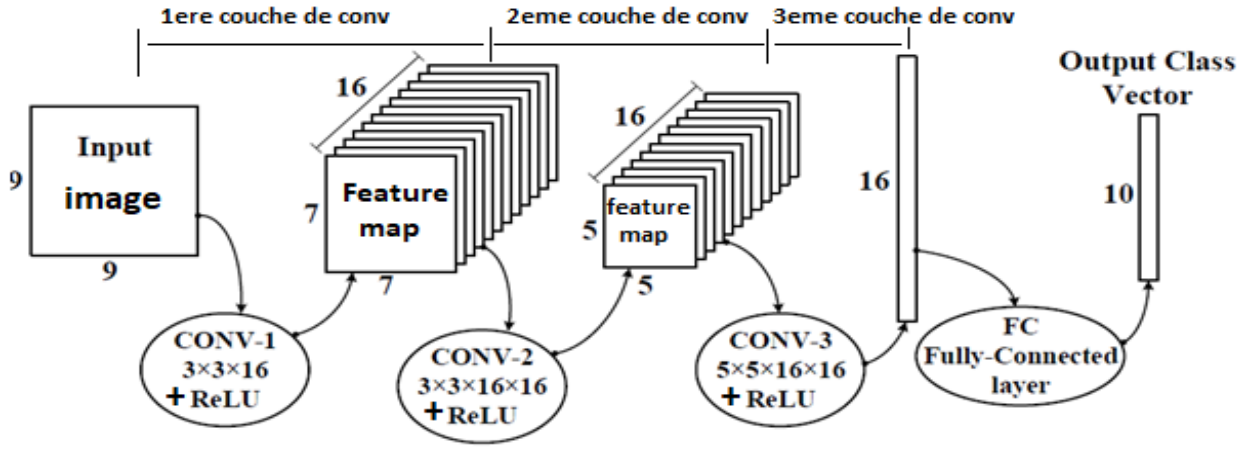


Figure 4.1 : Architecture du CNN implémentée en VHDL

Les modules sont réalisés sous VHDL avec le logiciel de synthèse et de simulation ISE (vivado 2016), la plateforme utilisée est de type FPGA (FPGA SPARTAN 6). Le tableau 5 montre les différents paramètres utilisés dans le type du CNN choisi. Nous détaillerons chaque module du CNN à savoir : la convolution, le Relu, le maxpooling ainsi que le Fully connected. Nous montrerons les résultats de synthèse et de simulation à la fin du chapitre.

Layer		**Featuremap**	**size**	**Kernel size**	**Stride**
Input	Image	1	9x9	-	-
1	Convolution + Relu	16	7x7	3x3	1
2	Convolution + Relu	256	5x5	3x3	1
3	Convolution + Relu	256	1x1	5x5	1
4	Fullyconnected	-	16	-	-
Output	Fullyconnected	-	10	-	-

Tableau 4 : Paramètres de l'architecture CNN implémentée

3. Convolution Layer

La convolution 2D (CONV_2D) est L'opération principale de notre model CNN, afin de l'appliquer nous avons utilisé 3 blocs essentiels qui sont : Conv_2D,FSM,adder tree. Avant d'expliquer le rôle de chaque bloc nous tenons en compte que dans notre projet nous avons décomposé la convolution 2D à plusieurs convolution 1D et pour cela nous avons utilisé le pipelining lié aux réseaux systoliques. Ainsi, les pixels vont traverser le système un par un ce qui va nous permettre d'éviter le prétraitement de données, lié au réarrangement de celles-ci en mémoire ce qui augmenterai la latence. La figure suivante montre le principe de la convolution 2D où w représente le poids du filtre, x représente le pixel de l'image.

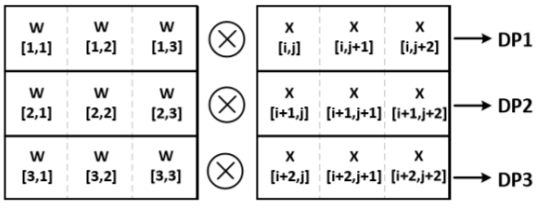


Figure 4.2 : **Convolution de deux matrices 3x3**

Dans notre model d'implémentation (figure 4.1), nous avons utilisé 3 couches de convolution CNN (suivant le model LeNet) :

La 1ere couche CONV1 contient 16 filtres (de type Kernel) d'une taille =3x3.

La 2eme couche CONV2 contient 16 filtres (de type Kernel) de taille =3x3.

La 3eme couche CONV3 contient 16 filtres (de type Kernel) de taille =5x5.

Avant d'expliquer le fonctionnement de chaque module de convolution, nous précisons que l'image est chargée à partir d'une mémoire externe pixel par pixel, et qu'un pixel de l'image d'entrée est représentée par 9bits (8bits pour la valeur du poids et le 9eme pour le signe). Le filtre utilisé est le filtre kernel représenté par 5 bits (4bits pour la valeur du poids et le 5eme pour le signe).

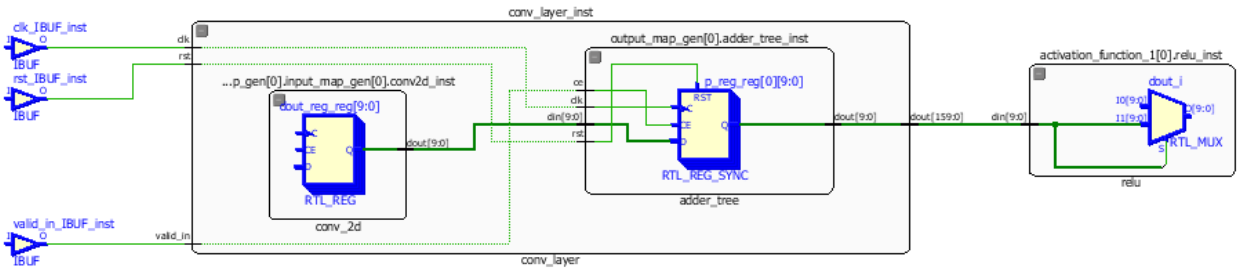


Figure 4.3 : **schéma RTL du module CONV_Layer**

3.1. CONV_2D :

Le bloc CONV_2D est le responsable des opérations mathématiques et contient 3 sous blocs qui sont le SE_CHAIN (SE), SWITCHING BLOC, ADDER(additionneur), DELAY BUFFER(figure 4.4). Ce module est le même utilisé dans les 3 couches de convolution :

- La première couche de convolution contient 16 module CONV_2D, où chaque module CON_2D reçoit respectivement 1 pixel de l'image d'entrée et une ligne de filtres 3x3 utilisés pour la première couche, le bloc SE calcule la convolution entre le pixel de l'image et celui du filtre qui correspond à la même position, une série d'additionneur calcule la somme des résultats de convolution afin de créer les pixels du feature map de taille (7x7) (figure 4.1).
- La deuxième couche de convolution contient 256 (16x16) modules CONV_2D, où chaque module CON_2D reçoit 1 pixel de chacune des 16 images résultantes (feature map 7x7) de la convolution de la première couche et une ligne de filtres 3x3 utilisés pour la deuxième couche, le bloc SE calcule la convolution entre le pixel de l'image et

celui du filtre qui correspond à la même position, une série d'additionneur calcule la somme des résultats de convolution afin de créer les pixels du feature map de taille (5x5) (figure 4.1). Ensuite 16 arbres d'additionneur (adder tree) sont ajoutés à la fin pour favoriser le pipeline qui est mis de façon à calculer la somme des résultats de convolution de la même image pixel par pixel.

- La troisième couche de convolution est similaire à la deuxième couche, elle contient 256 (16x16) modules CONV_2D, où chaque module CON_2D reçoit 1 pixel de chacune des 16 images résultantes (feature map 5x5) de la convolution de la première couche et une ligne de filtres 5x5 utilisés pour la troisième couche, le bloc SE calcule la convolution entre le pixel de l'image et celui du filtre qui correspond à la même position, une série d'additionneur calcule la somme des résultats de convolution afin de créer les pixels du feature map de taille 16 (figure 4.1). L'arbre d'additionneur ajouté à la fin joue le même rôle que précédemment.

La taille du feature map résultant de chaque couche de convolution est calculée suivant cette formule :

$$Size = (w-F+2P)/S+1$$

Où w : la taille de l'image d'entrée, F : la taille du filtre appliqué, P : le pading qui est égal à zéro, S : stride le décalage du filtre lorsqu'il balaye l'image d'entrée qui est égal à 1.

- Les paramètres des filtres pour les différentes couches de convolution sont générés à partir du model CNN adopté après un training [18].

Dans les sections qui suivent, nous les détaillons afin de bien comprendre le fonctionnement de chaque bloc de la CONV_2D.

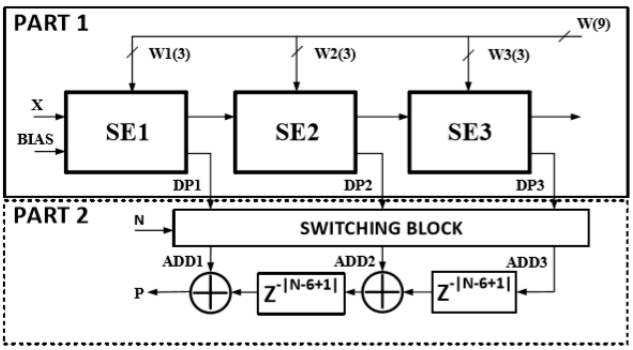


Figure 4.4: **Modèle implémenté pour la convolution 2D**

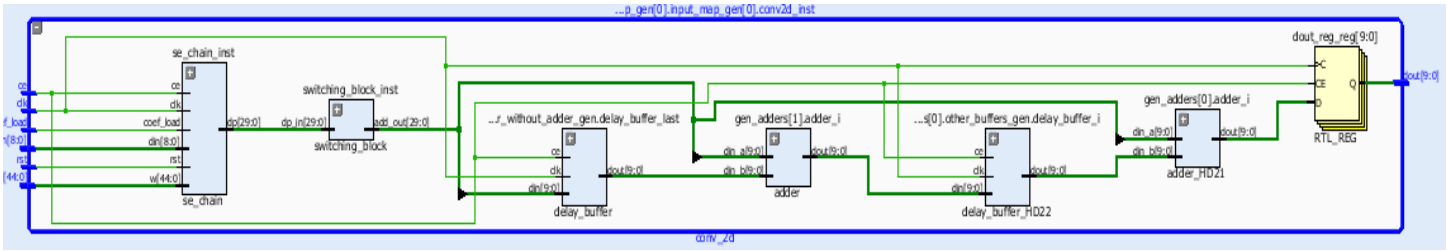


Figure 4.5 : **schéma RTL du module CONV_2D**

3.1.1. SE_CHAIN :

Ce bloc contient 3 blocs nommé SYSTOLIC FIR (figure 4.6), chaque bloc contient 3 DSPs et chaque DSP reçoit 5bits (poids du filtre), donc chaque bloc SYSTOLIC FIR va représenter une ligne du filtre (une ligne de 3 pixels). De cette manière, le bloc SE CHAIN reçoit les 45bits (5x3x3) qui représente le filtre complet. Chacun des DSP reçoit un pixel de l'image et les données du filtre afin d'appliquer l'opération MAC.

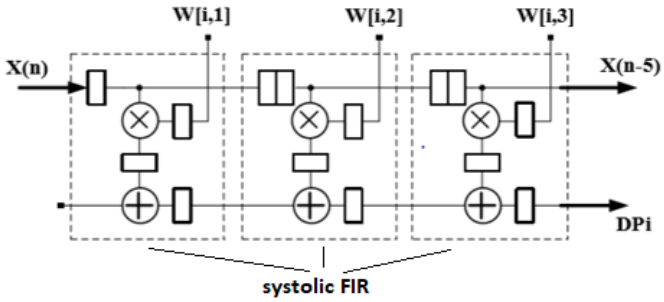


Figure 4.6 :**Modèle du SE_CHAIN**

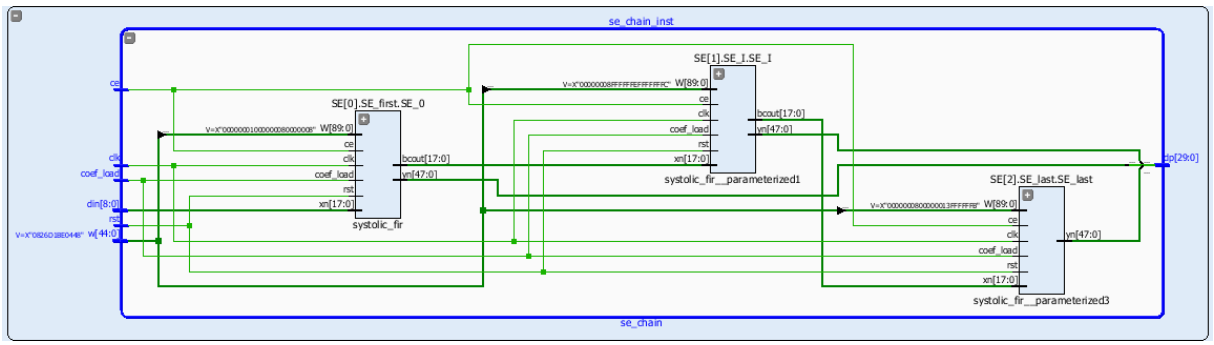


Figure 4.7 : **schéma RTL du module SE_chain**

3.1.2. SWITCHING BLOC :

Afin d'accélérer le calcul, chaque bloc SYSTOLIC FIR va envoyer le résultat de la convolution de chaque ligne du filtre (dpi) au SWITCHIG BLOC pour choisir (à l'aide du multiplexeur) l'additionneur qui va recevoir le résultat ou envoyer le résultat au DELAY BUFFER en attendant l'arrivée du résultat suivant. Le switching bloc règle aussi le problème d'un délai négatif.

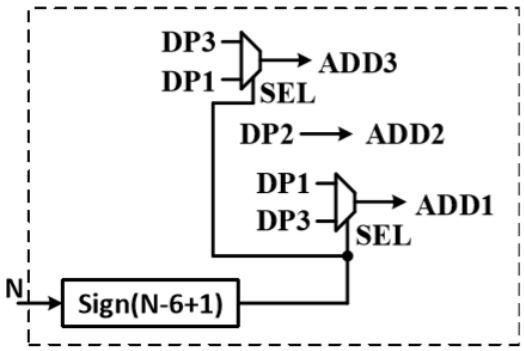


Figure 4.8 : **Modèle du swtitching bloc**

3.1.3. ADDER et DELAY BUFFER :

Le DELAY BUFFER a pour but de contrôler le début de fonctionnement de chaque additionneur (ADDER)pour assurer la bonne combinaison des dpi.

Le délai est calculé de cette manière :

Le délai=N-2K+ 1

N : le nombre des pixels entrant.

K :la taille du filtre.

3.1.4. Finite State Machine (FSM) :

Les machines à états finis (FSM) sont au cœur de la plupart des conceptions numériques. L'idée de base d'un FSM est de stocker une séquence de différents états uniques et une transition entre eux en fonction des valeurs des entrées et de l'état actuel de la machine. Le FSM peut être de deux types: Moore (où la sortie de la machine d'état dépend purement des variables d'état) et Mealy (où la sortie peut dépendre des valeurs des variables d'état actuelles ET des valeurs d'entrée).

La structure générale d'un FSM est illustrée à la figure 4.5

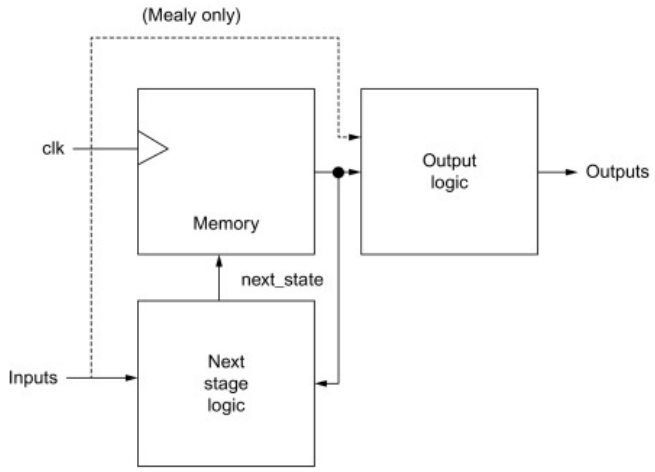


Figure 4.9 : **La structure générale d'un FSM**

4. RectifiedLinear Unit (ReLU) :

La correction ReLU est l'abréviation de Unité Linéaire Rectifiée : f(x)=(0,x). Cette fonction, appelée aussi « fonction d'activation non saturante », augmente les propriétés non linéaires de la fonction de décision et de l'ensemble du réseau sans affecter les champs récepteurs de la couche de convolution.

Elle est utilisée après chaque couche de convolution (CONV_2D), où toutes les valeurs de pixels négatifs sont mises à zéro. Le but de ReLU est d'introduire la non-linéarité dans notre CNN, dans le model d'implémentation réalisé le module Relu contient un multiplexeur 2-1 comme le montre le schéma suivant. (***Figure4.8***)

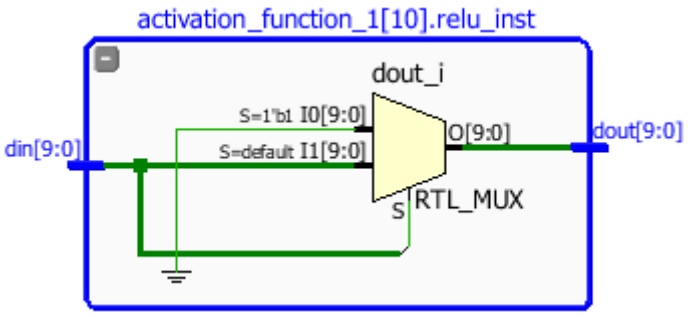


Figure 4.10 : schéma RTL du module Relu

5. Maxpooling_layer :

Après avoir défini de l'étape de convolution et relu, un autre outil très puissant utilisé par les CNNs s'appelle le Pooling. Le MaxPooling est une méthode permettant de prendre une large image et d'en réduire la taille tout en préservant les informations les plus importantes qu'elle contient. Les mathématiques derrière la notion de maxpooling ne sont pas très complexes. En effet, il suffit de faire glisser une petite fenêtre pas à pas sur toutes les parties de l'image et de prendre la valeur maximum de cette fenêtre à chaque pas

Ce module constitue l'étape du pooling, il se compose de deux principaux éléments pour assurer le bon fonctionnement du système :

5.1. Memory_part :

Le rôle principal de ce bloc est de recevoir les informations obtenues de l'étape précédente (Relu) au format pixel et faire traiter cette dernière pour obtenir à chaque fois des nouvelles fenêtres de taille 9 bits (3x3) de l'image principale , tout cela se fait à l'aide d'un ensemble d'outils et des portes logiques se trouvant dans ce module . L'un des plus importants de ces outils s'appelle**s shift_reg**

5.1.1. shift_reg (registre de décalage) :

Un registre à décalage est un registre, c'est-à-dire un ensemble de bascules synchrones, dont les bascules sont reliées une à une, à l'exception de deux bascules qui ne sont pas forcément reliées. À chaque cycle d'horloge, le nombre représenté par ces bascules est mis à jour. Le concept de décalage permet d'insérer une donnée dans le registre, ou la lire, bit par bit en série

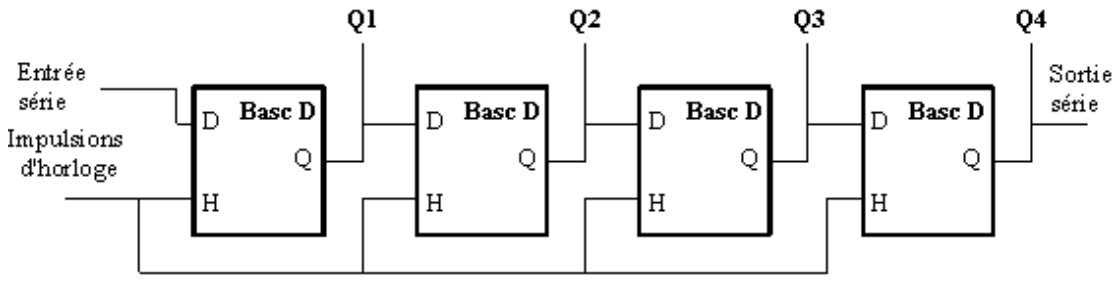


***Figure 4.11 :*Registre à décalage SISO ou SIPO de 4 bits**

Dans notre projet, nous avons 4 registres de décalage de 9 bits il est utilisé afin de décaler une fenêtre de pixel 3x3 d un pas dans toute la surface de l'image principale, puis envoyer les résultats en deux sorties en même temps au module de calcul qui s'appelle **compute_part** pour faire la comparaison nécessaire, ces opérations sont toutes contrôlées par le FSM (figure 4.12).

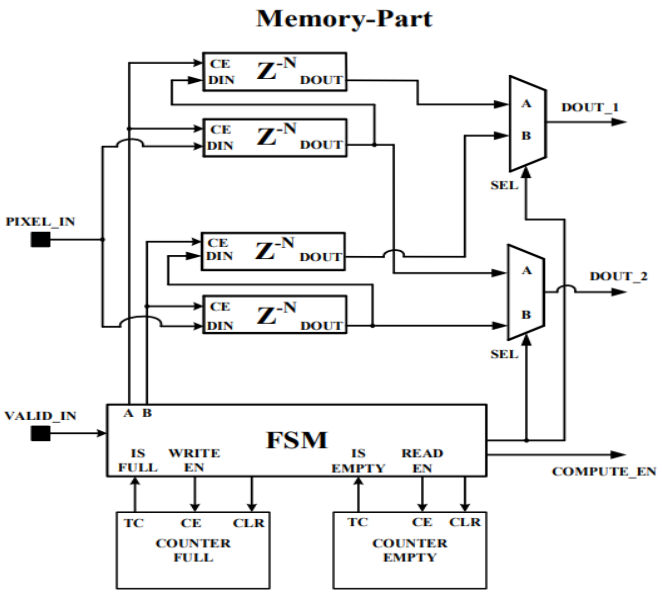


Figure 4.12 : **Architecture interne du memory_part**

5.2. Compute_part :

C'est la partie qui suit memory_part, son rôle principal est de comparer les deux entrées issues de l'étape précédente, afin de pouvoir choisir la valeur maximale. Cela dépend d'un groupe d'outils fonctionnant simultanément (unsignedGreater , Mux , fdre)

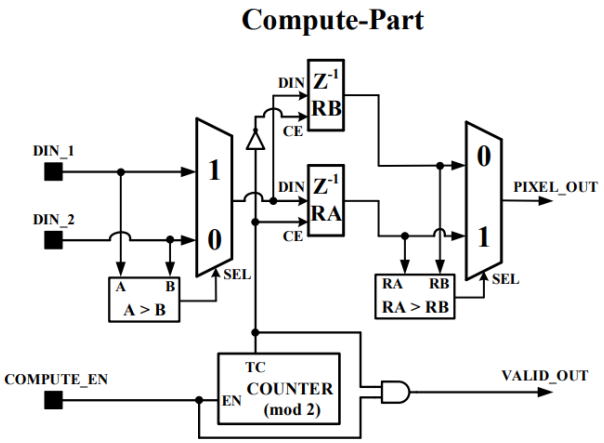


Figure 4.13 : **schéma Compute-Part**

L'outil greater est chargé de comparer les deux entrées pour voir la valeur maximal, si le premier signal est plus grand que le second, alors la sortie de Greater représente (1), sinon, la sortie représente (0), un multiplexeur 2-1 est utilisé afin de choisir l'une des sorties, enfin, la bascule FDRE qui s'active avec une entrée d'activation d'horloge (CE) et des entrées de réinitialisation synchrone (R) remplace toutes les autres entrées et réinitialise la sortie (Q) basse lors de la transition d'horloge basse à haute (C). Les données sur l'entrée D sont chargées dans la bascule lorsque R est Low et CE est High pendant la transition d'horloge Low-to-High

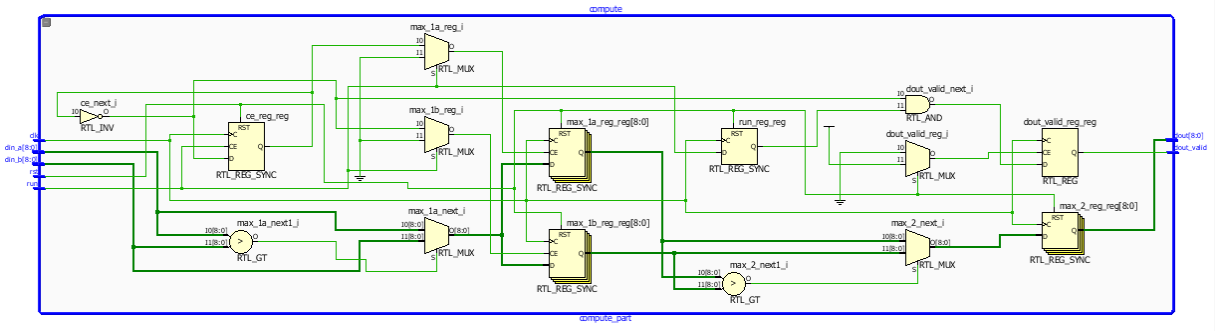


Figure 4.14 : **schéma RTL du Maxpooling**

6. Fullyconnected layer

Cette couche contient 160 multiplicateur et 10 arbres d'additionneur (adder tree), le nombre des filtres utilisé dans cette couche est 10 chaque filtre est d'une taille de 4x4.

Chaque combinaison de 16 multiplicateurs et 1 adder tree va produire un résultat qui s'appelle le score à partir duquel on peut déterminer a quoi correspond l'image d'entrée.

Chaque multiplicateur est affecté par un poids (5bits) et reçoit un pixel (10 bits) de l'étape précédente ensuite il multiplie les données et envoie le résultat au adder tree qui va additionner les 16 résultats pour former le score. Le fully connected est la dernière étape du model CNN implémenté, à travers lequel nous obtenons un seul résultat parmi 10 résultats possibles qui correspond à l'identification de l'image qu'on a fait entrer.

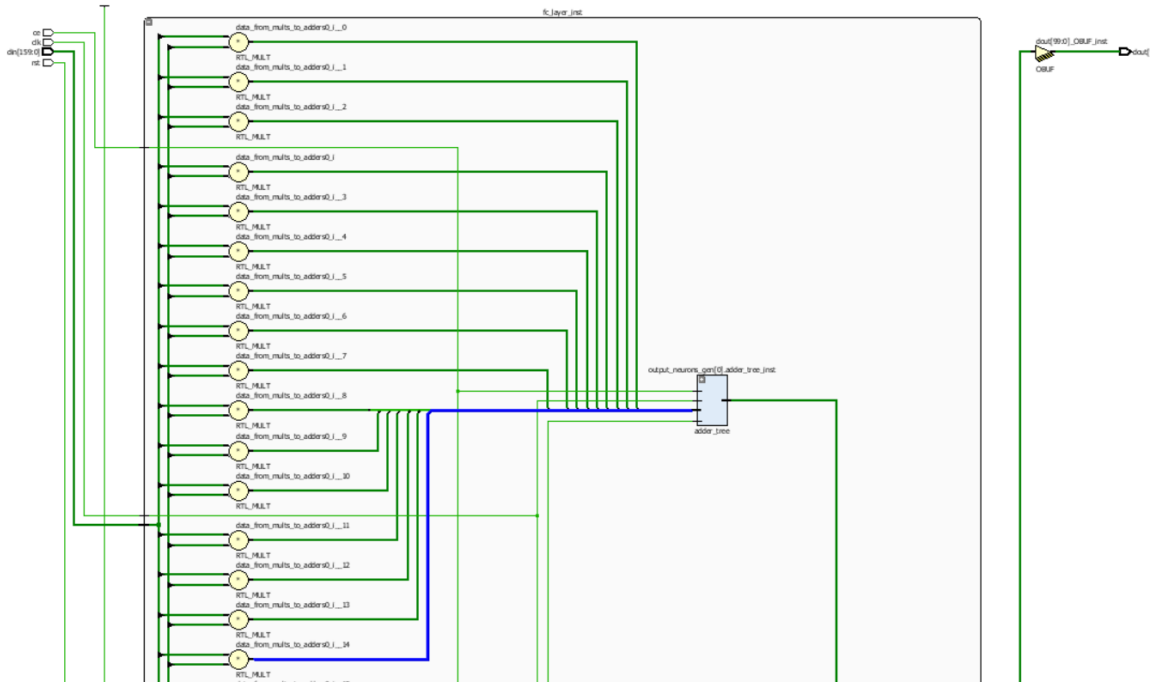


Figure 4.15: schéma **RTL du fully connected**

6.1. ADDER TREE :

La synthèse de haut niveau essaie toujours de construire une structure arborescente équilibrée à partir d'un certain nombre des ajouts qui peuvent être programmés en parallèle. L'équilibrage de l'arbre additionneur tend à réduire la surface d'une conception en minimisant la latence, ce qui à son tour réduit le nombre de registres. L'arbre d'additionneur est utilisé pour la réalisation du module fully connected layer, En effet, le module fully connected fait appliquer plusieurs filtres à l'image donnant des coefficients en résultat, ces coefficients sont additionnés afin d'avoir un coefficient final qui va servir pour la classification dans le CNN.

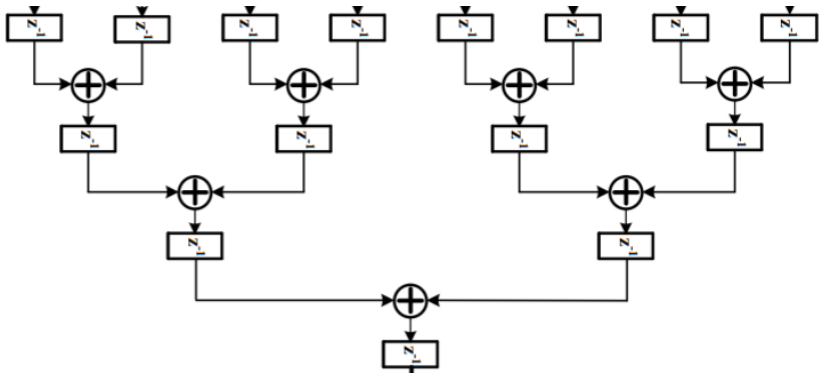


Figure 4.16:**AdderTree**

7. Résultat de synthèse et simulation

Les résultats de synthèse sont réalisés sous vivado 2016, utilisant la carte FPGA de type Spartan6 XA6SLX9. Nous montrons le taux d'utilisation de ressources pour chaque module implémenté, ainsi que sa fréquence de fonctionnement.

7.1 Table d'utilisation des ressources

7.1.1 Conv_2D

Resource	Utilization	Available	Utilization %
FF	819	82000	1.00
LUT	630	41000	1.54
Memory LUT	192	13400	1.43
I/O	173	300	57.67
DSP48	144	240	60.00
BUFG	1	32	3.12

Tableau 5 : **post synthesis Conv_2D**

Maximum Frequency: 537.288MHz

7.1.2 Maxpoling

Resource	Estimation	Available	Utilization %
FF	33	82000	0.04
LUT	43	41000	0.10
I/O	22	300	7.33
BUFG	1	32	3.12

Tableau 6 : **post synthesisMaxpoling**

Maximum Frequency: 227.066MHz

7.1.3 Fc_layer

Resource	Estimation	Available	Utilization %
FF	3953	82000	4.82
LUT	5228	41000	12.75
I/O	263	300	87.67
BUFG	1	32	3.12

Tableau 7 : **post synthesis fc_layer**

7.1.4 CNN_TOP (CONV2D+Relu+FC_layer)

Resource	Estimation	Available	Utilization %
FF	48587	106400	45.66
LUT	37666	53200	70.80
Memory LUT	10561	17400	60.70
I/O	103	200	51.50
BRAM	2.5	140	1.79
DSP48	8848	220	4021.82
BUFG	1	32	3.12

Tableau 8 : **post synthesis CNN_TOP**

Maximum Frequency: 445.297MHz

7.2 Simulation et test bench

7.2.1 Conv_layer :

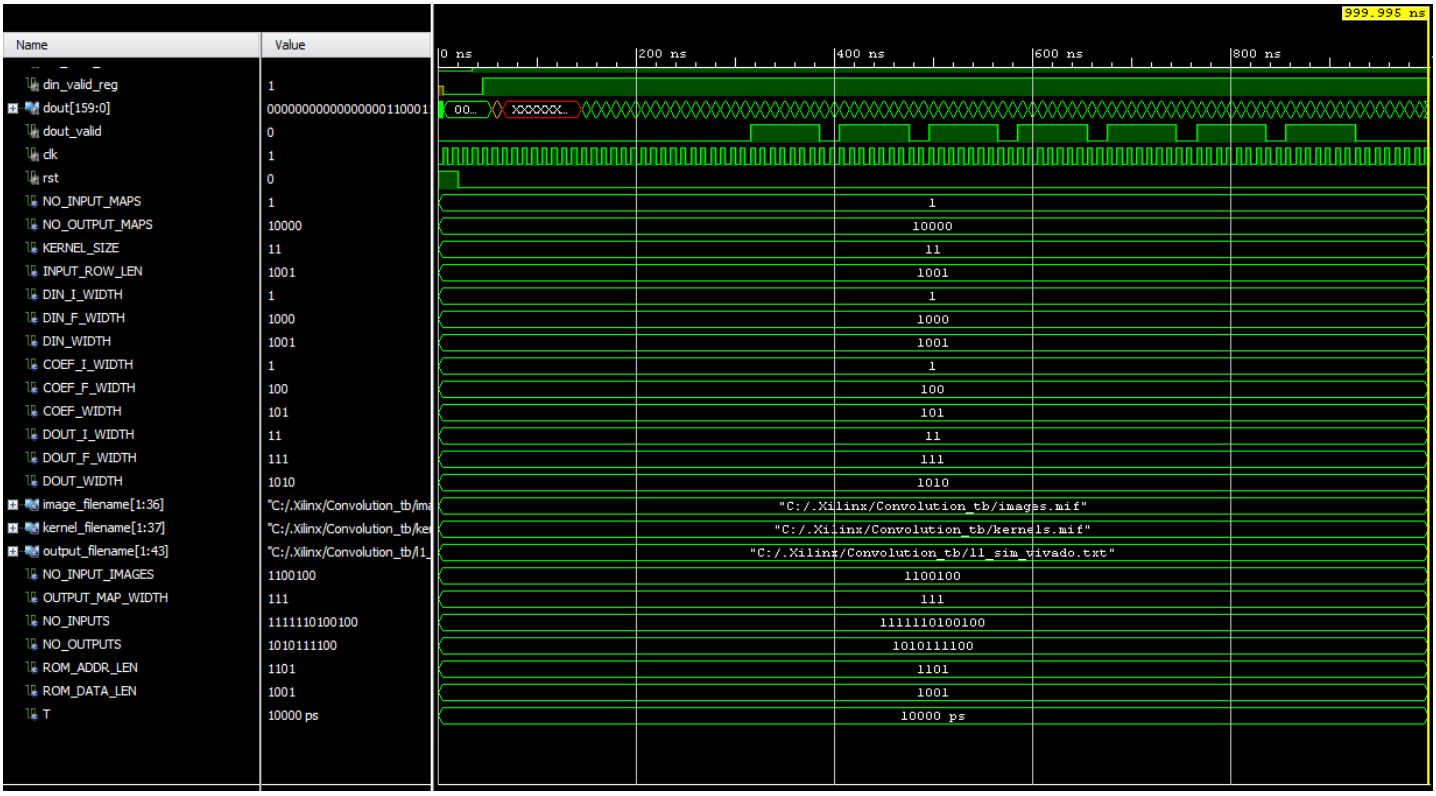


Figure 4.17: **Test bench Conv_Layer**

7.2.2 Maxpooling :

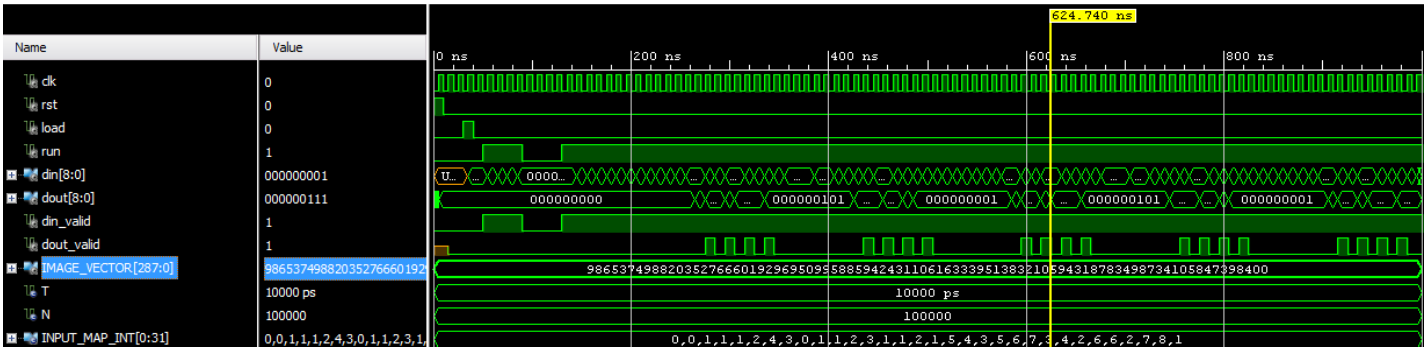


Figure 4.18: **Test bench Maxpooling**

7.2.3 Fc_layer :

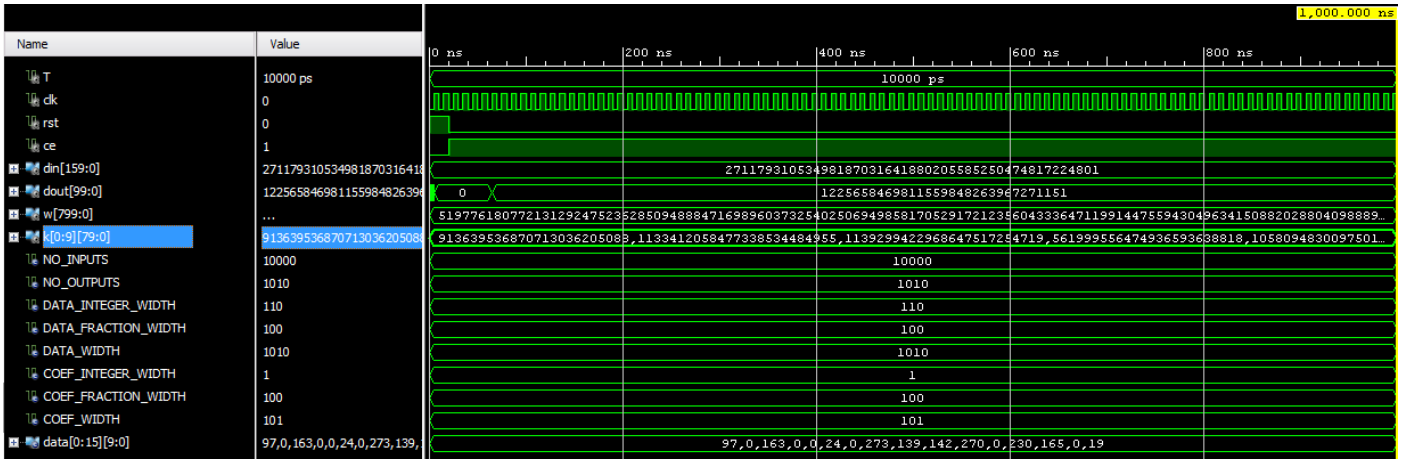


Figure 4.19: **Test Bench Fc_Layer**

8. Discussion

Selon la synthèse et l'occupation des ressources nous faisons les remarques suivantes :

Points positifs :

- Le Fully connected layer prend le plus de ressources (LUTs) par rapport aux autres modules de convolution et maxpooling, ceci est expliqué par l'utilisation des DSPs dans la convolution qui remplacent de façon efficace les LUTs.
- L'FPGA utilisé a suffisamment de ressources pour implémenter le model CNN adopté.
- Il est possible de bénéficier du parallélisme ainsi que du pipeline grâce à l'utilisation des arbres d'additionneur (adder trees) ce qui diminue le nombre de ressources occupés sans altérer les résultats.
- La fréquence de fonctionnement du modèle CNN implémenté est de l'ordre de 445 MHz.

Points négatifs :

- L'utilisation des DSP a dépassé les capacités de l'FPGA utilisé pour l'implémentation, avec 8848 DSP alors qu'il dispose de DSP uniquement.
- Les paramètres du filtre doivent être stocké dans des mémoires externes dans l'FPGA or ceci va augmenter le prix à payer.

- Nous avons manqué la réalisation et le test réel sur FPGA par le manque de carte dédiée spécialement au deep learning (cartes récentes comme le virtex7, Zynq et plus)

Dans le futur, nous prévoyons faire un test réel avec un exemple d'application sur une plateforme FPGA adaptée à ce type de traitement, une fois le travail validé, nous faisons une comparaison avec les travaux déjà réalisés que ce soit par rapport à la fréquence de fonctionnement ou à surtout l'efficacité du model CNN adopté par rapport à ce qui existe. Nous ne manquons pas de réaliser une comparaison entre une exécution sur FPGA et sur un GPU afin de trouver une solution rapide et efficace et remplacer ces GPUs qui coûtent des fortunes. Enfin le deep learning notamment le CNN reste des moyens les plus utilisés dans le domaine de l'imagerie de façon générale, à savoir pour identifier les objets, et faire leur classification, ce travail nous a permis de bien comprendre le déroulement de cette technologie en tentant de réaliser une implémentation matérielle afin d'accélérer le traitement à l'aide de l'FPGA, ce travail a été bénéfique et nous a permis d'apprendre à mieux gérer l'outil vivado pour la synthèse et la simulation, ainsi d'approfondir nos informations par rapport au langage de programmation VHDL et découvrir ses avantages.

9. Conclusion générale

Le travail décrit dans ce mémoire soulève le problème d'abondances de données et l'énorme quantité d'informations transmises et reçues afin de réaliser une architecture à base du deep learning notamment le CNN, ce problème peut être résolu en se dirigeant vers l'implémentation matérielle sous FPGA, en effet, L'FPGA présente l'avantage d'accélérer les calculs à travers plusieurs solutions, la première solution est de profiter du parallélisme et du pipeline dans l'exécution des opérations de calcul, et c'est cette solution qui a été adopté dans ce mémoire. En effet, dans ce dernier chapitre nous avons défini les différents modules du CNN implémentés, nous avons détaillé chaque bloc de chaque module ainsi que son fonctionnement, nous avons donné un schéma RTL à partir du logiciel ISE vivado (2016), une synthèse des ressources FPGA requis et des simulations par des test bench afin d'appuyer les résultats.

Le model CNN adopté est de type LeNet, les modules implémentés ont été adaptés selon des paramètres définis, ce model CNN est constitué de 3 couches de convolutions, suivi d'une opération de Relu après chacune de ces couches, puis une couche de Fully connected. Ce travail a été fait selon plusieurs étapes allant de l'étude des réseaux de neurones convolutifs, puis définir le moyen d'implémentation matérielle qui est l'FPGA. Quelques objectifs ont été atteints :

- Comprendre le flux de conception d'un projet FPGA sous vivado (2016),
- Définir les équations et algorithmes utilisés pour l'implémentation du model CNN en VHDL,
- Implémentation RTL de plusieurs modules du CNN à savoir :
    - Convolution
    - Maxpooling
    - Relu
    - FC layer

L'objectif principale de ce travail est de crée un environnement qui pourrait être utilisé dans de futures recherches basée sur le CNN, l'étape prochaine serait avant tout de valider cette approche au niveau de l'FPGA par un test en temps réel. Il est possible de faire des améliorations de l'architecture proposée en rajoutant plus de parallélisme et de pipeline dans la mesure où ça ne dépasse pas les ressources de l'FPGA utilisés.

[1] https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks

[2] https://www.supinfo.com/articles/single/6041-machine-learning-introduction

[3] https://fr.wikipedia.org/wiki/Apprentissage_automatique

[4]Deep Learning. Ivan Vasilev, Daniel Slater, GianmarioSpacagna, Peter Roelants, Valentino Zocca. 2019

[5]https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

[6]https://penseeartificielle.fr/focus-reseau-neurones-convolutifs

[7]https://fr.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[8]https://www.supinfo.com/articles/single/7923-deep-learning-fonctions-activation

[9] Lionel Pibre, Marc Chaumont, Dino Ienco, Jérôme Pasquet. Étude des réseaux de neurones sur la stéganalyse. CORESA: COmpression et REprésentation des Signaux Audiovisuels, May 2016, Nancy, France. ffhal-01374095f

[10]https://towardsdatascience.com/review-of-lenet-5-how-to-design-the-architecture-of-cnn-8ee92ff760ac

[11]https://www.researchgate.net/figure/Architecture-of-LeNet-2-AlexNet-2012-In-2012-Alex-Krizhevesky-and-others-proposed-a_fig4_323570864

[12]https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[13]https://fr.qwe.wiki/wiki/Logic_block

[14]https://www.xilinx.com/support/documentation/

[15]https://fr.wikipedia.org/wiki/Signal_d%27horloge

[16]https://www.pdx.edu/nanogroup/sites/www.pdx.edu.nanogroup/files/FPGA-architecture.pdf

[17]https://zone.ni.com/reference/en-XX/help/371599P-01/lvfpgaconcepts/dsp48e_top/

[18] https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

Conv_layer + Relu

```
--------- Lecture des filtres et signaux nécessaires à la convolution -----
architecture Structural of conv_layer is

    constant kernels_filename : string :=
"C:/.Xilinx/Convolution/l1_conv_kernels.mif";

  signal w : std_logic_vector(L1_NO_INPUT_MAPS * L1_NO_OUTPUT_MAPS *
(L1_KERNEL_SIZE**2) * L1_COEF_WIDTH - 1 downto 0);
    signal k : L1_KERNEL_MAP_T;

    signal dout_conv : std_logic_vector(dout'range);

begin

    k <= Init_L1_Kernel(kernels_filename);
    gen_kernel_map : for I in 0 to L1_NO_INPUT_MAPS * L1_NO_OUTPUT_MAPS - 1
generate
        w((I+1) * (L1_KERNEL_SIZE**2) * L1_COEF_WIDTH - 1 downto I *
(L1_KERNEL_SIZE**2) * L1_COEF_WIDTH) <= k(I);
    end generate gen_kernel_map;

--------------- L'activation ReLu -----------------------

    activation_function_1: for I in 0 to L1_NO_OUTPUT_MAPS - 1 generate
        relu_inst : relu
            generic map (
                WIDTH => L1_RESULT_WIDTH
            )
            port map (
                din => dout_conv((I+1) * L1_RESULT_WIDTH - 1 downto I *
L1_RESULT_WIDTH),
                dout => dout((I+1) * L1_RESULT_WIDTH - 1 downto I *
L1_RESULT_WIDTH)
            );
    end generate activation_function_1;

end Structural;

------génération des filtres kernel pour les 3 couches de convolution----

    k1 <= Init_L1_Kernel("C:/.Xilinx/TOP_CNN_V2/l1_conv_kernels.mif");
    gen_l1_kernel_map : for I in 0 to L1_NO_INPUT_MAPS * L1_NO_OUTPUT_MAPS
- 1 generate
        w1((I+1) * (L1_KERNEL_SIZE**2) * L1_COEF_WIDTH - 1 downto I *
(L1_KERNEL_SIZE**2) * L1_COEF_WIDTH) <= k1(I);
    end generate gen_l1_kernel_map;

    k2 <= Init_L2_Kernel("C:/.Xilinx/TOP_CNN_V2/l2_conv_kernels.mif");
    gen_l2_kernel_map : for I in 0 to L2_NO_INPUT_MAPS * L2_NO_OUTPUT_MAPS
- 1 generate
        w2((I+1) * (L2_KERNEL_SIZE**2) * L2_COEF_WIDTH - 1 downto I *
(L2_KERNEL_SIZE**2) * L2_COEF_WIDTH) <= k2(I);
    end generate gen_l2_kernel_map;

    k3 <= Init_L3_Kernel("C:/.Xilinx/TOP_CNN_V2/l3_conv_kernels.mif");
    gen_l3_kernel_map : for I in 0 to L3_NO_INPUT_MAPS * L3_NO_OUTPUT_MAPS
- 1 generate
```

```
        w3((I+1) * (L3_KERNEL_SIZE**2) * L3_COEF_WIDTH - 1 downto I *
(L3_KERNEL_SIZE**2) * L3_COEF_WIDTH) <= k3(I);
    end generate gen_l3_kernel_map;

    k4 <= Init_L4_Kernel("C:/.Xilinx/TOP_CNN_V2/l4_fc_kernels.mif");
    gen_l4_kernel_map : for I in 0 to L4_NO_OUTPUTS - 1 generate
        w4((I+1) * L4_NO_INPUTS * L4_COEF_WIDTH - 1 downto I * L4_NO_INPUTS
* L4_COEF_WIDTH) <= k4(I);
    end generate gen_l4_kernel_map;
```

CONV_2D :

```
begin

--------- génération du SE_chain -----------------

    se_chain_inst : se_chain
        generic map (
            KERNEL_SIZE => KERNEL_SIZE,
            DATA_WIDTH => DATA_WIDTH,
            DATA_FRAC_LEN => DATA_FRAC_LEN,
            COEF_WIDTH => COEF_WIDTH,
            COEF_FRAC_LEN => COEF_FRAC_LEN,
            RESULT_WIDTH => RESULT_WIDTH,
            RESULT_FRAC_LEN => RESULT_FRAC_LEN
        )
        port map (
            din => din,
            w => w,
            dp => dp_from_se,
            clk => clk,
            ce => ce,
            coef_load => coef_load,
            rst => rst
        );

----------- génération du switching bloc -------------------
    switching_block_inst : switching_block
        generic map (
            INPUT_ROW_LENGTH => INPUT_ROW_LENGTH,
            KERNEL_SIZE => KERNEL_SIZE,
            RESULT_WIDTH => RESULT_WIDTH
        )
        port map (
            dp_in => dp_from_se,
            add_out => from_switch
        );

------------ génération des additionneurs --------------------
    gen_adders: for I in (KERNEL_SIZE - 2) downto 0 generate
        adder_i : adder
            generic map (
                DATA_WIDTH => RESULT_WIDTH
            )
            port map (
                din_a => from_switch(RESULT_WIDTH * (I+1) - 1 downto
RESULT_WIDTH * I),
                din_b => from_buffer(RESULT_WIDTH * (I+1) - 1 downto
RESULT_WIDTH * I),
```

```
                dout => from_adder(RESULT_WIDTH * (I+1) - 1 downto
RESULT_WIDTH * I)
            );
    end generate;

    dout_next <= from_adder(RESULT_WIDTH - 1 downto 0);

----------------- génération des registres ------------
      registers: process(clk) is
      begin
            if (rising_edge(clk)) then
                  if (ce = '1') then
                        dout_reg <= dout_next;
                  end if;
            end if;
      end process registers;
    dout <= dout_reg;
end rtl;
```

Fully connected :

```
begin

output_neurons_gen: for J in 0 to NO_OUTPUTS - 1 generate
      mult_gen : for I in 0 to NO_INPUTS - 1 generate
            data_from_mults_to_adders(J * NO_INPUTS * MULT_RES_WIDTH +
(I+1) * MULT_RES_WIDTH - 1 downto J * NO_INPUTS * MULT_RES_WIDTH + I *
MULT_RES_WIDTH) <= std_logic_vector(
              signed(din((I+1) * DATA_WIDTH - 1 downto I * DATA_WIDTH)) *
signed(w(J * NO_INPUTS * COEF_WIDTH + (I+1) * COEF_WIDTH - 1 downto J *
NO_INPUTS * COEF_WIDTH + I * COEF_WIDTH))
            );
      end generate;

-------------- generation de l'arbre d'additionneur (adder tree) --------

adder_tree_inst : adder_tree
      generic map (
            NO_INPUTS => NO_INPUTS,
            DATA_WIDTH => MULT_RES_WIDTH
      )
      port map (
            din => data_from_mults_to_adders((J+1) * NO_INPUTS *
MULT_RES_WIDTH - 1 downto J * NO_INPUTS * MULT_RES_WIDTH),
            dout => res_from_adders((J+1) * MULT_RES_WIDTH - 1 downto J *
MULT_RES_WIDTH),
            clk => clk,
            ce => ce,
            rst => rst
      );

dout((J+1) * RESULT_WIDTH - 1 downto J * RESULT_WIDTH) <=res_from_adders((J
* MULT_RES_WIDTH) + MULT_RES_FRACTION_WIDTH + RESULT_INTEGER_WIDTH - 1
downto (J * MULT_RES_WIDTH) + MULT_RES_FRACTION_WIDTH -
RESULT_FRACTION_WIDTH);

      end generate;

end RTL;
```