الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DR. YAHIA FARES DE MEDEA

FACULTE DE LA TECHNOLOGIE

Département du Génie Électrique



Projet de Fin d'Etudes

Présenté pour l'obtention du Diplôme
De Master en Génie Électrique

Spécialité : *Systèmes de Télécommunications*

THEME

Implémentation d'un réseau de neurones artificiels sur une plateforme FPGA

Proposé et dirigé par :

M. BNGHARBIA Billel

Préparés et présentés par :

KERMEZLI Abd el fettah

BELZIANE Fayçal

Année universitaire : 2017\2018

Dédicaces

بسم الله و الصلاة و السلام على اشرف خلق الله سيدنا المصطفى صلى الله عليه و سلم القائل(اطلبوا العلم من المهد إلى اللحد),

بعد سنين من الجهد و الاجتهاد في طلب العلم تمكنا و بتوفيق من الله عز و جل و سهر ورعاية الوالدان من الوصول إلى مستوى الثانية ماستر, و عمل هاته المذكرة التي نعتبرها ثمرت جهدنا و تعبنا

و بدورنا نهدي هاته المذكرة إلى كل عائلة قرمزلي و بلزيان نخص بالذكر منهم الآباء و الأمهات الذين كان لهم دور كبير في مسارنا الدراسي

و في الأخير نطلب من الله عز و جل أن يوفقنا لمزيد من طلب العلم و ان ينفعنا بما علمنا , لنا و لجميع الطلبة

Remerciement :

En premier lieu, nous tenons d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir

Ce modeste travail.

En second lieu, nous tenons à remercier et à exprimer nos gratitudes à notre encadreur Mr,BNGHARBIA Billel

Un grand merci aux membres de nos familles pour leur amour et leur soutien, en particulier ma petite BELZIANE Hebbat ellah dont je suis très fière et ma chère épouse.

ملخص :

على مدى السنوات العشر الماضية , في مجال الانظمة الرقمية اكتشف نوع جديد من الدارات المنطقية القابلة للبرمجة و القادرة على تجسيد اي دارة رقمية, هذه الانظمة تجمع ما بين قوة الاجهزة و سهولة البرمجة.

خلال هذه الدراسة , الهدف الاساسي هو تطبيق خاصية الشبكة العصبونية الاصطناعية على دارة قابلة للبرمجة من نوع PYNQ-Z1 لإنجاز مشروع يهدف للتعرف على اشارات المرور , و لبرمجت هاته الدارة استعملنا برنامج بايثون

كلمات مفتاحية : الشبكة العصبونية الاصطناعية , الدارات المنطقية القابلة للبرمجة , بايثون,

Résumé :

Depuis une dizaine d'années, il existe des systèmes numériques basées sur de nouveau type de circuit appelés FPGA capable d'incarner un circuit numérique quelconque, ces systèmes combinent la puissance du matériel avec la souplesse d'utilisation du logiciel. Au cours de cette étude, l'objectif principal est d'appliquer la propriété de réseau neural artificiel sur un circuit programmable PYNQ-Z1 pour la réalisation d'un projet visant à identifier des panneaux de signalisation dans une image, en utilisant la programmation en PYTHON.

Mots clés : FPGA, PYNQ, PYTHON, Réseau de neurones artificiels,

Abstract

For ten years, there were digital systems based on a new type of circuit called FPGA, which is able to implement any digital circuit ,these systems combine between the power of the materiel and the flexibility of the software's use. Through this study. The main purpose is to apply feature of artificial neural network to a programmable circuit PYNQ-Z1, for the realization of a project to identify road signs in an image using PYTHON programming.

Key words: FPGA, PYNQ, PYTHON, Artificial Neural Network.

Table des matières

DEDICACE……I
REMERCIMENT……II
RESUME ……III
TABLE DES MATIERES……IV
LISTE DES FIGURES ……V
LISTE DES TABLEAUX ……VI
LISTE DES ABREVIATIONS……VII
INTRODUCTION GENERAL……1
CHAPITRE I : les circuits programmables
I.1 Introduction ……2
I.2 Les circuits programmables ……2
I.2.1 Structure de base d'un PLD ……3
I.3 FPGAs ……4
I.3.1 Configuration et reconfiguration des FPGA ……8
I.3.2 Fiabilité ……9
I.3.3 Le choix des FPGAs ……9
I.4 La plateforme PYNQ-Z1 ……10
I.4.1 Présentation de la carte PYNQ-Z1 ……10
I.4.2 Caractéristiques ……13
I.4.3 Configuration de ZYNQB ……14
I.5 Conclusion……16
CHAPITRE II : les réseaux neurone artificiels
II.1. Introduction……17
II 2. Historique……17
II.3. Le neurone……17
II.3. 1. Structure de neurone biologique ……17
II.3.2 Structure de neurone artificiel ……18
II.3.3 Les connexions entre les neurones ……19
II.3.4 Structure d'interconnexion ……19
II.3.5 Différentes architectures neuronales ……21
II.3.6 Apprentissage d'un réseau de neurone ……22
II.3.7 Interprétation mathématique ……23
II.3.8 Réseaux neuronaux bayésiens ……26
II.4 Conclusion ……30
CHAPITRE III : Implantation des réseaux neurone artificiel sur la carte PYNQ
III.1. Introduction……32
III.2 Jupyter Notebook ……32
III.2.1 Composants ……32
III.2.2 Jupyter Notebook en PYNQ ……33
III.3 PYTHON ……33
III.4 Configuration de la carte PYNQ-Z1 ……36
III.5 BNN sur Pynq ……38
III.5.1 Discussion……44
III.6. Conclusion ……44
Conclusion générale ……46

LISTE DES FIGURES

Figure I.1 Illustration de l'architecture PAL…….…….4

Figure I.2 Structure interne de l'FPGA basée sur l'architecture de Xilinx…….……...6

Figure I.3 *PYNQ-Z1*……..*10*

Figure I.4 Architecture de Zynq APSoC ……..…..……..12

Figure II.1 Un neurone avec son arborisation dendritique …….18

Figure II.2 Structure de Neurone artificiel…..…….18

Figure II.3 Connexion simple entre deux neurones ……19

Figure II.4 Définition des couches d'un réseau multicouch……..……*20*

Figure II.5 Réseau à connexions locales…… ……20

Figure II.6 Réseau à connexions récurrentes……...21

Figure II.7 Réseau à connexions complète ……21

Figure II.8 Différente architectures neuronales……...22

Figure II.9 La composition d'un neurone artificiel……..22

Figure II.10 Étapes de construction d'un réseau bayésien ……....27

Figure II.11 Réseau bayésien naïf……..28

Figure II.12 Réseau bayésien naïf augmenté……...29

Figure III.1 Interprétation du byte code compilé……..…… 33

Figure III.2 les différentes étapes de Configuration de la carte PYNQ-Z1 ……...37

Figure III.3 l'interface jupyter…..……...38

Figure III.4 reconnaissance d'image avec un neurone binarisé …….…….39

Figure III.5 les images que nous voulons classer……40

Figure III.6 Identifier les images que nous voulons classer……41

Figure III.7 La première image d'entrée *Exemple 1*……..……..41

Figure III.8 Chercher le panneau dans l'image *Exemple 1*……..……42

Figure III.9 Le résultat final *Exemple 1*……...……42

Figure III.10 La première image d'entrée *Exemple 2*……..……43

Figure III.11 Chercher le panneau dans l'image *Exemple 2*……..…..44

Figure III.12 Le résultat final *Exemple 2*……..…44

LISTE DES TABLEAUX

Tableau I.1 L'évolution des PLD…….3

Tableau II.1 Transition du neurone biologique au neurone artificiel ……...19

Tableaux II.2 fonctions d'activation ….……25

LISTE DES ABREVIATIONS

ADALINE : ADAptive LINear Elément ou Adaptive Linear Neuron (Réseau adaptative linéaire)
AMBA: Advanced Micro controller Bus Architecture
AMD: Advanced Micro Device
ANN : Artificiel neural networks (réseaux de neurones artificiels)
API: Application programming interface
ASIC: Application Specific Integrated Circuits
CLB: Configurable Logic Bloc
CMOS: Complementary metal oxyde semiconductor
CPLD: Complex programmable logic Device
CPU: Central processing unit
DDR: Double data rate
DMA: Direct memory access
DRAM: dynamique Random Access Memory
DSP: Digital signal processor
EEPROM: Electrically Erasable Programmable Read Only Memory
FPGA: Field Programmable Gate Array
EPROM: Erasable Programmable Read Only Memory
GAL: Generic array logic
HDMI: High definition multimedia interface
ISP: In Sito Programming
JTAG: Joint test action group
Ko : Kilo octets
LED: Light Emitting Diode
LUT: Lock Up Table
MMCM: Mixed-mode clock manager
Mo: méga octets
OTP: One Time Programmable
PAYTHON: Language de programmation
PDM : Program Decision Memorandum
PHY : prend en charge l'hôte uniquement
PL: Programmable logic
PLA: Programmable Logic Array
PLD: Programmable Logic Device
RAM: Read Only Memory
ROM: Read Only Memory
SD : Secure Digital
SDIO: Secure digital input output
SRAM: Static Random Access Memory
SoPC: System on programmable chip
UART: Universal asynchrony receiver transmitter
USB: Universal serial bus
V: Volte
XADC: Xilinx Analog to Digital Converter

INTRODUCTION GENERALE

Pendant les dix dernières années, plusieurs architectures combinant des processeurs et/ou des circuits reconfigurables (FPGA) ont été proposées pour accélérer l'exécution d'applications de plus en plus complexes. Les systèmes dédiés au traitement du signal et de l'image utilisent actuellement soit des processeurs à usage général ou dédiés, soit des solutions câblées intégrant des circuits spécifiques de type ASIC ou la combinaison des deux. Cependant, la conjugaison de la complexité croissante des algorithmes et d'un grand volume de données à traiter, le tout avec des contraintes temps réel fortes, réclame des performances que les processeurs ne peuvent fournir. La mise en parallèle de processeurs peut apporter la rapidité et la souplesse de programmation, mais au détriment du coût et de la complexité de mise en œuvre les architectures reconfigurables représentent une réponse intermédiaire appropriée, Elles utilisent des composants logiques reconfigurables qui permettent à l'utilisateur de modifier l'architecture après fabrication de façon logicielle, contrairement aux ASICs dont les algorithmes sont câblés dans le silicium.

Le terme reconfiguration désigne l'opération qui vise à implanter dans le ou les composants une nouvelle fonctionnalité sans modification de l'architecture matérielle du système.

Dans cette optique, les FPGAs SRAM, par leur reconfigurabilité rapide, insitu et illimitée, offrent des perspectives riches dans le domaine du traitement d'image La tâche qui nous incombe, entre dans ce contexte et consiste à exploiter les performances des circuits FPGAs pour montrer l'efficacité d'implémenter des applications liées au traitement d'image.

A travers ce travail, nous avons exploité les performances des circuits FPGAs pour montrer l'efficacité d'y implémenter des applications liées au traitement d'image et cela en grande parti grâce la tendance au parallélisme dans le calcul permis par les FPGAs.

Dans ce mémoire, au chapitre 1 nous avons commencé par donner des notions sur les FPGA et leurs performances, puis nous décrirons, de même, le matériel mis à disposition. La carte PYNQ-Z1 qui est la plate-forme matérielle du framework open-source PYNQ. Dans le chapitre 2 nous avons introduire les réseaux de neurones, en particulier les réseaux Bayésiens exploités dans notre travail car ils ont plusieurs avantages par rapport à d'autres techniques dans le domaine du traitement d'image.

Dans le chapitre 3, nous avons fait embarquer le programme de BNN dans la carte **PYNQ-Z1**, puis nous avons montré comment faire l'implémentation en utilisant un langage de description matérielle **PYTHON** et cela pour un test de reconnaissance des panneaux de signalisation dans une image.

I.1 Introduction :

Les dispositifs logiques programmables (PLD) ont été introduits au milieu des années 1970. L'idée était de construire des circuits logiques combinatoires programmables. Cependant, contrairement aux microprocesseurs, qui peuvent exécuter un programme mais possèdent un matériel fixe, la programmabilité des PLD était prévue au niveau matériel. En d'autres termes, un PLD est une puce à usage général dont le matériel peut être reconfiguré pour répondre à des spécifications particulières. Dans le présent chapitre nous avons introduire une des classes les plus performant des circuits logiques programmable qui sont les FPGA. Ces circuits qui sont largement utilisé dans la conception des systèmes embarquée.

I.2 Les circuits programmables

Pour diminuer les coûts de fabrication, de développement et de maintenance, les fabricants de circuits intégrés ont donné naissance aux Circuits Logique Programmable ou encore P.L.D (Programmable Logic Device). Ces circuits sont capables de réaliser plusieurs fonctions logiques dans un seul circuit. Si ces fonctions étaient réalisées à base de circuits logiques classique, il faudrait plusieurs circuits [1].

Les premiers PLD ont été appelés PAL (Programmable Array Logic) ou PLA (Programmable Logic Array), selon le schéma de programmation. Ils utilisaient uniquement des portes logiques (pas de bascules), permettant uniquement la mise en œuvre de circuits combinatoires. Pour contourner ce problème, des PLD ont été lancés peu de temps après, qui comprenaient une bascule à chaque sortie du circuit. Avec eux, des fonctions séquentielles simples pourraient ensuite être mises en œuvre [1].

Au début des années 1980, des circuits logiques supplémentaires ont été ajoutés à chaque sortie PLD. La nouvelle cellule de sortie, appelée Macrocell, contenait (outre la bascule) des portes logiques et des multiplexeurs. De plus, la cellule elle-même était programmable, permettant plusieurs modes de fonctionnement. De plus, il fournissait un signal de « retour » (retour) de la sortie du circuit à la matrice programmable, ce qui donnait au PLD une plus grande flexibilité. Cette nouvelle structure PLD a été appelée PAL générique (GAL) [1].

Plus tard, plusieurs dispositifs GAL ont été fabriqués sur la même puce, en utilisant un schéma de routage plus sophistiqué, une technologie silicium plus avancée et plusieurs fonctionnalités supplémentaires (comme le support JTAG et l'interface avec plusieurs standards logiques). Cette approche est devenue connue sous le nom de CPLD (Complexe

PLD). Les CPLD sont actuellement très populaires en raison de leur forte densité, de leur haute performance et de leur faible coût.

Enfin, au milieu des années 1980, des FPGA (Field Programmable Gate Arrays) ont été introduits. Les FPGA diffèrent des CPLD en termes d'architecture, de technologie, de fonctionnalités intégrées et de coût. Ils visent principalement la mise en œuvre de circuits de grande taille et haute performance.

Un résumé de l'évolution des PLD est présenté dans le tableau ci-dessous.

PLDs	Simple PLD (SPLD)	PAL PLA Registered PAL/PLA GAL
	Complex PLD (CPLD)	
	FPGA	

Tableau I.1 L'évolution des PLD.

I.2.1 Structure de base d'un PLD

La réalisation des fonctions combinatoires pour des composants du type PLD est obtenue en deux étapes : la réalisation des termes produits puis leur sommation. Ce qui conduit à une structure matricielle telle que, la première matrice permet de créer des min termes et la deuxième réalise les fonctions. En effet la pluparts des circuits PLD suivent une structure possédant [2]:

a) Un ensemble d'opérateurs ''ET'' (AND) sur lesquels viennent se connecter les variables d'entrée et leurs compléments.

b) Un ensemble d'opérateurs ''OU'' (OR) sur lesquels les sorties des opérateurs ''ET'' sont connectés.

L'architecture de base est illustrée symboliquement dans la (*Figure I.1)*

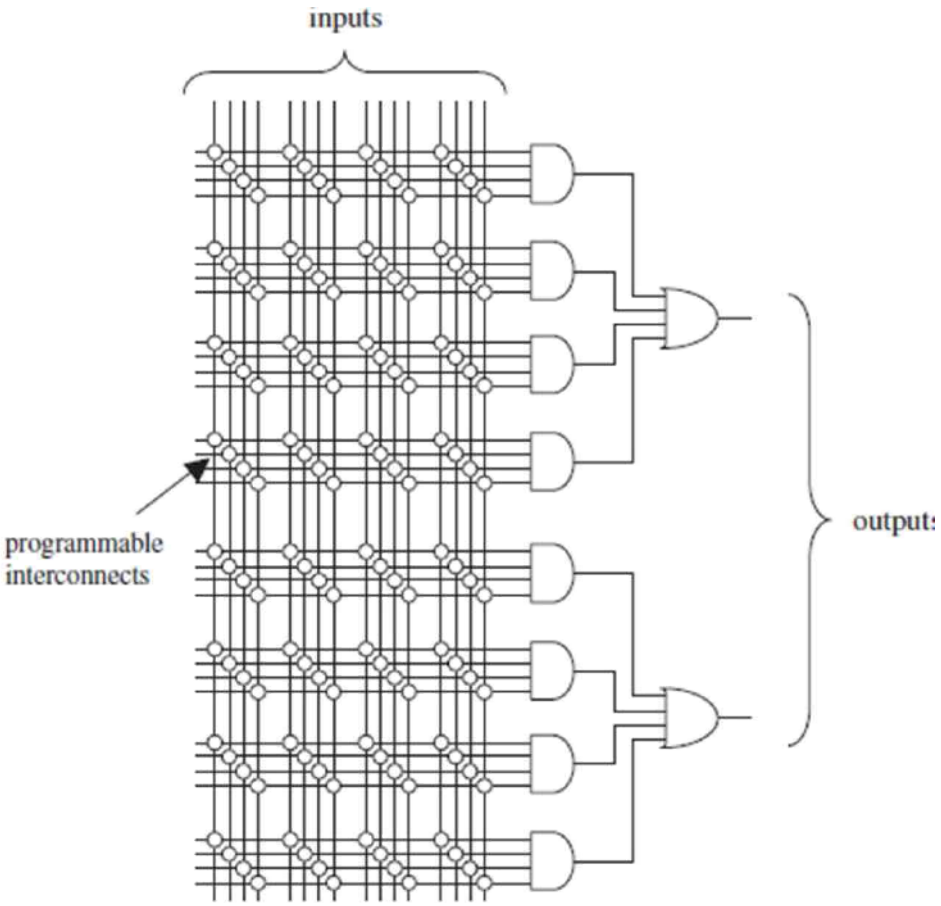


Figure I.1 Illustration de l'architecture PAL.

I.3 FPGAs

Les circuits FPGAs ont été introduits comme une simple technologie des circuits logiques. Ils ont connu une croissance rapide et sont devenus un moyen d'implémentation populaire pour les circuits numériques. L'évolution de la technologie a considérablement amélioré la capacité logique des FPGAs et les a rendus comme une alternative d'implémentation pour des conceptions plus grandes et complexes. En outre, la nature programmable de leurs ressources logiques et de routage a un grand effet sur la taille de la surface, la vitesse et la consommation d'énergie du circuit. Les FPGAs peuvent être reprogrammés électroniquement selon les exigences d'application ou de fonctionnalité souhaitée pour devenir pratiquement n'importe quel type de circuit ou système numérique. Cette fonctionnalité distingue les FPGAs des ASICs, qui eux sont fabriqués sur mesure pour des tâches de conception spécifique. Cette approche permet de corriger les erreurs de conceptions détectées au niveau final du développement, éventuellement par une simple reprogrammation de l'FPGA, permettant ainsi de modifier l'inter-connectivité des composants selon les besoins. Bien que cette approche introduise des retards supplémentaires en raison de l'interconnexion programmable, elle évite

une reconception coûteuse [3].

La nature souple des FPGAs les rend considérablement plus grands, plus lents et plus consommateurs d'énergie que leurs homologues ASICs. Ces inconvénients résultent en grande partie de l'interconnexion de routage programmable qui comprend près de 90% de la superficie totale des FPGAs. Cependant, malgré ces inconvénients, les FPGAs offrent une bonne solution pour l'implémentation des systèmes numériques en raison de leur faible coût et du délai de mise sur le marché plus rapide par rapport aux ASIC, qui nécessitent en générale beaucoup de temps et d'argent pour obtenir un premier prototype [3].

Les FPGAs comprennent dans leur structure de base, les composants suivants (*Figure I.2*):

1. Blocs logiques configurables (CLB : Configurable Logic Block) qui implémentent des fonctions logiques à travers des LUTs (LookUp Table). Ces blocs sont placés dans une grille bidimensionnelle et sont interconnectés par des réseaux d'interconnexion programmables.
2. Les blocs d'Entrées/Sorties qui sont placés sur la périphérie de la grille et sont connectés à des blocs logiques via le réseau d'interconnexion programmable.
3. Le réseau d'interconnexions configurables entre ces différents blocs.

L'architecture interne des FPGAs actuels comporte, dans sa structure, des blocs additionnels tels que les registres pour stocker les résultats, des blocs mémoires RAM à haute vitesse qui permettent d'implanter des mémoires adressables, des ressources spécifiques comme les blocs génériques DSP dédiés à la réalisation facile de nombreuses opérations de traitement, ainsi que des processeurs hardware intégrés [3].

Il existe un certain nombre de technologies de configuration par la programmation qui ont été utilisées pour les architectures reconfigurables comme les FPGAs. Ces technologies ont des caractéristiques différentes qui ont à leur tour un effet significatif sur l'architecture reconfigurable. Certaines des technologies bien connues incluent la mémoire statique, flash et anti-fusible.

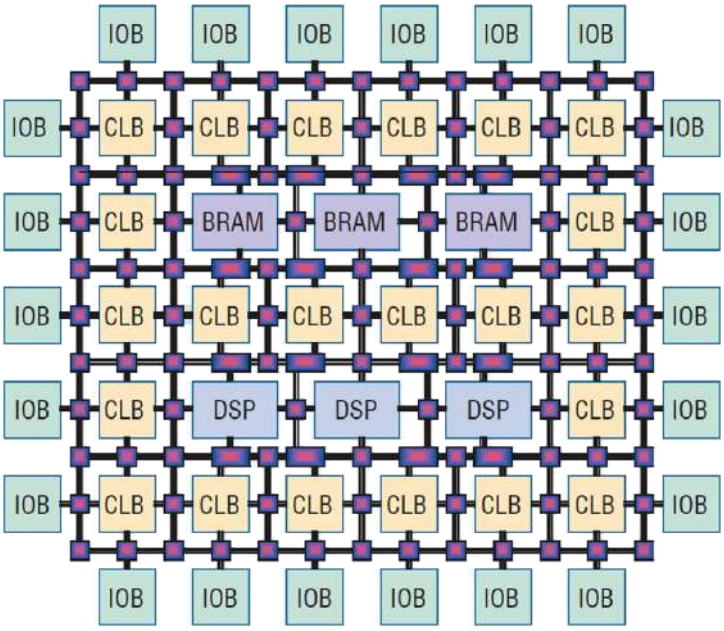


Figure I.2 Structure interne de l'FPGA basée sur l'architecture de Xilinx

Technologie de programmation SRAM

Il existe deux versions principales de périphériques RAM à semi-conducteurs : RAM dynamique (DRAM) et RAM statique (SRAM). Dans le cas des DRAM, chaque cellule est formée d'une paire de transistors-condensateurs. Le terme "dynamique" est utilisé car le condensateur perd sa charge dans le temps, de sorte que chaque cellule doit être périodiquement rechargée si elle doit conserver ses données. Cette opération, connue sous le nom de rafraîchissement, est un peu complexe et nécessite une quantité substantielle de circuits supplémentaires. Lorsque le coût de ce circuit de rafraîchissement est amorti sur des dizaines de millions de bits dans un dispositif de mémoire DRAM, cette approche devient très rentable. Cependant, la technologie DRAM présente peu d'intérêt pour la logique programmable.

Par analogie, le terme "statique" associé à la SRAM est utilisé, car une fois qu'une valeur a été chargée dans une cellule SRAM, elle reste inchangée à moins qu'elle ne soit modifiée spécifiquement ou jusqu'à ce que le circuit soit mis hors tension.

La technologie de programmation basée sur la mémoire statique (SRAM) est devenue l'approche dominante pour les FPGAs, en raison de sa ré-programmabilité et de l'utilisation d'une technologie CMOS standard. Ceci permet d'augmenter le taux d'intégration sur puce et donc une vitesse plus élevée et une consommation d'énergie dynamique plus faible avec une surface plus petite. Actuellement, la plupart des fabricants tels que Lattice et Xilinx adoptent

cette technologie dans leurs circuits. Ces derniers utilisent des cellules de mémoire statique qui sont réparties sur tout l'FPGA pour assurer la configurabilité. Les cellules SRAM sont principalement utilisées pour les buts suivants:

- Programmer le réseau d'interconnexions des FPGAs qui sont généralement pilotés par des multiplexeurs.
- Programmer des CLBs qui sont utilisés pour implémenter des fonctions logiques.

Il existe cependant un certain nombre d'inconvénients associés à la technologie de programmation SRAM. Par exemple, une cellule SRAM nécessite 6 transistors, ce qui rend l'utilisation de cette technologie coûteuse en termes de superficie par rapport à d'autres technologies de programmation. D'autres part, les cellules SRAM sont de nature volatile et des circuits externes sont nécessaires pour stocker en permanence les données de configuration. Ces circuits externes s'ajoutent au coût et à la surface de fonctionnement des FPGAs.

Technologie de programmation Flash

Une alternative à la technologie de programmation SRAM est l'utilisation de la technologie de programmation basée sur une mémoire flash ou EEPROM. Cette technologie offre plusieurs avantages. Par exemple, elle est de nature non volatile et est également plus efficace que la technologie de programmation SRAM. Cependant, elle a également ses propres inconvénients. Contrairement à la technologie basée sur la SRAM, les composants à mémoire flash ne peuvent pas être reconfigurés un grand nombre de fois. De plus, la technologie flash utilise une technologie CMOS non standard.

Technologie de programmation anti-fusible

La technologie de programmation anti-fusible est une alternative aux technologies SRAM et flash, dans laquelle chaque chemin configurable a une liaison associée appelée anti-fusible.

Un anti-fusible représente au début une colonne microscopique de silicium amorphe (non-cristallin) reliant deux pistes métalliques. Dans son état non programmé, le silicium amorphe agit comme un isolant avec une très haute résistance supérieure à un milliard d'ohms. Il peut être considéré comme un circuit ouvert, c'est l'état initial de l'FPGA avant qu'il ne soit programmé. Les anti-fusibles peuvent être programmés en appliquant des impulsions de tension et de courant relativement élevées à ses entrées, en convertissant le silicium amorphe isolant en silicium poly-cristallin conducteur.

Le principal avantage de la technologie de programmation anti-fusible est sa faible surface. En outre, cette technologie est non volatile par défaut. Il existe toutefois d'importants inconvénients associés à cette technologie de programmation. Par exemple, cette dernière n'utilise pas la technologie CMOS standard. En outre, les circuits basés sur cette technologie ne peuvent pas être reprogrammés.

1.3.1 Configuration et reconfiguration des FPGA

La configuration des FPGA peut prendre différentes formes, que ce soit pour configurer les ressources opératoires ou que ce soit pour configurer les réseaux de routage. Cependant, les FPGA de technologie anti-fusible ne sont configurables qu'une seule fois. En anglais, on parle de circuits OTP (One Time Programmable). Notons l'utilisation en anglais du mot « programmable » alors que les circuits FPGA ne sont pas programmables (ils n'exécutent pas un programme) mais ils sont configurables (la configuration mémorisée de tous les éléments fonctionnels et de routage du FPGA lui permet de réaliser une application). Les seuls FPGA reconfigurables (configurables plusieurs fois) sont les circuits de technologie FLASH (non volatile) et les circuits de technologie SRAM (volatile et qui nécessite la sauvegarde externe dans une mémoire non volatile de la configuration)

Dans ces deux derniers cas, la reconfiguration de l'architecture peut être mise en œuvre de plusieurs façons au cours de l'exécution de l'application. Elle peut intervenir une seule fois sans remise en cause durant l'exécution de l'application [4].

Nous parlerons dans ce cas de reconfiguration statique. Les processus de reconfiguration et d'exécution de l'application sont, dans ce cas, distincts et clairement séparés dans le temps. Une nouvelle reconfiguration sera effectuée dans deux cas. Tout d'abord elle peut être la conséquence d'une perte de la configuration pouvant être due pour certains dispositifs à un arrêt de la source d'alimentation (cas des FPGA de technologie SRAM sans sauvegarde externe de la configuration). Mais elle peut aussi faire suite à une modification de l'application par le concepteur si celle-ci est défaillante ou peut être améliorée (cas du prototypage) [4].

Cependant une étude approfondie de l'exécution d'une application peut éventuellement mettre en évidence que certaines parties configurées de l'architecture ne sont nécessaires qu'un temps court vis à vis de la durée de l'exécution, et peuvent par contre utiliser une place conséquente dans l'architecture. Le ratio entre le temps d'utilisation de la partie configurée et

la taille qu'elle occupe sur l'architecture peut être faible. D'où l'idée d'introduire un dynamisme temporel à la reconfiguration. Lorsqu'une partie de l'application est exécutée, et dans le cas où elle ne l'est plus dans un temps proche, on peut reconfigurer les éléments qui lui étaient dédiés afin de les utiliser pour une autre partie de l'application [5].

La reconfiguration dynamique permet donc d'optimiser la surface configurée dans le temps. Etant donné qu'il s'agit de modifier dans le temps uniquement une partie des éléments configurables, il faut nécessairement utiliser une architecture à reconfiguration partielle. Ici l'inconvénient vient qu'il est nécessaire de bien déterminer le partitionnement dans le temps de l'application afin de profiter de toute la surface du circuit. Il peut exister des problèmes de fragmentation comme sur les disques durs des ordinateurs. De plus, il faut correctement établir les communications entre les partitions [5].

I.3.2 Fiabilité

Tandis que les outils logiciels fournissent l'environnement de programmation, les circuits FPGA sont une véritable implémentation matérielle de l'exécution logicielle. Les systèmes basés processeur comprennent souvent plusieurs couches d'abstraction, pour aider à la planification des tâches et à la répartition des ressources entre les différents processus. La couche de driver contrôle les ressources matérielles et le système d'exploitation gère la mémoire et la bande passante du processeur. Sur chaque cœur de processeur, une seule instruction peut s'exécuter à la fois [6]

1.3.3 Le choix des FPGAs

Pour réussir une application à base d'un FPGA et afin d'obtenir un système plus performant, consommant un minimum de puissance, il est nécessaire de respecter un certain nombre de règles. En premier lieu il faut connaitre les caractéristiques qui doivent être dans le FPGAs cible pour assurer son adéquation avec les besoins du projet, puisque les fondeurs des FPGAs définissent ses propres structures des blocs logiques configurables, des cellules logiques ; de la logique fixe, comme des multiplicateurs ; des ressources de mémoire, comme des blocs de RAM intégrés. Et bien que souvent les FPGAs contiennent aussi d'autres composants, comme des micro-processeurs et des circuits spécifiques ainsi que la taille des circuits (nombres de blocs disponibles, nombre de port d'entrée sortie). Ceci est dû au fait que les FPGAs sont utilisée dans des applications variées et il n'y a pas un seul circuit pour toutes les applications [7].

I.4 La plateforme PYNQ-Z1

I.4.1 Présentation de la carte PYNQ-Z1

L'objectif principal de PYNQ, Python Productivity pour Zynq, est de permettre aux concepteurs de systèmes embarqués d'exploiter plus facilement les avantages uniques des APSoC dans leurs applications. Plus précisément, PYNQ permet aux architectes, ingénieurs et programmeurs qui conçoivent des systèmes embarqués d'utiliser les APSoC de Zynq, sans avoir à utiliser des outils de conception de type ASIC pour concevoir des circuits logiques programmables [8].

Au lieu de cela, l'APSoC est programmé en utilisant le langage Python, le code étant développé et testé directement sur le PYNQ-Z1. Les circuits logiques programmables sont importés en tant que bibliothèques matérielles et programmés via leurs API de la même manière que les bibliothèques logicielles soient importées et programmées [8].

La carte PYNQ-Z1 est la plate-forme matérielle du framework open-source PYNQ. Le logiciel fonctionnant sur les processeurs ARM A9 comprend:

- Un serveur Web hébergeant l'environnement de conception Jupyter Notebook
- Le noyau IPython et les paquets
- Linux
- Bibliothèque matérielle de base et API pour le FPGA

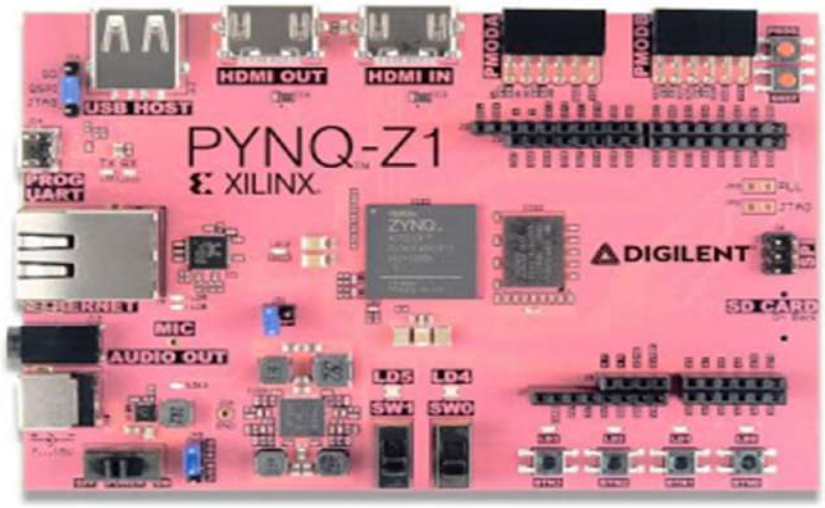


Figure I.3 PYNQ-Z1

PYNQ est destiné à être utilisé par un large éventail de concepteurs et de développeurs, compris:

- Les développeurs de logiciels qui veulent tirer parti des capacités de Zynq et du matériel programmable sans avoir à utiliser des outils de conception de type ASIC pour concevoir du matériel.
- Les architectes de système qui veulent une interface logicielle facile et un cadre pour leur conception de Zynq.
- Les concepteurs de matériel qui veulent que leurs conceptions soient utilisées par le plus large public possible.

Architecture Zynq APSoC

L'APSoC de Zynq est divisé en deux sous-systèmes distincts: le système de traitement (PS) et la logique programmable (PL). La (*Figure I.4)* montre une vue d'ensemble de l'architecture Zynq APSoC, Notez que le contrôleur PCIe Gen2 et les émetteurs-récepteurs multi-gigabits ne sont pas disponibles sur le périphérique Zynq 7020 [9].

Le PL est presque identique à un FPGA Artix Xilinx série 7, sauf qu'il contient plusieurs ports et bus dédiés qui le couplent étroitement au PS. Le PL ne contient non plus le même matériel de configuration qu'un FPGA 7-series typique, et il doit être configuré directement par le processeur ou via le port JTAG [9].

Le PS est constitué de plusieurs composants, dont l'unité de traitement d'application (APU, qui comprend 2 processeurs Cortex-A9), l'architecture AMBA (Advanced Micro controller Bus Architecture), le contrôleur mémoire DDR3 et divers contrôleurs périphériques avec leurs entrées et sorties multiplexées à 54 broches (appelées E / S multiplexées ou broches MIO). Les contrôleurs périphériques dont les entrées et les sorties ne sont pas connectées aux broches MIO peuvent à la place acheminer leurs E / S via le PL, via l'interface EMIO (Extended-MIO). Les contrôleurs périphériques sont connectés aux processeurs en tant qu'esclaves via l'interconnexion AMBA et contiennent des registres de contrôle lisibles / inscriptibles qui peuvent être adressés dans l'espace mémoire des processeurs. La logique programmable est également connectée à l'interconnexion en tant qu'esclave, et les conceptions peuvent implémenter plusieurs coeurs dans la structure FPGA qui contiennent chacun des registres de contrôle adressables. De plus, les cœurs implémentés dans le PL

peuvent déclencher des interruptions vers les processeurs et effectuer des accès DMA à la mémoire DDR3 [9].

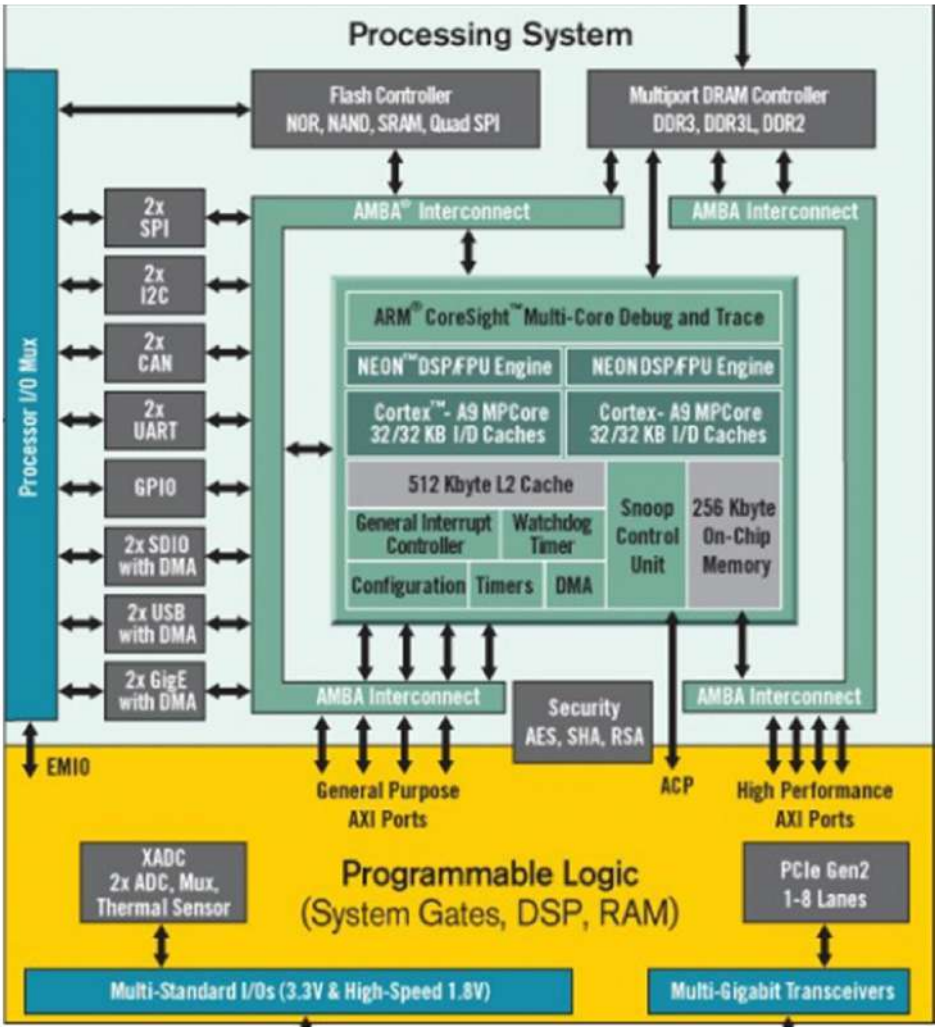


Figure I.4 Architecture de Zynq APSoC [9]

I.4.2 Caractéristiques [10]

- **ZYNQ XC7Z020-1CLG400C**
  - Processeur Cortex-A9 dual-core 650 MHz
  - Contrôleur de mémoire DDR3 avec 8 canaux DMA et 4 ports esclaves AXI3 haute performance
  - Contrôleurs périphériques à bande passante élevée: Ethernet 1G, USB 2.0, SDIO
  - Contrôleur périphérique à faible bande passante: SPI, UART, CAN, I2C
  - Programmable à partir de JTAG, du flash Quad-SPI et de la carte micro SD
  - Logique programmable équivalente à Artix-7 FPGA

        - 13 300 slices logiques, chacune avec quatre LUT à 6 entrées et 8 bascules
        - 630 Ko de RAM
        - 4 tuiles de gestion d'horloge, chacune avec une boucle à verrouillage de phase (PLL) et un gestionnaire d'horloge en mode mixte (MMCM)
        - 220 slices DSP
        - Convertisseur analogique-numérique sur puce (XADC)
- **Mémoire**
    - DDR3 512 Mo avec bus 16 bits à 1050 Mbits / s
    - Flash Quad-SPI de 16 Mo avec identificateur compatible EUI-48/64 ™ 48 bits programmé en usine
    - emplacement micro SD
- **Puissance**
    - Alimenté par USB ou toute source d'alimentation externe 7V-15V
- **USB et Ethernet**
    - Gigabit Ethernet PHY
    - Circuit de programmation USB-JTAG
    - Pont USB-UART
    - USB OTG PHY (prend en charge l'hôte uniquement)
- **Audio et vidéo**
    - Port de récepteur HDMI (entrée)
    - Port source HDMI (sortie)
    - Microphone avec interface PDM
    - Sortie audio mono pilotée par PWM avec jack 3,5 mm
- **Commutateurs, boutons -poussoirs et voyants**
    - 4 boutons -poussoirs
    - 2 interrupteurs à coulisse
    - 4 LEDs
    - 2 LED RVB
- **Connecteurs d'extension**
    - Deux ports Pmod standard
        - 16 E / S FPGA totales
    - Connecteur Arduino / chipKITShield
        - 49 E / S FPGA totales

- 6 entrées analogiques 0-3.3V asymétriques vers XADC
- 4 Entrées analogiques différentielles 0-1.0V vers XADC

Alimentations

Le PYNQ-Z1 peut être alimenté par le port Digilent USB-JTAG-UART (J14) ou par un autre type de source d'alimentation, telle qu'une batterie ou une alimentation externe. Le cavalier JP5 (près de l'interrupteur d'alimentation) détermine la source d'alimentation utilisée.

I.4.3 Configuration de ZYNQ

Contrairement aux périphériques FPGA Xilinx, les périphériques APSoC tels que le Zynq-7020 sont conçus autour du processeur, qui agit comme un maître de la structure logique programmable et de tous les autres périphériques sur puce du système de traitement. Cela entraîne que le processus de démarrage de Zynq soit plus similaire à celui d'un microcontrôleur qu'un FPGA. Ce processus implique que le processeur charge et exécute une image de démarrage Zynq, qui comprend un chargeur de démarrage de premier niveau (FSBL), un train de bits pour la configuration de la logique programmable (facultatif) et une application utilisateur. Le processus de démarrage est divisé en trois étapes [10]:

Étape 1

Après la mise sous tension du PYNQ-Z1 ou la réinitialisation de Zynq, l'un des processeurs (CPU0) commence à exécuter un morceau interne de code en lecture seule appelé BootROM. Si et seulement si le Zynq venait d'être mis sous tension, le BootROM verrouillera d'abord l'état des broches de mode dans le registre de mode (les broches de mode sont attachées à JP4 sur le PYNQ-Z1). Si le BootROM est exécuté en raison d'un événement de réinitialisation, les broches de mode ne sont pas verrouillées et l'état précédent du registre de mode est utilisé. Cela signifie que le PYNQ-Z1 a besoin d'un cycle d'alimentation pour enregistrer tout changement dans le cavalier du mode de programmation (JP4). Ensuite, le Boot ROM copie un FSBL à partir d'une mémoire non volatile spécifiée par le registre de mode sur les 256 Ko de mémoire vive interne de l'APU (appelée mémoire sur puce, ou OCM). Le FSBL doit être enveloppé dans une image de démarrage Zynq pour que le Boot ROM le copie correctement. La dernière chose que fait le Boot ROM est de transférer l'exécution au FSBL dans OCM.

Étape 2

Au cours de cette étape, le FSBL termine d'abord la configuration des composants PS, tels que le contrôleur de mémoire DDR. Ensuite, si un flux de bits est présent dans l'image de démarrage ZYNQ, il est lu et utilisé pour configurer le PL. Enfin, l'application utilisateur est chargée en mémoire à partir de l'image de démarrage ZYNQ, et l'exécution lui est transmise.

Étape 3

La dernière étape est l'exécution de l'application utilisateur chargée par le FSBL. Cela peut être n'importe quel type de programme, depuis un simple design «Hello World», jusqu'à un chargeur de démarrage de seconde étape utilisé pour démarrer un système d'exploitation comme Linux.

L'image de démarrage Zynq est créée à l'aide du kit de développement logiciel Vivado et Xilinx SDK.

I.5 Conclusion

Dans ce chapitre, nous avons présente une vue d'ensemble sur les circuits logiques programmable. Une description des circuits logiques programmables et plus précisément les FPGA ont été étudies.

Nous avons présenté dans une dernière partie la carte de développement PYNQ de Xilinx qui sera utilisé par la suite dans l'implémentation d'un réseau de neurone binarisés.

L'objectif principal de l'utilisation des FPGAs pour l'implémentation d'un réseau de neurone binarisés sur puce est de minimiser l'écart entre le développement de l'idée/algorithme et la conception du système embarque. Aussi, l'FPGA présente d'autres avantages tels que, sa reconfigurabilité, son faible coût et son faible délai de mise sur le marché.

II.1 Introduction :

Les réseaux de neurones artificiels ou réseaux connexionnistes sont fondés sur des modèles qui tentent d'expliquer comment les cellules du cerveau et leurs interconnexions parviennent, d'un point de vue global, à exécuter des calculs complexes.

Ces systèmes qui stockent et retrouvent l'information de manière "similaire" au cerveau sont particulièrement adaptés aux traitements en parallèle de problèmes complexes comme la reconnaissance automatique de la parole, la reconnaissance de visages ou bien la simulation de fonctions de transfert. Ils offrent donc un nouveau moyen de traitement de l'information utilisé en reconnaissance de formes (vision, image, parole, etc.).

II.2 Historique

La première modélisation d'un neurone date de 1943. Elle a été présentée par McCulloch et Pitts. L'interconnexion de ces neurones permet le calcul de plusieurs fonctions logiques. En 1949, Hebb propose le premier mécanisme dévolution des connexions, appelées (par analogie aux systèmes biologiques) des synapses. L'association de ces deux méthodes permit à Rosenblatt en 1958 de décrire le Premier modèle opérationnel de réseaux de neurones: le perceptron. Celui-ci est capable d'apprendre à calculer un grand nombre de fonctions booléennes, mais pas toutes. Ses limites théoriques furent mises en évidence par Minsky et Papert en 1969. Depuis 1985, de nouveaux modelés mathématiques ont permis de les dépasser et ont donné naissance aux réseaux multicouches que nous étudierons plus particulièrement. [11]

II.3 Le neurone

II.3.1 Structure de neurone biologique

Le neurone est une cellule composée d'un corps cellulaire et d'un noyau. Le corps cellulaire se ramifie pour former ce que l'on nomme les dendrites. Celles-ci sont parfois si nombreuses que l'on parle alors de chevelure dendritique ou d'arborisation dendritique. C'est par les dendrites que l'information est acheminée de l'extérieur vers le soma, corps du neurone. [11]

L'information traitée par le neurone chemine ensuite le long de l'axone (unique) pour être transmise aux autres neurones. La transmission entre deux neurones n'est pas directe. En fait, il existe un espace intercellulaire de quelques dizaines d'Angströms [10^{-9} m] entre

l'axone du neurone afférent et les dendrites (on dit une dendrite) du neurone efférent. La jonction entre deux neurones est appelée la synapse (Figure II.1).

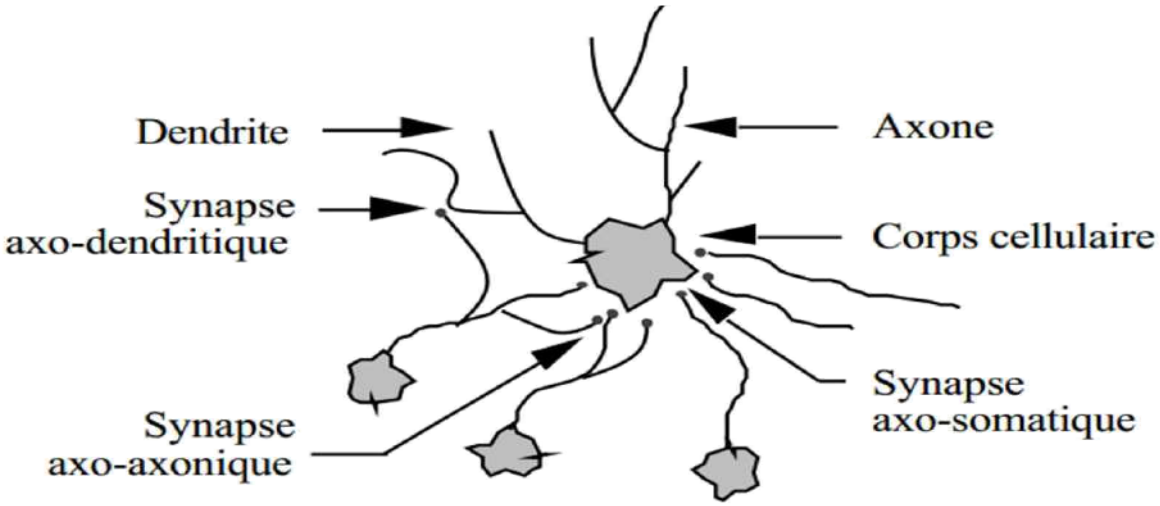


Figure II.1 Un neurone avec son arborisation dendritique [12]

II.3.2 Structure de neurone artificiel

Figure.II.2 montre la structure d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associé un poids w abréviation de weight (poids en anglais) représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals. A chaque connexion est associé un poids. **[11]**

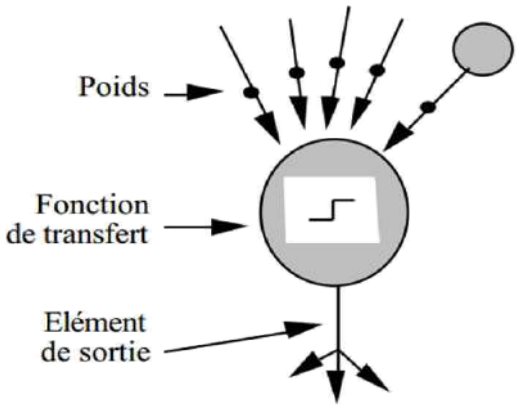


Figure II.2 Structure de Neurone artificiel [12]

Neurone biologique	Neurone artificial
Synapses	Poids de connexion
Axones	Signal de sortie
Dendrite	Signal d'entrée
Somma	Fonction d'activation

Tableau II.1: Transition du neurone biologique au neurone artificiel [12]

II.3.3 Les connexions entre les neurones

Les connexions permettent de construire le réseau de neurones en liant les différentes entités ensemble. La plus simple et la plus classique des connexions est la connexion unidirectionnelle pondérée. Il s'agit d'un arc unidirectionnel pondéré par un poids **wij** qui relie le neurone **j** au neurone **i** et qui correspond au poids utilisé dans la fonction de sommation. Ces connexions simples sont représentées dans la (Figure II.3)**.** [13]

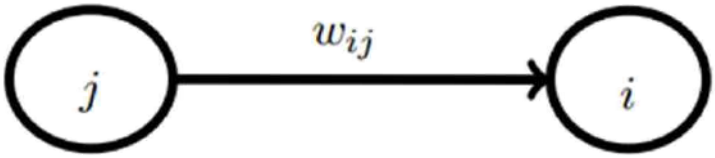


Figure II.3 Connexion simple entre deux neurones

II.3.4 Structure d'interconnexion

Les connexions entre les neurones qui composent le réseau décrivent la topologie du modèle. Elle peut être quelconque, mais le plus souvent il est possible de distinguer une certaine régularité. [13]

- *Réseau multicouche (au singulier)*

Les neurones sont arrangés par couche. Il n'y a pas de connexion entre neurones d'une même couche et les connexions ne se font qu'avec les neurones des couches avalent (Figure II.4). Habituellement, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau et donc définir les concepts de neurone d'entrée, neurone de sortie. Par extension, on appelle couche d'entrée l'ensemble des

neurones d'entrée, couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelés couches cachées. [11]

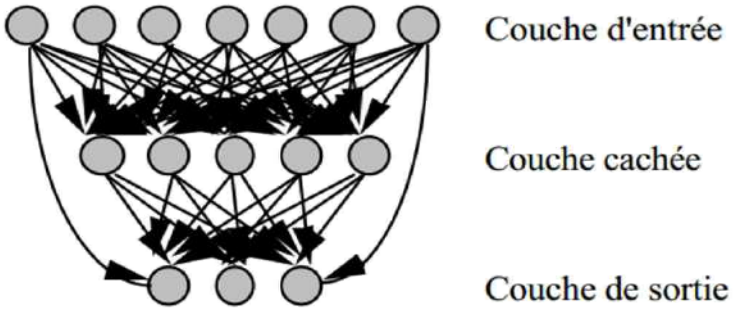


Figure II.4 Définition des couches d'un réseau multicouche. [11]

- *Réseau à connexions locales :*

Il s'agit d'une structure multicouche, mais qui à l'image de la rétine, conserve une certaine topologie. Chaque neurone entretien des relations avec un nombre réduit et localisé de neurones de la couche avale (Figure II.5.). Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique. [11]

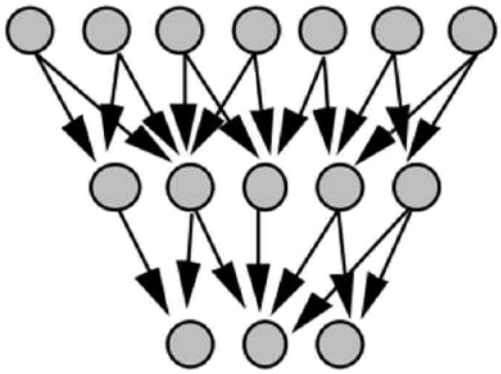


Figure II.5 Réseau à connexions locales [11]

- *Réseau à connexions récurrentes :*

Les connexions récurrentes ramènent l'information en arrière par rapport au sens de propagation défini dans un réseau multicouche. Ces connexions sont le plus souvent locales (Figure II.6).

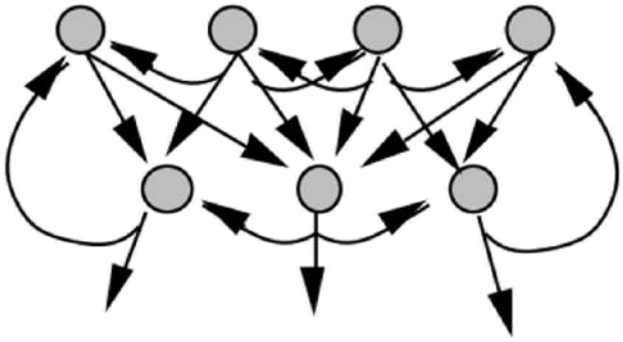


Figure II.6 Réseau à connexions récurrentes [11]

- *Réseau à connexion complète :*

C'est la structure d'interconnexion la plus générale (Figure II.7.). Chaque neurone est connecté à tous les neurones du réseau (et à lui-même).

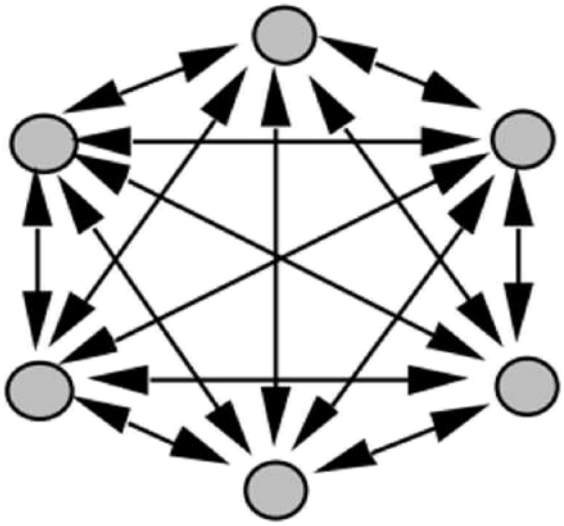


Figure II.7 Réseau à connexions complète [11]

II.3.5 Différentes architectures neuronales

Architectures d'un réseau de nuerons est l'organisation des neurones entre eux au sein d'un même réseau. Autrement dit, il s'agit de la façon dont ils ordonnés et connectés.

La majorité des réseaux de neurones utilise le même type de neurones. Quelques architectures plus rares se basent sur des neurones dédiés. [14]

L'architecture d'un réseau de neurones dépend de la tâche à apprendre (problème à résoudre).

Un réseau de neurone est en général composé de plusieurs couches de neurones, des entrées jusqu'aux sorties.

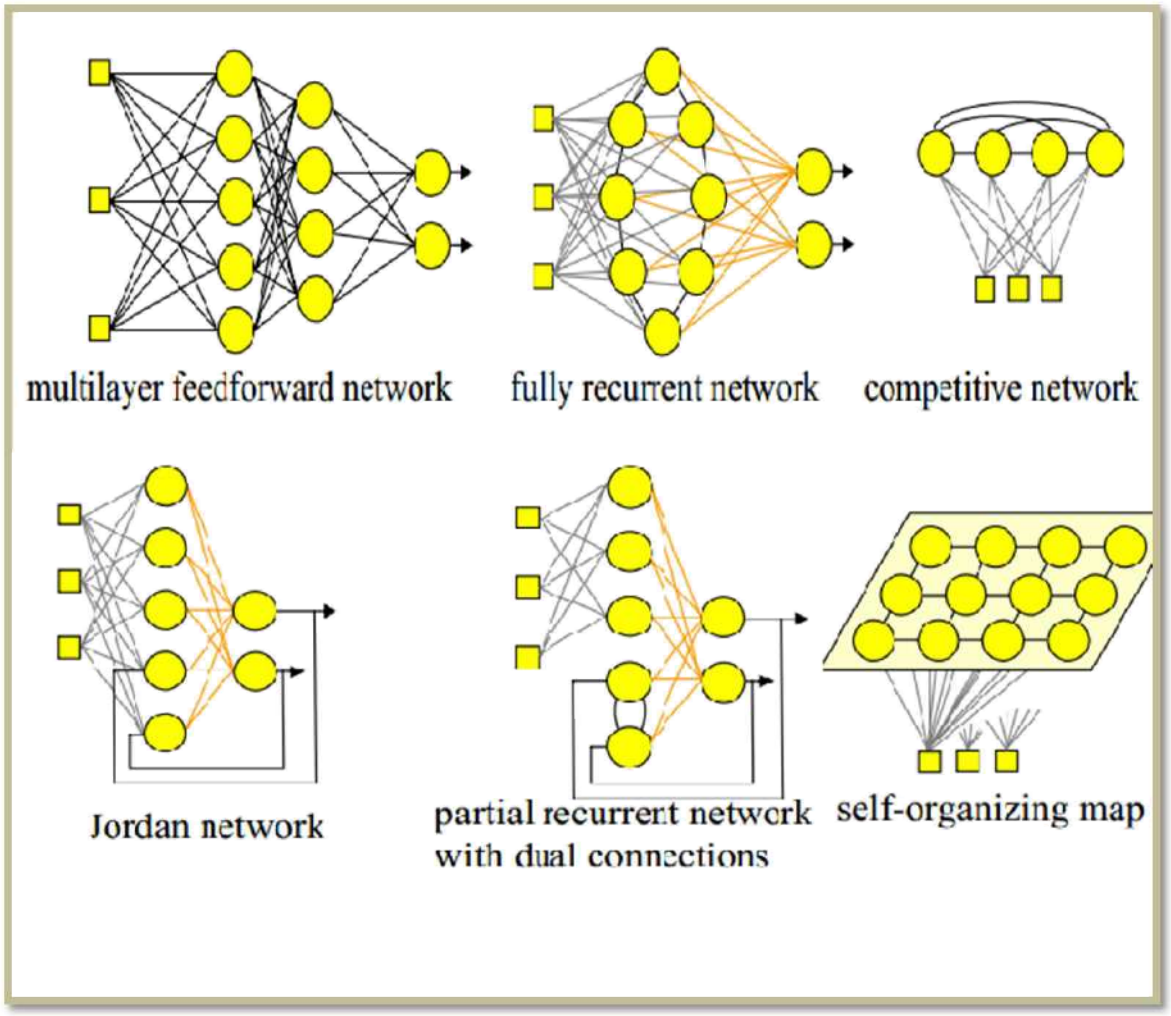


Figure II.8 Différente architectures neuronales. **[12]**

II.3.6 Apprentissage d'un réseau de neurone

L'apprentissage est une étape très importante du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié itérativement jusqu'à l'obtention du comportement désiré, et ce par l'ajustement des poids (connexion ou synapse) des neurones à une source d'information bien définit (Hebb 1949; Grossberg 1982; Rumelhart et al. 1986). L'apprentissage consiste également à extraire des conformités parmi les données utilisées pour l'entrainement du réseau mais l'objectif essentiel de l'apprentissage est la résolution du problème par la prévision, la classification ... etc. (Weiss et Kulikowski 1991). [15]

Pour un RNA, l'apprentissage peut être regardé également comme étant le processus de la mise à jour des poids (connexion) au sein du réseau dans le but d'ajuster la réponse du réseau à l'expérience et aux exemples.

Il existe plusieurs types de règles d'apprentissage qui peuvent être rassemblées en deux catégories qui sont l'apprentissage supervisé et l'apprentissage non supervisé. [15]

Apprentissage supervisé

Dans ce type d'apprentissage, on cherche à imposer au réseau un fonctionnement donné en forçant les sorties des réseaux à prendre des valeurs bien données (choisie par l'opérateur) et ce en modifiant les poids synaptiques.

Le réseau se comporte alors comme un filtre dont les paramètres de transfert sont ajustés à partir des couples entrée-sortie présentés (Hassoum 1995).

L'adaptation des paramètres du réseau s'effectue à partir d'un algorithme d'optimisation, l'initiation des poids synaptiques étant le plus souvent aléatoire. [16]

Apprentissage non supervisé

Contrairement à l'apprentissage supervisé, seules les valeurs d'entrée sont disponibles dans ce type d'apprentissage non supervisé qui est appelé aussi « apprentissage par compétition ». Dans ce cas, les exemples présentés à l'entrée provoquent une auto adaptation du réseau afin d'engendrer des valeurs de sortie qui soient proches en réponse à des valeurs d'entrée similaires. [16]

II.3.7 Interprétation mathématique

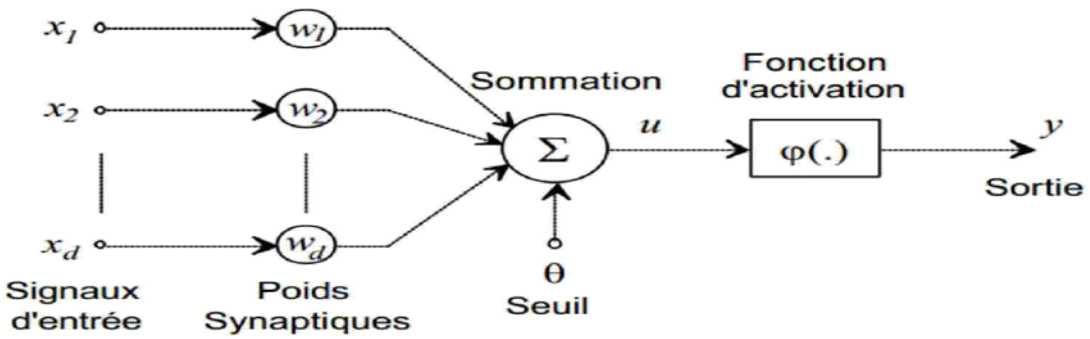


Figure II.9 La composition d'un neurone artificiel [12]

Le schéma montre la composition d'un neurone artificiel. Chaque neurone faisant partie du réseau est une machine élémentaire. Il reçoit un certain nombre de variables d'entrées provenant de neurones en amont. À chacune de ces entrées, est associé un poids qui représente la force de la connexion entre les deux neurones. Chaque neurone est doté d'une unique sortie qui se ramifie ensuite pour représenter les entrées qui alimenteront d'autres neurones en aval. Pour résumer, chaque neurone calcule une sortie unique en se basant sur les informations qui lui sont données. [17]

Le neurone en tant qu'unité élémentaire agit de la façon suivante :

- 1ère phase : Le neurone fait le calcul de la somme pondérée des entrées (en fonction de la force des connexions). L'apprentissage ayant été réalisé auparavant, le poids des connexions est ici déjà déterminé et fixe.
- 2^{e} phase : À partir de la valeur obtenue, une fonction d'activation ou de transfert va calculer la valeur de l'état du neurone. Le neurone compare la somme pondérée des entrées à une valeur de seuil et fournit alors une réponse en sortie.

Fonction d'activation

La fonction d'activation (ou fonction de transfert) sert à convertir le résultat de la somme pondérée des entrées d'un neurone en une valeur de sortie, cette conversion s'effectue par un calcul de l'état du neurone en introduisant une non-linéarité dans le fonctionnement du neurone (Cybenko 1989)

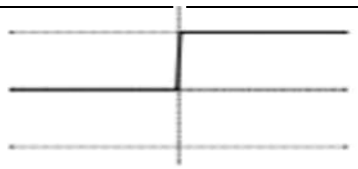
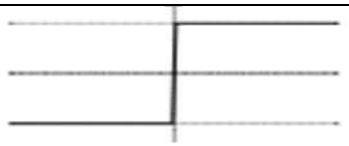
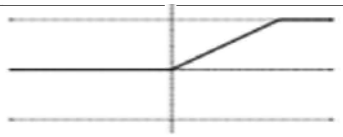
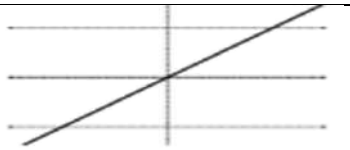
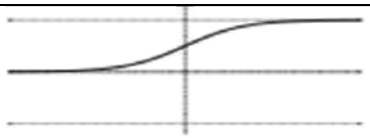
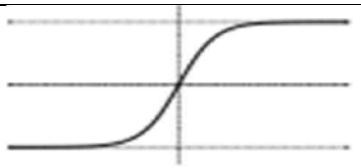
Le biais b joue un rôle de seuil, quand le résultat de la somme pondérée dépasse ce seuil, l'argument de la fonction de transfert devient positif ou nul; dans le cas contraire, il est considéré négatif. Finalement si le résultat de la somme pondérée est [17]:

1- en dessous du seuil, le neurone est considéré comme non-actif

2- aux alentours du seuil, le neurone est considéré en phase de transition.

3- au-dessus du seuil, le neurone est considéré comme actif

Il y a plusieurs types de fonctions de transfert qui peuvent être utilisées dans les RNA, les fonctions d'activation souvent utilisées sont représentées dans le tableau suivant :

Nom	Valeur	Représentation
Seuil	f (x) = 0six ≤ 0 ≥ f (x) = 1 si x > 0	
Seuil symétrique	f (x) = -1 si x < 0 f (0) = 0 f (x) = 1 si x > 0	
Linéaire saturée	f (x) = 0 si x < 0 f (x) = x si 0 ≤ x ≤ 1 f (x) = 1 si x > 1	
Linéaire	f (x) = x	
Sigmoïde	$f(x) = \frac{1}{1+e^{-x}}$	
Tangente hyperbolique	$f(x) = \frac{e^{x}- e^{-x}}{e^{x}+ e^{-x}}$	

Tableaux II.2 fonctions d'activation

II.3.8 Réseaux neuronaux bayésiens

Les réseaux bayésiens sont de graphes acycliques orientés où les variables sont représentées par des nœuds et les relations de dépendance ou de corrélation entre les variables sont représentées par des arcs directionnels.

Chaque variable est représentée par un tableau de probabilités qui sont déterminées en utilisant le théorème de Bayes:

P (H/E) = P (H/E)*P (H)/P (E).

Si l'hypothèse H est basée sur des données observées et E est l'évidence (la preuve), alors P(H) est une probabilité a priori de l'hypothèse, en fait le degré initial de confiance dans l'hypothèse. P(E|H) est la vraisemblance des données observées, donc la mesure dans laquelle

l'évidence a été observée quand l'hypothèse était vraie. P(H|E) est la probabilité a posteriori de l'hypothèse étant donné l'évidence.

Le théorème de Bayes offre l'adaptabilité et la flexibilité qui permettent à l'utilisateur de réviser et changer les estimations et les prédictions si de nouvelles données pertinentes sont accueillies.
Les réseaux bayésiens sont la représentation de la dépendance entre un ensemble de variables. Un arc entre deux variables, disons de V à, W, dénote qu'il existe une relation de dépendance directe du V sur W et dans ce cas V est un parent de W. Les tableaux de probabilités associés aux nœuds sont la distribution de probabilités du nœud en considérant tous les parents de celui-là. [18]

II.3.8.1 Chois du Réseaux bayésiens

Selon le type d'application, l'utilisation pratique d'un réseau bayésien peut être envisagée au même titre que celle d'autres modèles : réseaux neuronaux, systèmes experts, arbres de décision, modèles d'analyse de données (régressions linéaires), arbres de défaillances, modèles logiques, etc. Naturellement, le choix de la méthode fait intervenir différents critères, comme la facilité, le coût et le délai de mise en .œuvre d'une solution. En dehors de toute considération théorique, les aspects suivants des réseaux bayésiens les rendent, dans de nombreux cas, préférables à d'autres modèles [18]:

- Acquisition des connaissances.
- Représentation des connaissances.
- Utilisation des connaissances.
- Qualité de l'offre en matière de logiciels

II.3.8.2 La mise en œuvre du réseau Bayésien

Nous avons vu diverses techniques pour effectuer de l'inférence bayésienne dans les réseaux Bayésiens. Mais avant de pouvoir utiliser ces modèles, il faut pouvoir les construire. La construction d'un réseau bayésien se décompose en trois étapes distinctes présentées au-dessous :

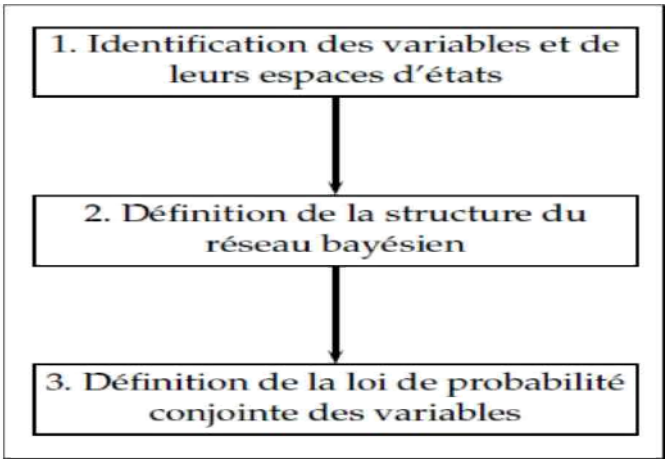


Figure II.10 Étapes de construction d'un réseau bayésien [19]

La première étape est l'identification de variables et pour chaque variable l'ensemble de ses valeurs possibles, pour cette étape l'intervention des experts du système est toujours nécessaire.

La deuxième étape est la définition de la structure du réseau bayésien, trouver les liens d'influence entre les variables tout en s'assurant qu'il n'y a pas de boucle ou cycle.

La dernière étape vise la création des tableaux de probabilités pour les variables, soit de variables sans parentes pour lesquelles des probabilités marginales doivent être définies, soit de variables qui ont de variables partantes et, dans ce cas, de probabilités conditionnelles sont définies. [18]

II.3.8.3 Structures de réseaux bayésiens pour la classification

Dans les tâches de classification, une variable précise correspond à la classe qu'il faut « reconnaître » à partir des autres variables (les caractéristiques). Plusieurs méthodes d'apprentissage vont donc proposer des structures où ce nœud classe aura un rôle central [18]

Structure de Bayes naïve

Le classifieurs de Bayes naïf correspond à la structure la plus simple qui soit, en posant l'hypothèse que les caractéristiques $X_1 \ldots X_{n-1}$ sont indépendantes conditionnellement à la classe Xc. Cela mène à la structure type de la (Figure.II.10) Cette structure, pourtant très simple, donne de très bons résultats dans de nombreuse applications [18]

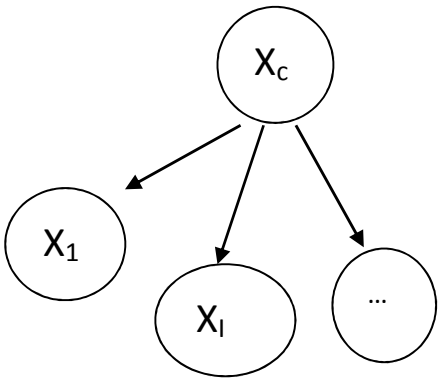


Figure II.11 Réseau bayésien naïf

Structure augmentée

Afin d'alléger l'hypothèse d'indépendance conditionnelle des caractéristiques, il a été suggéré « d'augmenter » la structure naïve en ajoutant des liens entre certaines caractéristiques.

Parmi les différentes méthodes avancées pour augmenter le réseau bayésien naïf, citons TANB (Tree Augmented Naive Bayes) qui utilise une structure naïve entre la classe et les caractéristiques et un arbre reliant toutes les caractéristiques. [18]

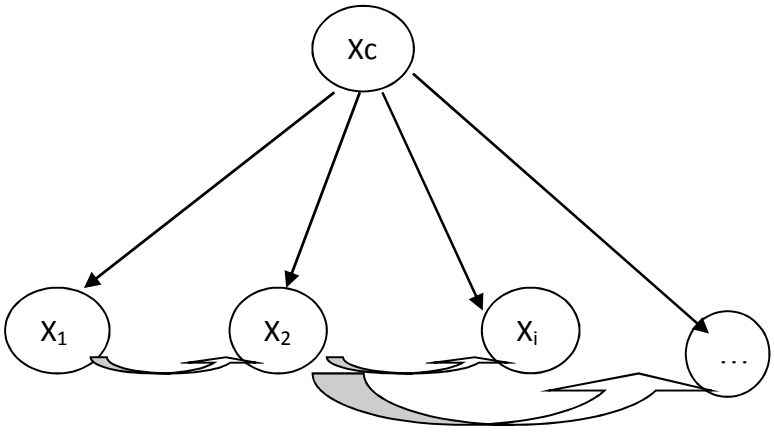


Figure II.12 Réseau bayésien naïf augmenté (par un arbre)

II.3.8.4 Les étapes de conception d'un réseau de neurones :

Choix et préparation des échantillons

Le processus d'élaboration d'un réseau de neurones commence par le choix et la préparation des échantillons de données. En général une partie de l'ensemble de données est simplement écarte de l'échantillon d'apprentissage et conserve pour les tests hors échantillon afin de vérifier la capacité de généralisation du réseau. [20]

Elaboration de la structure du réseau

La structure du réseau dépend étroitement du type de réseau : un perceptron standard, un réseau de Hopfield, un réseau de Kohonen, etc. Ensuite en fonction du type choisi, l'élaboration de la structure du réseau peut varier : dans le cas du perceptron, il faudra aussi déterminer le nombre de neurones dans la couche cachée. [20]

Apprentissage Bayésien

Un réseau Bayésien est constitué à la fois d'un graphe et d'un ensemble de probabilités conditionnelles.

L'apprentissage d'un réseau Bayésien doit donc répondre aux deux questions suivantes [20]:

• Comment estimer les lois de probabilités conditionnelles ?

• Comment trouver la structure du réseau Bayésien ?

Nous allons donc séparer le problème de l'apprentissage en deux parties :

• L'apprentissage des paramètres, où nous supposerons que la structure du réseau a été fixée, et où il faudra estimer les probabilités conditionnelles de chaque nœud du réseau.

• L'apprentissage de la structure, où le but est de trouver le meilleur graphe représentant la tâche à résoudre.

En faisant l'hypothèse classique d'indépendance des paramètres, les paramètres des distributions de probabilités conditionnelles de chaque variable sont estimés indépendamment des autres variables.

Les méthodes d'apprentissage des réseaux Bayésiens utilisent en pratique la méthode du maximum de vraisemblance et fréquemment l'approche Bayésienne pour estimer les paramètres du modèle. Dans la méthode Bayésienne, les lois a prioris des paramètres sont souvent déterminées par un expert ou définies de manière uniforme. De plus, les algorithmes d'apprentissage de la structure c'est-à-dire la recherche d'un "bon" graphe, qui correspond aux données et aux a prioris, se base aussi sur la maximisation d'une fonction, appelée

fonction score, sur laquelle des hypothèses sont aussi faites concernant la distribution a priori des paramètres.

Généralement, l'apprentissage des paramètres se fait en utilisant la méthode du maximum de vraisemblance. Si cette approche paraît naturelle, elle présente néanmoins un inconvénient majeur dans le cas d'une base d'apprentissage de taille limitée ; si une configuration particulière peut exister, avec une probabilité faible non nulle et qu'elle n'est pas présente dans la base de données, alors l'approche du maximum de vraisemblance aboutit à une estimation de la probabilité nulle. Or, le fait qu'une configuration n'a pas été observée ne signifie pas nécessairement qu'elle a une probabilité nulle. Afin d'y remédier, il serait bon de pouvoir exprimer et quantifier la possibilité de la survenance d'un tel événement. C'est cet objectif que remplit l'approche Bayésienne.

II.4Conclusion

Les neurones « artificiels » se veulent au plus proche du réel fonctionnement de ceux biologiques. De plus, les scientifiques en viennent à imaginer des réseaux entièrement artificiels. C'est pourquoi, le développement de ces réseaux de neurones artificiels pourrait bien aboutir un jour à la création de diverses intelligences artificielles (ordinateur intelligent, robot humanoïde...).

Les réseaux Bayésiens sont souvent utilisés car ils ont plusieurs avantages par rapport à d'autres techniques. L'apprentissage d'un réseau Bayésien consiste à estimer le graphe (la structure) et les paramètres des distributions de probabilités conditionnelles associées à ce graphe. Les algorithmes d'apprentissage de réseaux Bayésiens utilisent en pratique une approche Bayésienne classique d'estimation à posteriori dont les paramètres sont souvent déterminés par un expert ou définis de manière uniforme Le cœur de cette thèse concerne l'application aux réseaux Bayésiens de plusieurs avancées dans le domaine de reconnaissance d'image .

III.1 Introduction

Ce chapitre a pour objectif de présenter comment utiliser un réseau de neurones artificiel dans le domaine de reconnaissance d'image et de décrire le processus de classification et d'identification d'image à rechercher. Il s'agit d'un réseau de neurones Bayésiens embarqué dans un circuit FPGA de la carte **Pynq** ; l'objectif de ce chapitre de montrer l'avantage de l'exécution du réseau Bayésiens BNN dans une architecture hardware dans la zone PL de l'FPGA par rapport à une exécution software dans le processeur ARM présent dans le circuit Zunq.

III.2 Jupyter Notebook

Jupyter Notebook est une application Web open-source qui vous permet de créer et de partager des documents contenant du code en direct, des équations, des visualisations et du texte narratif. Les utilisations incluent: le nettoyage et la transformation des données, la simulation numérique, la modélisation statistique, la visualisation des données, l'apprentissage automatique et bien plus encore [21].

Le Jupyter Notebook est un environnement informatique interactif qui permet aux utilisateurs d'écrire des documents de cahiers qui incluent:

- ✓ Code en direct
- ✓ Widgets interactifs
- ✓ Parcelles
- ✓ Texte narratif
- ✓ Équations
- ✓ Images
- ✓ Vidéo

III.2.1 Composants

Le bloc-notes Jupyter combine trois composants:

L'application Web de bloc-notes : Une application Web interactive pour écrire et exécuter du code de manière interactive et créer des documents de cahier.

Noyaux : processus séparés démarrés par l'application Web du cahier qui exécute le code des utilisateurs dans une langue donnée et renvoie la sortie à l'application Web du cahier. Le

noyau gère également des choses comme les calculs pour les widgets interactifs, l'achèvement des tabulations et l'introspection [21].

Documents de cahier : Documents autonomes contenant une représentation de tout le contenu de l'application Web de cahier, y compris les entrées et les sorties des calculs, le texte narratif, les équations, les images et les représentations multimédias des objets. Chaque document de cahier a son propre noyau [21].

III.2.2 Jupyter Notebook en PYNQ

PYNQ est écrit en Python, qui est le noyau par défaut pour Jupyter Notebook, et le seul noyau installé pour Jupyter Notebook dans la distribution PYNQ.

Les noyaux communiquent avec l'application Web du cahier et le navigateur Web en utilisant un protocole de message JSON sur ZeroMQ / Web Sockets. La plupart des utilisateurs n'ont pas besoin de connaître ces détails, mais il est important de comprendre que les noyaux s'exécutent sur Zynq, alors que le navigateur Web sert d'interface avec ce noyau [21].

III.3 PYTHON

III.3.1 Historique

Python est un langage de programmation, dont la première version est sortie en 1991. Créé par **Guido van Rossum**, il a voyagé du Macintosh de son créateur, qui travaillait à cette époque au *Centrum voor Wiskunde en Informatica* aux Pays-Bas, jusqu'à se voir associer une organisation à but non lucratif particulièrement dévouée, la **Python Software Foundation**, créée en 2001. Ce langage a été baptisé ainsi en hommage à la troupe de comiques les « Monty Python » [22].

III.3.2 Technique de production de Python

• Technique mixte : l'interprétation du byte code compilé. Bon compromis entre la facilité de développement et la rapidité d'exécution ;

• Le byte code (forme intermédiaire) est portable sur tout ordinateur muni de la machine virtuelle Python [21].



Figure III.1– Interprétation du byte code compilé [22]

Pour exécuter un programme, Python charge le fichier source .PY (ou .PYW) en mémoire vive, en fait l'analyse syntaxique, produit le byte code et enfin l'exécute. Pour chaque module importé par le programme, Python vérifie d'abord s'il existe une version précompilée du byte code (dans un fichier .PYO ou .PYC) dont la date correspond au fichier .PY. S'il y en a un, Python l'utilise, sinon il fait une analyse syntaxique du module .PY, et utilise le Byte code qu'il vient de générer.

En pratique, il n'est pas nécessaire de compiler explicitement un module, Python gère ce mécanisme de façon transparente.

III.3.3 La construction des programmes

Le génie logiciel étudie les méthodes de construction des programmes. Plusieurs modèles sont envisageables, entre autres :

• La méthodologie procédurale. On emploie l'analyse descendante (division des problèmes) et remontante (réutilisation d'un maximum de sous algorithmes). On s'efforce ainsi de décomposer un problème complexe en sous-programmes plus simples. Ce modèle structure d'abord les actions [23];

• La méthodologie objet. On conçoit des fabriques (classes) qui servent à produire des composants (objets) qui contiennent des données (attributs) et des actions (méthodes).

Les classes dérivent (héritage et polymorphisme) de classes de base dans une construction hiérarchique.

Python offre les deux techniques.

III.3.4 Les deux modes d'exécution d'un code Python [23]

• Soit on enregistre un ensemble de commandes Python dans un fichier grâce à un éditeur (on parle alors d'un script Python) que l'on exécute par une commande ou par une touche du menu de l'éditeur ;

• Soit on utilise un interpréteur (par exemple IDLE) pour obtenir un résultat immédiat grâce à l'interpréteur Python embarqué dans IDLE qui exécute la boucle d'évaluation

III.3.5 Les caractéristiques de python [23]

- ✓ «Open-source» : son utilisation est gratuite et les fichiers sources sont disponibles et modifiables ;
- ✓ Simple et très lisible ;
- ✓ Doté d'une bibliothèque de base très fournie ;
- ✓ Importante quantité de bibliothèques disponibles : pour le calcul scientifique, les statistiques, les bases de données, la visualisation . . . ;
- ✓ Grande portabilité : indépendant vis à vis du système d'exploitation (Linux, Windows, MacOS) ;
- ✓ Orienté objet ;
- ✓ Typage dynamique : le typage (association à une variable de son type et allocation zone mémoire en conséquence) est fait automatiquement lors de l'exécution du programme, ce qui permet une grande flexibilité et rapidité de programmation, mais qui se paye par une surconsommation de mémoire et une perte de performance ;
- ✓ Présente un support pour l'intégration d'autres langages.

III.3.6 Comment faire fonctionner le code source ?

Il existe deux techniques principales pour traduire un code source en langage machine :

- ✓ La compilation : une application tierce, appelée compilateur, transforme les lignes de code en un fichier exécutable en langage machine. A chaque fois que l'on apporte une modification au programme, il faut recompiler avant de voir le résultat.
- ✓ L'interprétation : un interpréteur s'occupe de traduire ligne par ligne le programme en langage machine. Ce type de langage offre une plus grande commodité pour le développement, mais les exécutions sont souvent plus lentes.

Dans le cas de Python, on peut admettre pour commencer qu'il s'agit d'un langage interprété, qui fait appel des modules compilés. Pour les opérations algorithmiques coûteuses, le langage Python peut s'interfacer à des bibliothèques écrites en langage de bas niveau comme le langage C [23].

III.3.7 Le mode programmation

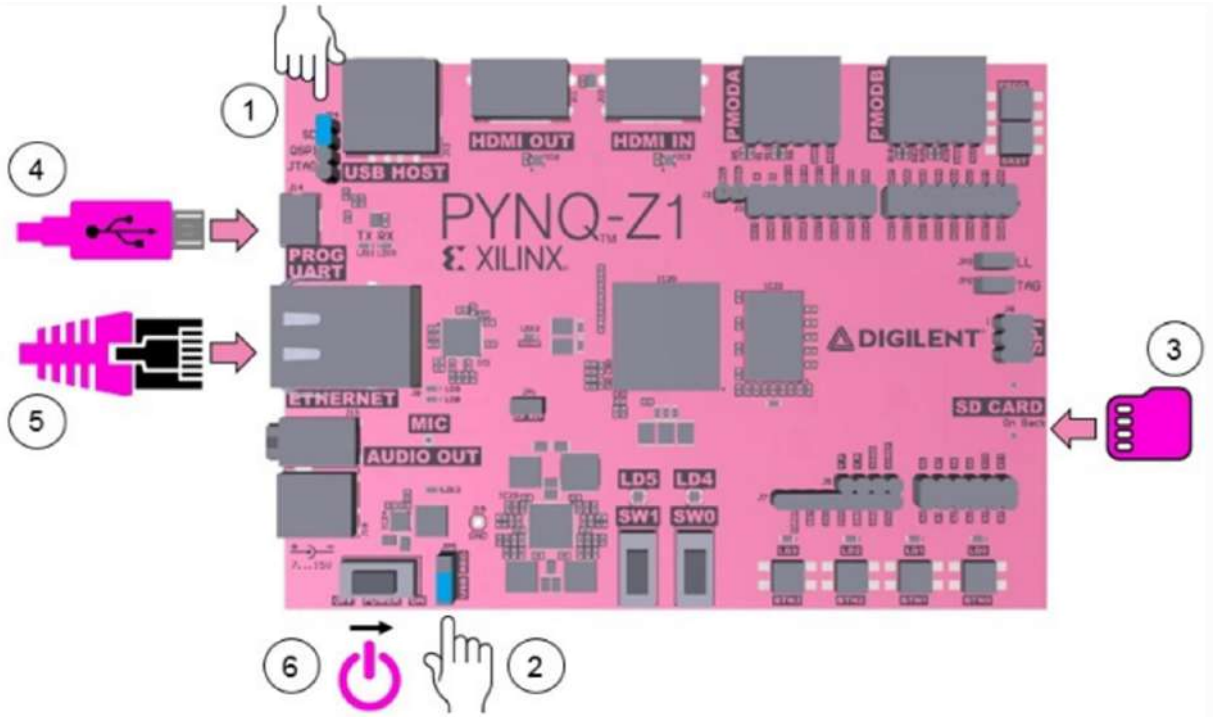
Il s'agit d'écrire dans un fichier une succession d'instructions qui ne seront effectuées que lorsque vous lancerez l'exécution du programme. Cela permet tout d'abord de sauvegarder les commandes qui pourront être utilisées ultérieurement, et d'autre part d'organiser un programme, sous-forme de fichier principal, modules, fonctions . . ..

Le fichier à exécuter devra avoir l'extension .PY, et devra contenir en première ligne le chemin pour accéder au compilateur Python, ainsi que l'encodage [23]:

#!/usr/bin/env python# -*- coding: utf-8 -*-

III.4 Configuration de la carte PYNQ-Z1

1. Réglez le cavalier JP4 / Boot sur la position SD en plaçant le cavalier sur les deux broches supérieures de JP4 comme indiqué sur l'image. (Cela configure la carte pour démarrer à partir de la carte Micro-SD)
2. Pour alimenter le PYNQ-Z1 à partir du câble micro-USB, réglez le cavalier JP5 / Power sur la position USB en plaçant le cavalier sur les deux broches supérieures, comme indiqué sur l'image. (Vous pouvez également alimenter la carte à partir d'un régulateur d'alimentation 12 V externe en réglant le cavalier sur REG .)
3. Insérez la carte Micro SD chargée avec l'image PYNQ-Z1 dans la fente pour carte Micro SD située sous la carte.
4. Connectez le câble USB à votre PC / ordinateur portable et au port Micro USB PROG - UART / J14 du tableau
5. Connectez la carte à Ethernet en suivant les instructions de la configuration Ethernet
6. Allumez le PYNQ-Z1 en suivant les instructions de la section Activation du PYNQ-Z1



Figuier III.2schéma représenté les différentes étapes de Configuration de la carte PYNQ-Z1

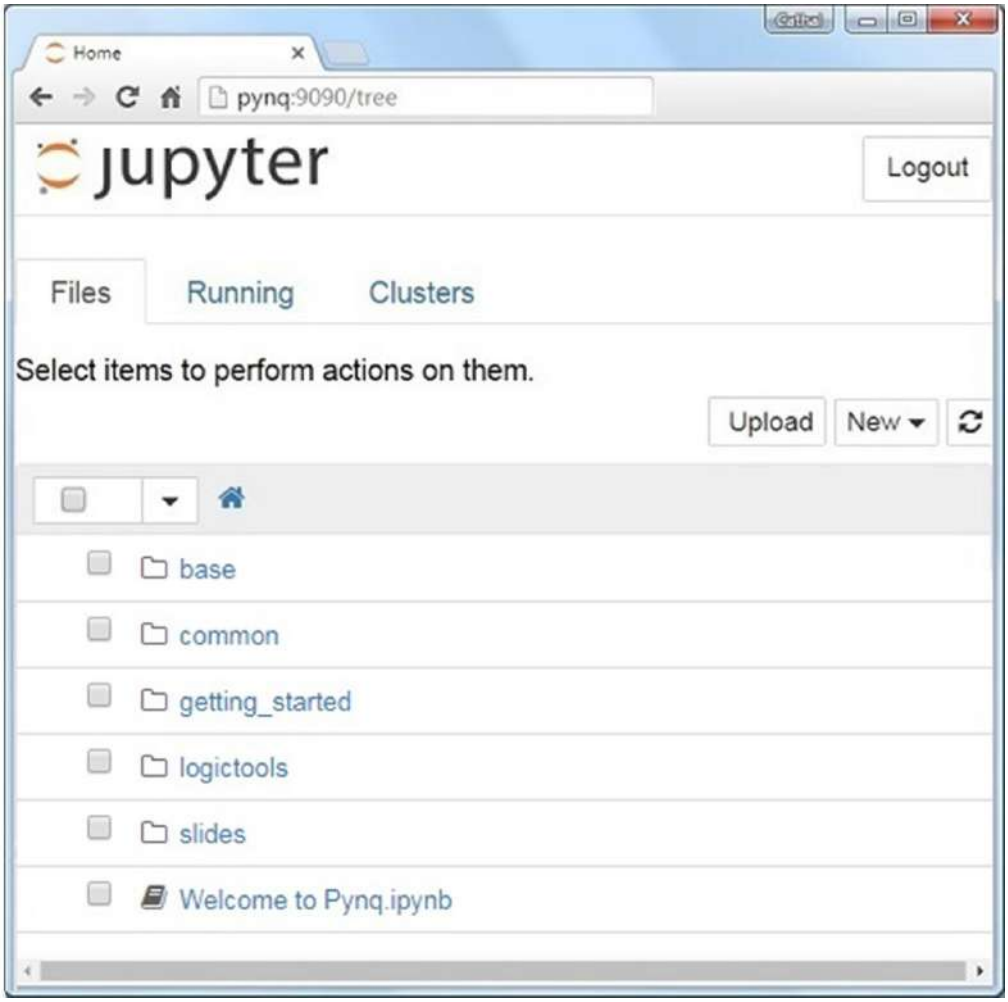
Connexion aux ordinateurs portables Jupyter

Pour vous connecter aux ordinateurs portables Jupyter, ouvrez un navigateur Web et accédez à:

- http: // pynq: 9090 si votre carte PYNQ-Z1 est connectée à un routeur ou un réseau
- http://192.168.2.99:9090 Si votre carte PYNQ-Z1 est connectée à un ordinateur

Si votre carte est configurée correctement, un écran de connexion s'affichera. Le nom d'utilisateur est **xilinx** et le mot de passe est également **xilinx** .

Après vous être connecté, vous recevrez l'écran suivant:



Figuier III.3 l'interface jupyter

Le nom d'hôte par défaut est pynq et l'adresse IP statique par défaut est 192.168.2.99. Si vous avez modifié le nom d'hôte ou l'adresse IP statique du forum, vous devrez modifier l'adresse à laquelle vous accédez.

La première fois que vous vous connectez, votre ordinateur peut mettre quelques secondes à résoudre le nom d'hôte / l'adresse IP.

III.5 BNN sur Pynq

L'exemple présente comment utiliser les réseaux neuronaux binaires sur Pynq. Il montre un exemple de reconnaissance d'image avec un neurone binarisé inspiré du model VGG-16 (Visual Geometry Group). Le réseau de neurones VGG est un réseau de neurones convolutionnels pour la classification d'images. Étant donné une image, le réseau VGG

affichera des probabilités des différentes classes auxquelles une image pourrait potentiellement appartenir (Figure III.4)

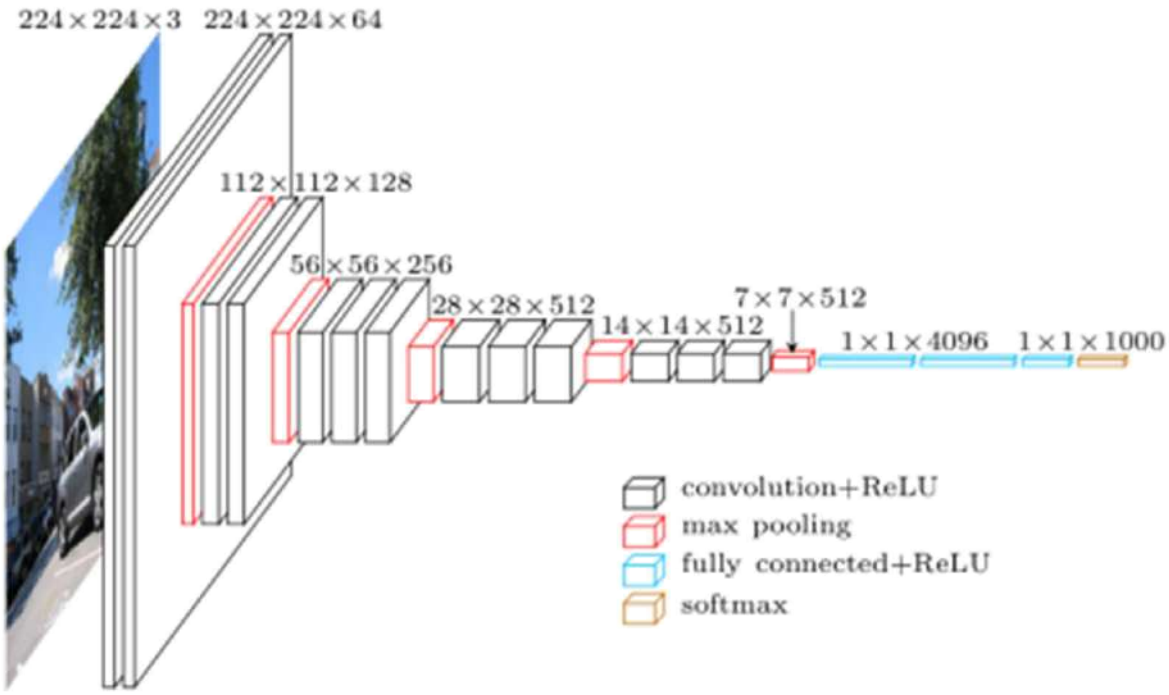


Figure III.4 reconnaissance d'image avec un neurone binarisé

Le model proposé est constitué de 6 couches convolutives, 3 couches MaxPool et 3 couches entièrement connectées. La création d'un classificateur télécharge automatiquement le bitstream sur le périphérique et charge les poids formés sur l'ensemble de données spécifié. Par défaut, il y a trois ensembles de poids à choisir. Cet exemple utilise l'ensemble de données de panneaux de signalisation allemands. L'ensemble de données sélectionné peut classer les images dans 42 classes, dont les noms sont accessibles via le classificateur.

III.5.1 Ouvrir les images à classer :

Dans une première étape les images que nous voulons classer sont chargées et montrées à l'utilisateur. Comme le montre la (Figure III.5.)



Figure III.5 les images que nous voulons classer

III.5.1 Lancement de BNN

Pour montrer l'intérêt des FPGA dans le processus de classification des images, et qui permet de réduire considérablement le temps d'exécution. Nous avons lancé dans une première étape le classificateur BNN dans une architecture hardware dans la zone PL de l'FPGA. Ensuite, dans une seconde étape, nous avons exécuter le programme BNN directement dans le processeur ARM du circuit Zynq, ce que nous considérons comme une exécution software du BNN.

Il faut noter que pour la première étape nous avons utilisé le programme donné dans [24].

Les résultats de l'exécution montrent que le temps nécessaire pour un traitement hardware est de 3874.00 µs, ce qui est très négligeable devant une exécution software qui un temps de 8611883.25 µs.

Après exécution un identificateur de classe est attribué à chaque image (Figure III.6,) ces identificateurs sont utilisés par la suite dans la classification de plusieurs images.

```
Identified classes: [27 17 41 18 14  3]
Identified class name: Pedestrians in road ahead
Identified class name: No entry for vehicular traffic
Identified class name: End of no-overtaking zone
Identified class name: Danger Ahead
Identified class name: Stop
Identified class name: 60 Km/h
```

Figure III.6 Identifier les images que nous voulons classer

Exemple 1 :

Cet exemple va créer une chaîne d'images à partir d'une seule image d'entrée (Figure III.7), pavant l'image pour essayer de localiser un objet. Cette image est constituée de plusieurs panneaux de signalisation, ces panneaux ont la même forme du panneau que nous visons à trouver, qui est le panneau de limitation de vitesse (60 km/h).



Figure III.7 La première image d'entrée

Nous lançons la classification sur toutes les mosaïques de l'image source, et toutes les images dans lesquelles le BNN a identifié un panneau de limitation de vitesse sont affichées **(**Figure III.8.)



Figure III.8 Chercher le panneau dans l'image

Par la suite, la classification est post-analysée afin de ne sélectionner que les zones de l'image dans lesquelles le panneau de 60 km/h est identifié avec un score supérieur à un certain seuil, afin d'éliminer les faux panneaux. Le résultat final est représenté sur la (Figure III.9)



Figure III.9 Le résultat final

Exemple 2 :

Dans un deuxième exemple nous allons utiliser une image plus complexe (Figure III.10.) Le programme suit les mêmes étapes Comme on l'a expliqué précédemment. Dans le présent exemple l'objective est d'identifier le panneau AUTRES DANGERS identifier par le numéro 18 dans les résultats de classification.



Figure III.10 La première image d'entrée

Le BNN commence par identifier les différents panneaux similaires au panneau recherché (Figure III.11.)



Figure III.11 Chercher le panneau dans l'image

Le résultat final est représenté sur la (Figure III.12)



Figure III.12 Le résultat final

III.5.2 Discussion :

Les résultats obtenus montrent que dans les deux exemples l'algorithme réussir à identifier avec succès les déférents panneaux de signalisation recherché avec un temps qui se diffère sel on la complexité de l'image d'entrée ainsi que le type de panneau recherché. Dans le premier exemple un temps de 32742.00 µs avec un taux de classification de 2870.93 images par secon de, alors que pour le deuxième cas un temps de 48763.00 µs et un taux de classification de 29 32.55 images par seconde.

III.6 Conclusion

La conception de systèmes embarqués nécessitant une réponse en temps réal, nous avons implémenté une application comportant une itération d'opérations qui généralement, dans des systèmes classiques (software) s'exécute en plusieurs cycles de temps et donc handicape les cas temps réel.

L'application choisie concerne une implémentation d'un réseau de neurones artificiel dans une plateforme reconfigurable (FPGA).

Ce réseau est de type Bayésiens c'est le plus favorable dans le domaine de reconnaissance d'image grâce à ses avantages par rapport à d'autre techniques.

Le langage python, a été utilisé pour la modélisation et la synthèse des différentes unités et modules du réseau de neurones artificiel sur un environnement de développement utilisant le Vivado de Xilinx.

Les résultats obtenus par les tests pratique sur la carte de développement Pynq, montre que le temps nécessaire pour la reconnaissance des panneaux de signalisation est très négligeable da ns le cas d'une implémentation hardware sur la zone reconfigurable de l'FPGA. Un avantage pour l'application des FPGA dans le domaine de l'apprentissage en profondeur.

CONCLUSION GENERALE

L'objectif du projet a été de proposer un système a base de réseau neurones artificielle implémenté sur une puce FPGA intégrée dans la carte PYNQ-Z1 ; et ce dans le but d'identifier des images spécifiques dans une image ou un ensemble d'images, en utilisant l'proche Bayésienne et cela en fonction du programme BNN et de la carte PYNQ-Z1, en exploitant les avantages offerts par les réseaux Bayésien.

Le langage utilisé est le python, c'est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
Nous avons eu l'occasion de travailler avec une carte récente, innovante et performante, la carte PYNQ-Z1 qui est la plate-forme matérielle du framework open-source PYNQ. Dont celle-ci est munie est effectivement à la pointe de la technologie. Nous avons eu l'occasion de mettre en œuvre un programme embarqué sur PYNQ-Z1, et qui nous a permis de réinvestir de nombreuses notions. Nous avons pu observer les différentes étapes de configuration et implémentation du système de reconnaissance d'images embarqué sur la PYNQ-Z1, par la génération des différents fichiers nécessaires.
Ce projet nous a beaucoup apporté sur la recherche d'informations techniques, ainsi la mise en œuvre d'un système embarqué sur PYNQ, car les démarches que nous avons effectuées peuvent être vues comme proches de la réalité de l'univers professionnel. Il est avéré que les réseaux Bayésien ont des bons résultats dans le domaine du traitement d'image.

BIBILIOGRAPHIQUE

[1]:D.Deckers, Composants programmables, Cours en Electronique appliquée, Haute Ecole de la Communauté française en Hainaut Unité Electronique, 2008-2009

[2]:S.Abdelhakim, implémentation sur FPGA d'un algorithme de bruitage, thèse de doctorat, Université badji mokhtar-annaba, 2013

[3]:C. Trabelsi, Contrôle matériel des systèmes partiellement reconfigurables sur FPGA : de la modélisation à l'implémentation, thèse de doctorat, Université des Sciences et technologies de Lille, 03 juillet 2013

[4]:H.Abbad, Méthodologie de développement et d'implantation sur puce FPGA d'algorithme de commande, Thèse de Magister en Electronique, Université Mohamed Khider – Biskra, Soutenu publiquement le 29/06/2016

[5]:L.Lelong, Architecture SoC-FPGA pour la mesure temps réel par traitement d'image, Conception d'un système embarqué, thèse de doctorat, Spécialité : électronique , l'universite jean monnet de saint-etienne , Soutenue le 7 /12/2005

[6]:M.Khitas, Implantation d'application de traitement de signal sur système mono-puce reconfigurable 'SoPc', Thèse de Magister, Université Ferhat Abbas –setif-1-, 2014

[7]: R.benmouhoub, méthodologies de conception pour multiprocesseurs sur circuits logiques programmables, thèse de doctorat, université paris-sud xi, Soutenue le 07 Mai 2007

[8]: DIGILENT, PYNQ- Z1 Board Reference Manual, 1300 Henley Court Pullman, WA 99163, 509.334.6306, 13/04/2017.

[9]: https://reference.digilentinc.com/

[10]: www.pynq.io

[11]:C.Touzet, les reseaux de neurones artificiels introduction au connexionnisme, Juillet 1992

[12]:G. dreyfus, les réseaux de neurones, École Supérieure de Physique et de Chimie Industrielles de la Ville de Paris (ESPCI), septembre 1998

[13] :C.Aymen , Réseaux de neurones, SVM et approches locales pour la prévision des séries temporelles, thèse de doctorat , université françoisrabelais de tours, École Doctorale miptis, soutenue le : 17 juillet 2013

[14]:P.wira , Réseaux de neurones artificiels : architectures et applications ,Université de haute Alsace , laboratoire MIPS , Avril 2009

[15]:B.Elmahdi, réseaux de neurones artificiels appliqués à la méthode électromagnétique transitoire infini tem, mai 2011

[16]:G.Bernard, application de réseaux de neurones artificiels a la reconnaissance automatique de caractères manuscrits, thèse de doctorat, Faculté Polytechnique de Mons, Année académique 95-96

[17]:P.David, Simulation comportementale par réseau de neurones et apprentissage par algorithme génétique, 2002-2003

[18]:A .JARRAYA, Nouvelles paramétrisations de réseaux Bayésiens et leur estimation implicite, thèse de doctorat , Université de Nantes Angers Le Mans , Soutenue le 26 octobre 2013

[19]:C. Chirca , Un modèle de réseau bayésien pour la gestion des risques dans la gouvernance d'un centre de services informatiques, Université de Sherbrooke, Canada, mai 2014

[20]:M.Embarki , Etude comparative entre réseaux bayésien et autre méthodes de classification appliqué sur une base cardiologue , thèse de Master en Informatique , Université Abou Bakr Belkaid– Tlemcen , Année universitaire :2011-2012

[21]:http://jupyter.org/

[22]:C.Bob , Introduction à Python 3, version 2.71828.

[23]:M.Vikram, Introduction à la programmation en langage Python, Université Paris-Sud, Année 2015-2016.

[24]:Umuroglu, Yaman , Fraser, Nicholas J, Gambardella, Giulio , Blott, Michaela , Leong, Philip , Jahre, Magnus et Vissers, FINN: A Framework for Fast, ScalableBinarized Neural Network Inference , 2017