République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École Nationale Polytechnique
المدرسة الوطنية المتعددة التقنيات

Département d'Électronique

Mémoire de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Implémentation sur FPGA de réseaux de neurones artificiels : application à la classification d'arythmies cardiaques

Réalisé par :
Mme. MELLOUK Zineb
Mme. FADEL Ouiem

Encadré par :
Dr. LANI Fatiha
Co-encadré par :
Pr. ADNANE Mourad

Soutenu le 26 juin 2023, Devant le jury composé de :

Pr. Adel BELOUCHRANI : ENP - Président
Dr. Mustapha DJEDDOU : ENP - Examinateur

ENP 2023

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École Nationale Polytechnique
المدرسة الوطنية المتعددة التقنيات

Département d'Électronique

Mémoire de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Implémentation sur FPGA de réseaux de neurones artificiels : application à la classification d'arythmies cardiaques

Réalisé par :
Mme. MELLOUK Zineb
Mme. FADEL Ouiem

Encadré par :
Dr. LANI Fatiha
Co-encadré par :
Pr. ADNANE Mourad

Soutenu le 26 juin 2023, Devant le jury composé de :

Pr. Adel BELOUCHRANI : ENP - Président
Dr. Mustapha DJEDDOU : ENP - Examinateur

ENP 2023

Dédicace

“

À chère mère précieuse, tu es l'incarnation de la bonté et mon pilier de soutien. Ce travail est dédié à toi, en signe de mon amour profond. Que Dieu te préserve en te comblant de santé, de longévité et de bonheur.
À mon cher père, je veux exprimer tout mon amour, mon respect et ma reconnaissance pour les sacrifices que tu as consentis pour mon éducation et mon bien-être. Ce travail est dédié à ta mémoire, ta présence me manque profondément. Tes enseignements et ton amour demeureront à jamais gravés dans mon cœur.
À mon frère Zinddine, à mes sœurs Samar et Sara, ainsi qu'à toute ma famille ,je vous témoigne tout mon amour, mon affection et ma gratitude.
À mes chères amies Vanessa, Nyhed , Dounia ,Thanina , Hiba ,Khadija ,Sabrina ainsi qu'à tous mes amis polytechniciens et camarades électroniciens(ELN Match) je vous exprime ma reconnaissance et l'amitié qui nous unit . Que celles et ceux que j'aurais oubliés ici me pardonnent.
Enfin, je remercie toute personne qui m'a apporté son aide, même minime.

”

Ouiem

This work is dedicated to :

“

***To my beloved parents**, their boundless love, unyielding belief in my abilities, and countless sacrifices have been the foundation upon which my success stands. Their unwavering support and constant encouragement have propelled me forward, instilling in me the determination to overcome obstacles and pursue my dreams fearlessly.*

***To my incredible sisters**, Loubna,Ikram,Rabab, for being my confidantes and a source of inspiration.*

***To my little boy, Zayn**, you are the light of my life. Your innocent smile and boundless love have motivated me to overcome challenges.*

***To my dear friends**, thank you for your unwavering friendship and for always standing by my side. Your encouragement and words of wisdom have meant the world to me.*

***To Eln Match**, my second family.*

***To all the remarkable individuals**, who have stood by me during my darkest times, your presence and encouragement have provided me with strength and resilience. I am deeply grateful for your kindness and support.*

Thank you for believing in me and for being a part of my journey.

”

Zineb

Remerciements

Ce mémoire marque une étape capitale de notre parcours, symbolisant la clôture d'un long cheminement éducatif et l'amorce de nouvelles perspectives. Il est le fruit de notre persévérance et de notre assiduité, mais il doit également une reconnaissance profonde au soutien indéfectible de notre entourage. Ainsi, nous souhaitons exprimer notre gratitude sincère envers toutes les personnes qui ont apporté leur contribution, de manière directe ou indirecte, à la réalisation de ce travail.

Nous tenons tout particulièrement à remercier nos parents, qui ont été présents dès les premiers pas de notre parcours, créant les conditions propices à notre développement intellectuel et personnel. Nous souhaitons également adresser nos remerciements chaleureux à nos amis et camarades de classe**(ELN match)**, dont l'appui et l'entraide ont été inestimables tout au long de ce projet.

Une reconnaissance particulière est également due à nos encadrants,**le Professeur ADNANE Mourad** et **Mme. LANI Fatiha** , pour leur précieuse orientation tout au long de cette expérience. Leur patience, leur motivation et leur dévouement sans faille ont contribué à faire de ce travail une réalisation de qualité. Leur disponibilité et leurs conseils avisés ont été d'une valeur inestimable.

Enfin, nous tenons à exprimer notre profonde gratitude envers l'ensemble de nos enseignants du département d'électronique, qui ont joué un rôle primordial dans notre formation tout au long de ces trois années d'études.

Nous sommes reconnaissants envers toutes ces personnes qui ont joué un rôle déterminant dans notre réussite, et qui ont été des soutiens solides tout au long de ce périple éducatif.

Ouiem & Zineb

ملخص

أدت التطورات في الإلكترونيات الدقيقة إلى تحسين التشخيص التلقائي لمشاكل القلب بشكل كبير.حيث لعبت الأنظمة الذكية المضمنة، مثل ،FPGA دورًا حاسمًا في تمكين التكامل السريع وإعادة التشكيل لتطوير النماذج الأولية للأجهزة الطبية.

في مشروعنا، طورنا خوارزمية تستند إلى نموذج شبكة عصبية تلافيفية (CNN) مصمم خصيصًا لتصنيف عدم انتظام ضربات القلب باستخدام بيانات تخطيط القلب. تم .تدريب النموذج على تصنيف إشارات تخطيط القلب إلى خمس فئات من دقات القلب. لتحسين الأداء، قمنا بتنفيذ هذه الخوارزمية على منصة Pynq-Z1.

من خلال تشغيل النموذج بنجاح على منصة Pynq-Z1 , يساهم مشروعنا في تحسين التصنيف المبكر لمشاكل القلب، والتي لها تأثير كبير على نتائج المرضى ولديها القدرة على إنقاذ الأرواح.

كلمات مفتاحية : تعلم الآلة ، التصنيف ، عدم انتظام ضربات , ECG ، CNN ، PYNQZ1 FPGA.

Abstract

Advancements in microelectronics have significantly improved the automatic diagnosis of cardiac issues. Intelligent embedded systems, such as FPGA, have played a crucial role in enabling fast integration and reconfiguration for the development of medical device prototypes.

In our project, we have developed an algorithm based on a convolutional neural network (CNN) model specifically designed to classify cardiac arrhythmias using ECG data. The model has been trained to classify ECG signals into five categories of heartbeats. To enhance performance, we have implemented this algorithm on the FPGA platform, PYNQ Z1

By successfully running the model on the Pynq-Z1 FPGA platform, our project contributes to improving the early classification of cardiac problems, which has a significant impact on patient outcomes and has the potential to save lives.

Keywords : Classification, Cardiac Arrhythmias, ECG, CNN, FPGA, PYNQ Z1.

Résumé

Les avancées en microélectronique ont considérablement amélioré le diagnostic automatique des problèmes cardiaques. Les systèmes embarqués intelligents, tels que les FPGA, ont joué un rôle essentiel en permettant une intégration et une reconfiguration rapides pour le développement de prototypes de dispositifs médicaux.

Dans le cadre de notre projet, nous avons développé un algorithme basé sur un modèle de réseau de neurones convolutifs (CNN) spécifiquement conçu pour classer les arythmies cardiaques à partir de données d'ECG. Le modèle a été entraîné à classer les signaux ECG en cinq catégories de battements cardiaques. Pour améliorer les performances, nous avons réalisé l'implémentation de cet algorithme sur la plateforme FPGA PYNQ Z1.

En exécutant avec succès le modèle sur la plateforme FPGA Pynq-Z1, notre projet contribue à améliorer la classification précoce des problèmes cardiaques, ce qui a un impact significatif sur les résultats des patients et peut potentiellement sauver des vies.

Mots clés : Classification , Arythmies cardiaques , ECG , CNN , FPGA , PYNQ Z1.

Table des matières

Table des figures

Liste des tableaux

Liste des sigles et acronymes

ACP	*Analyse en Composantes Principales*
ARM	*Advanced RISC Machines*
ASIC	*Application-specific Integrated Circuit*
AXI	*Advanced Extensible Interface*
CNN	*Convolution Neural Network*
CLB	*Configurable Logique Bloc*
CMOS	*Complementary metal-oxide-semiconductor*
CPU	*Central processing unit*
DSP	*Digital Signal Processor*
E/S	*entré/sortie*
FPGA	*Field-programmable Gate Array*
GPU	*Graphics processing unit*
HLS	*High Level Synthesis*
IP	*Intellectual Property*
KNN	*K-Nearest Neighbors*
MLP	*Multi Layer Perceptron*
PE	*Process Element*
PS	*processing system*

PYNQ *Python Productivity for Zynq*

RAM *Random Access Memory*

ReLU *Rectified Linear Unit*

RNA *Réseaux de Neurones Artificiels*

SoC *System on a Chip*

SVM *Support Vector Machine*

TPU *Tensor Processing Unit*

USB *Universal Serial Bus*

VHDL *Very High-Speed Integrated Circuit Hardware Description Language*

Introduction générale

Contexte

Les maladies cardiovasculaires sont une préoccupation majeure de santé publique à l'échelle mondiale, étant l'une des principales causes de décès. Cependant, grâce à une surveillance attentive des signes vitaux, il est possible de réduire significativement ces chiffres alarmants. Ces dernières années, les dispositifs portables de surveillance en temps réel des battements cardiaques ont connu une croissance exponentielle, offrant une opportunité précieuse pour améliorer l'état actuel de la technologie des électrocardiogrammes (ECG) portables. L'une des avancées majeures dans ce domaine est la classification des arythmies cardiaques, qui permet d'identifier et de diagnostiquer différents types de troubles du rythme cardiaque. La classification précise des arythmies cardiaques à l'aide de dispositifs portables peut contribuer à une surveillance plus efficace de la santé cardiovasculaire, permettant une détection précoce des anomalies et une intervention rapide pour prévenir les complications potentielles(Saptono 2011) .

Différentes méthodes ont été utilisées pour classer les battements cardiaques dans les ECG, avec des mesures d'exactitude et de sensibilité rapportées pour chaque méthode. Les CNN ont obtenu des performances élevées, avec une exactitude de 94,90% (Kiranyaz et al. 2016)pour une étude, 93,18% et 89,07%(Acharya et al. 2017) pour d'autres configurations spécifiques. Une autre étude utilisant un CNN a atteint une exactitude de 99,05%(Jun et al. 2018). Les SVM, combinés avec une régression logistique, ont obtenu une exactitude de 98,84%(Mondéjar-Guerra et al. 2019). Enfin, l'utilisation d'un DNN a permis d'atteindre une exactitude de 99%(A. H. Ribeiro et al. 2020). Ces résultats démontrent l'efficacité de ces méthodes dans la classification des battements cardiaques dans les ECG.

Problématique

Les dispositifs portables conçus pour surveiller en continu les signaux d'électrocardiogramme (ECG) offrent de nombreux avantages, tels que la commodité, la portabilité et l'intégration discrète dans la vie quotidienne. Cependant, ces dispositifs sont confrontés à deux défis majeurs : des vitesses de traitement lentes et une consommation élevée d'énergie, ce qui limite leur application pratique et leur efficacité dans la surveillance cardiaque en temps réel. Les vitesses de traitement lentes peuvent entraîner des retards dans l'identification des lectures d'ECG critiques nécessitant une attention immédiate, compro-

mettant ainsi une intervention rapide et efficace pour les problèmes cardiaques. De plus, la consommation élevée d'énergie des dispositifs ECG portables constitue un défi majeur en raison de leur capacité limitée de batterie et de leur taille compacte, ce qui limite leur capacité à assurer une surveillance continue et à fournir des avertissements efficaces pour les événements cardiaques. Pour répondre à ces problématiques, nous proposons le développement d'un classifieur ECG spécifiquement optimisé pour les réseaux de portes programmables sur FPGA aux ressources limitées. Cette approche vise à combiner des capacités de traitement rapides et une gestion efficace de l'énergie, améliorant ainsi la détection des événements cardiaques et la surveillance de l'utilisateur.

Objectifs

Dans cette étude, nous avons deux objectifs principaux visant à améliorer les capacités des dispositifs ECG portables et à améliorer la surveillance de la santé cardiovasculaire. Tout d'abord, nous développons un classifieur ECG implanté sur la plate-forme FPGA (Field-Programmable Gate Array) Xilinx Pynq-Z1. L'utilisation de la technologie FPGA nous permet de réduire considérablement la latence du modèle de classification ECG, ce qui permet une détection rapide d'événements cardiaques potentiellement mortels. Cette détection et réponse rapides peuvent améliorer les résultats des patients et sauver des vies.

Ensuite, notre objectif est de limiter l'utilisation des ressources du classifieur ECG. Cette approche permet non seulement d'intégrer le classifieur ECG sur des cartes de taille réduite, mais aussi de réduire la consommation dynamique d'énergie de l'appareil. Un classifieur ECG léger rend l'appareil plus convivial pour les patients (plus facile à porter) et lui permet de fonctionner pendant une plus longue durée. Pour atteindre ces objectifs, nous avons entraîné un réseau neuronal convolutif (CNN) à classer les signaux ECG en cinq catégories de battements cardiaques, y compris les battements cardiaques normaux. Le modèle a été entraîné à l'aide de la base de données d'arythmie MIT-BIH.Après l'entraînement, le modèle a été exporté en C++ et synthétisé à l'aide de Vitis (Bai et al. 2018). Notre modèle fonctionne avec succès sur la carte FPGA Pynq-Z1.

Organisation du mémoire

Ce mémoire est organisé en quatres chapitres :

Le premier chapitre "**Fondements théoriques**" introduit les concepts fondamentaux des réseaux de neurones artificiels et explore leur utilisation dans différents domaines. Il présente également les circuits FPGA et leur rôle dans l'implémentation des réseaux de neurones, ainsi que les outils logiciels utilisés dans le développement.

Le deuxième chapitre "**État de l'art**" offre un aperçu des connaissances actuelles dans le domaine des arythmies cardiaques. Il explore le système cardiovasculaire, les électrocardiogrammes (ECG) et les différentes méthodes de détection et de classification des arythmies cardiaques. De plus, il examine l'état de l'art en ce qui concerne l'implémen-

tation de réseaux de neurones sur FPGA pour la classification des arythmies cardiaques, en mettant en évidence les différentes approches et les résultats obtenus.

Le troisième chapitre "**Conception et implémentation** " se concentre sur la conception et l'implémentation du modèle CNN en utilisant différents langages de programmation et outils. Il décrit la conception en Python du modèle CNN, en détaillant la base de données utilisée, l'algorithme choisi, les méthodes d'optimisation et le choix de Pytorch. De plus, il présente différentes approches d'implémentation, y compris l'implémentation en C/C++, l'utilisation de VITIS HLS et l'implémentation sur la carte PYNQ Z1.

Le quatrième chapitre " **Résultats et évaluation**" présente les résultats obtenus à partir des différentes approches d'implémentation. Il évalue les performances du modèle CNN en utilisant différentes mesures, et compare les résultats avec d'autres modèles de classification tels que SVM, KNN et la régression logistique. De plus, il analyse les résultats de la synthèse dans Vitis HLS et de l'implémentation sur Vivado, ainsi que les résultats de l'implémentation sur la carte PYNQ Z1.

En conclusion, ce mémoire aboutit à des conclusions significatives et ouvre la voie à plusieurs perspectives prometteuses.

Chapitre 1

Fondements théoriques

1.1 Introduction

Le chapitre "Fondements théoriques" constitue une étape cruciale dans notre exploration de l'implémentation sur FPGA des réseaux de neurones artificiels pour la classification des arythmies cardiaques. Cette section présente les principaux sujets abordés, à savoir les réseaux de neurones artificiels, l'utilisation des FPGA dans leur implémentation, ainsi que les outils logiciels de développement associés.

Nous commençons par explorer en profondeur les principes fondamentaux des réseaux de neurones artificiels, notamment leurs différentes architectures et leur fonctionnement interne. Nous examinons également les techniques d'apprentissage utilisées pour entraîner ces réseaux et améliorer leurs performances.

Ensuite, nous nous concentrons sur les FPGA, des dispositifs offrant flexibilité et puissance de traitement parallèle, idéaux pour l'implémentation efficace des réseaux de neurones. Nous discutons des avantages et des défis liés à l'utilisation des FPGA, en mettant en évidence leurs caractéristiques spécifiques.

Enfin, nous présentons les outils logiciels de développement qui facilitent la conception et la mise en œuvre de réseaux de neurones sur FPGA. Nous mettons en évidence l'importance de ces outils conviviaux et des bibliothèques optimisées pour la réalisation pratique de nos réseaux de neurones sur FPGA.

1.2 Réseaux de neurones artificiels

Les neurones sont des cellules responsables de l'évolution de tout le système nerveux d'un être humain. Cette invention a été la première description de la structure et de l'organisation des neurones biologiques par Camillio Golgi et Santiago Ramdon Y. Cajal en 1906, ce qui leur a valu le prix Nobel de médecine. En 1943, MCCulloch et Walter Pitts ont mené une seconde étude sur les neurones artificiels, qui a fourni un modèle mathématique simplifié des neurones biologiques (NATURAL-SOLUTIONS.WORLD April 9, 2018) .

1.2.1 Introduction aux réseaux de neurones

Le neurone biologique

Un neurone biologique est une cellule nerveuse fondamentale du système nerveux. Cette cellule est adaptée à une fonction spécifique dans le traitement des signaux électriques. aussi connu sous le nom de “soma”, est le cerveau de l'opération du neuronec. Il reçoit et traite les informations qui lui parviennent, puis renvoie le résultat sous forme de signaux électriques à l'entrée des autres neurones via l'axone. Les axones relient les neurones entre eux et sont primordiaux dans le comportement logique de l'ensemble. Le neurone biologique est formé de plus de dendrites, qui sont des branches chargées de recevoir des informations de l'extérieur et de diffuser ces informations au corps cellulaire. La

synapse d'un neurone reçoit des informations d'autres neurones via les axones, assurant l'échange d'informations entre les neurones.(*Neurone biologique* 2021)

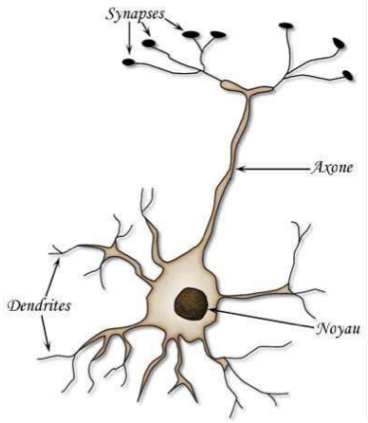


FIG. 1.1 : Modèle d'un neurone biologique (NEURONE 2015)

Le neurone artificiel

Un neurone artificiel est un élément de traitement de l'information qui effectue une fonction mathématique bornée et non linéaire. Les coefficients ou poids qui lui sont attribués influencent sa sortie qui est obtenue en fonction de ses entrées. La représentation visuelle du neurone est souvent réalisée sous forme d'un diagramme graphique.

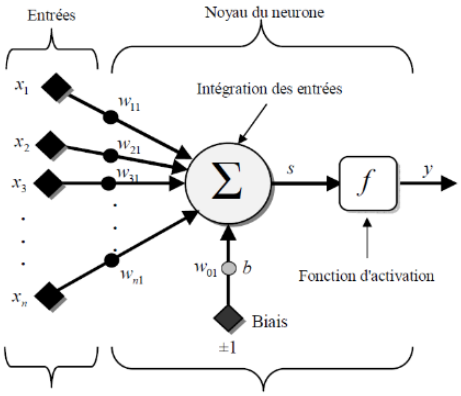


FIG. 1.2 : Modèle d'un neurone artificiel(NATURAL-SOLUTIONS.WORLD April 9, 2018)

Le modèle de neurone formel, qui a été présenté en 1943 par W.M. Culloch et W. Pitts, est basé sur les recherches en neurobiologie. (NATURAL-SOLUTIONS.WORLD April 9, 2018)
Ce modèle est constitué de plusieurs éléments, tels que

- les vecteurs d'entrées x_i, qui peuvent provenir de sorties d'autres neurones ou de stimuli sensoriels tels que la vision ou l'ouïe.

- Les poids synaptiques w_{ij} sont utilisés pour pondérer les entrées et sont ajustables par apprentissage. Les poids positifs ($w_{ij} > 0$) correspondent à des synapses excitatrices, tandis que les poids négatifs ($w_{ij} < 0$) correspondent à des synapses inhibitrices.
- Le biais est souvent utilisé comme une entrée supplémentaire prenant les valeurs -1 ou +1 pour ajuster les poids et le seuil de déclenchement du neurone pendant l'apprentissage.
- Le noyau calcule la sortie du neurone à l'aide d'une fonction d'activation non linéaire pour offrir une plus grande flexibilité d'apprentissage.

Le neurone élaboré par McCulloch et Pitts utilise une fonction d'activation f binaire à seuil qui produit une sortie binaire de 0 ou 1, dérivée de la fonction de transfert, qui est transmise aux neurones suivants.

Nom de la fonction	Relation entrée/sortie	Icône
Seuil	$a = 0 \quad \text{si } n < 0$ $a = 1 \quad \text{si } n \geq 0$	
Seuil symétrique	$a = -1 \quad \text{si } n < 0$ $a = 1 \quad \text{si } n \geq 0$	
Linéaire	$a = n$	
Linéaire positive	$a = 0 \quad \text{si } n < 0$ $a = n \quad \text{si } n \geq 0$	
Linéaire saturée	$a = 0 \quad \text{si } n < 0$ $a = n \quad \text{si } 0 \leq n \leq 1$ $a = 1 \quad \text{si } n > 1$	
Linéaire saturée symétrique	$a = -1 \quad \text{si } n < -1$ $a = n \quad \text{si } -1 \leq n \leq 1$ $a = 1 \quad \text{si } n > 1$	
Sigmoïde	$a = \frac{1}{1+\exp^{-n}}$	
Tangente hyperbolique	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$	
Compétitive	$a = 1 \quad \text{si } n \text{ maximum}$ $a = 0 \quad \text{autrement}$	

TAB. 1.1 : Différentes fonctions d'activations

Les trois fonctions les plus usuelles sont les fonctions "seuil", "linéaire" et "sigmoïde".

Similitude entre le neurone formel et biologique

Nous pouvons récapituler cette modélisation à travers le tableau 1.2,ce qui nous permettra d'avoir une vision claire de la transformation effectuée entre le neurone biologique et le neurone formel

Neurone biologique	Neurone artificiel
Synapses	poids de la connexion (entrées ou sorties)
Axones	Sorties
Dendrites	Entrées
Noyau	Fonction d'activation

Tab. 1.2 : Similitude entre le neurone formel et biologique

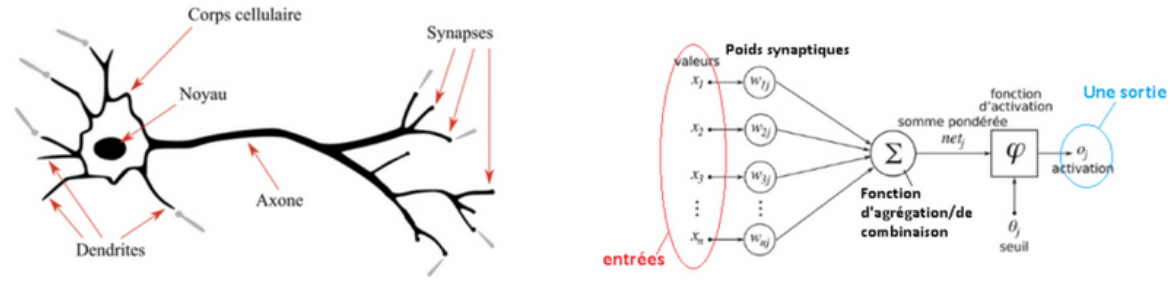


Fig. 1.3 : Transposition mathématique/informatique d'un neurone biologique (Maurice 2023)

1.2.2 Apprentissage des réseaux de neurones

Consiste en une procédure d'optimisation visant à ajuster les paramètres du réseau de manière à minimiser une fonction de coût prédéfinie sur un ensemble de données d'entraînement. Cette capacité d'apprentissage est la caractéristique distinctive des réseaux de neurones en tant qu'outil mathématique, et est réalisée grâce à un processus d'entraînement. L'apprentissage des réseaux de neurones peut être classé en deux catégories distinctes : l'apprentissage supervisé et l'apprentissage non supervisé.

l'apprentissage supervisé

Un professeur qui connait parfaitement la sortie désirée ou correcte, guide le réseau en lui apprenant à chaque étape le bon résultat. Donc l'apprentissage consiste à comparer le résultat obtenu entre les deux.

l'apprentissage non supervisé :

Le réseau modifie ses paramètres en tenant compte seulment des informations locale. Ces méthodes n'ont pas besion de sorties désirées préétablies.

1.2.3 Les types de réseaux de neurones

Les perceptrons multicouches (MLP)

Les perceptrons multicouches appartiennent à la famille des réseaux de neurones artificiels à propagation avant (feedforward). Constituant le réseau de neurones profond le plus élémentaire, les MLP sont composés d'une série de couches entièrement connectées. Les méthodes d'apprentissage automatique basées sur les MLP permettent actuellement de pallier l'exigence en puissance de calcul élevée requise par les architectures modernes d'apprentissage profond. Chaque nouvelle couche est une combinaison linéaire de toutes les sorties (entièrement connectées) de la couche précédente, suivie d'une fonction d'activation non linéaire.

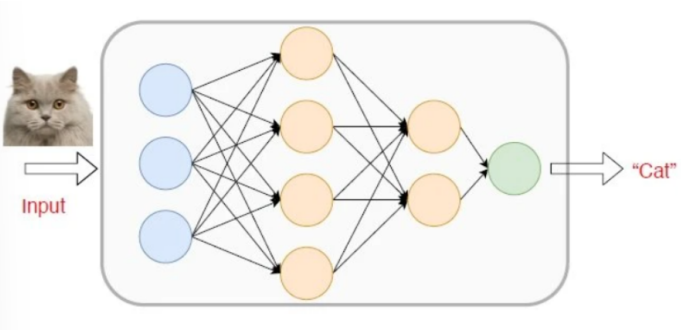


Fig. 1.4 : Concept de perceptrons multicouches (MLP) (Rossi 2023)

Les réseaux de neurones récurrents (RNN)

Les réseaux de neurones récurrents constituent une catégorie de réseaux de neurones artificiels qui utilisent des données séquentielles en entrée. Ils ont été développés pour résoudre le problème de séries temporelles de données d'entrées séquentielles. L'entrée d'un RNN se compose de l'entrée courante et des échantillons précédents. Par conséquent, les connexions entre les nœuds forment un graphe dirigé le long d'une séquence temporelle. En outre, chaque neurone dans un RNN possède une mémoire interne qui conserve les informations de calcul des échantillons précédents. Dans les RNN, chaque couche ultérieure est une collection de fonctions non linéaires de sommes pondérées des sorties et de l'état précédent. Ainsi, l'unité de base du RNN est appelée "cellule", et chaque cellule est composée de couches et d'une série de cellules qui permettent le traitement séquentiel des modèles de réseaux de neurones récurrents.

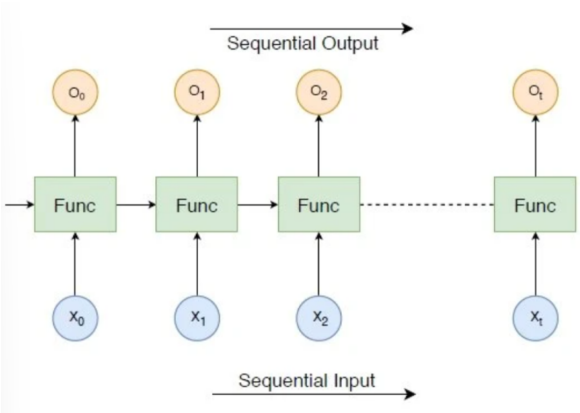


Fig. 1.5 : Concept de réseau neuronal récurrent (RNN) (Rossi 2023)

les réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutifs (CNN ou ConvNet) sont une classe de réseaux de neurones profonds utilisée principalement en vision par ordinateur. Ils permettent d'extraire automatiquement les caractéristiques d'une série d'images ou de vidéos du monde réel afin de réaliser une tâche spécifique telle que la classification d'images, l'authentification faciale ou la segmentation sémantique d'images. Contrairement aux couches entièrement connectées des MLP, les modèles CNN comportent une ou plusieurs couches de convolution qui extraient les caractéristiques simples de l'entrée en effectuant des opérations de convolution. Chaque couche est un ensemble de fonctions non linéaires de sommes pondérées à différentes coordonnées de sous-ensembles spatialement proches des sorties de la couche précédente, ce qui permet la réutilisation des poids. Les modèles d'apprentissage automatique CNN permettent de capturer la représentation de haut niveau des données d'entrée en appliquant divers filtres de convolution, ce qui les rend largement populaires dans les tâches de vision par ordinateur.

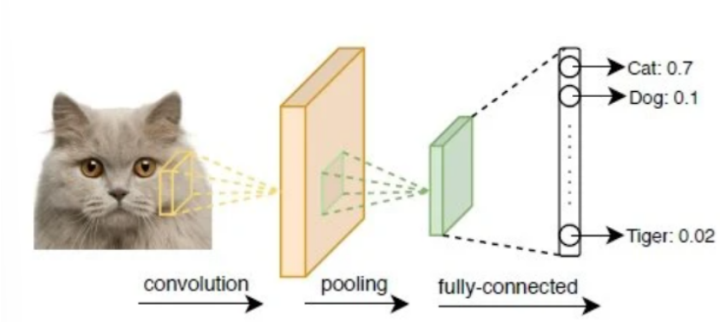


Fig. 1.6 : Concept de réseau neuronal à convolution (CNN) (Rossi 2023)

1.2.4 Domaines d'applications

Les réseaux de neurones artificiels trouvent aujourd'hui de nombreuses applications dans des domaines variés :

- **Domaine du traitement d'images :** reconnaissance de caractères et de signatures, compression d'images, reconnaissance de formes, cryptage, classification, etc.
- **Domaine du traitement du signal** : filtrage, classification, identification de sources, traitement de la parole, etc
- **Domaine du contrôle :** commande de processus, diagnostic, contrôle qualité, asservissement des robots, systèmes de guidage automatique des automobiles et des avions, etc.
- **Domaine de l'optimisation :** planification, allocation de ressources, gestion et finances, etc.
- **Domaine de la simulation :** simulation de vol, simulation de boîte noire, prévision météorologique, recopie de modèle, etc.
- **Domaine médical :** analyse des signaux EEG, ECG, prothèses, analyse du cancer, etc. (Izeboudjen 1999)(Y.Djeriri 2017)

Ces exemples illustrent la diversité des applications des réseaux de neurones artificiels dans des domaines aussi variés que l'imagerie, le contrôle, la défense, l'optimisation, la simulation et la médecine.

1.3 FPGA et leur utilisation dans l'implémentation de réseaux de neurones

1.3.1 Les circuits FPGA

Les FPGA, ou "réseaux logiques programmables" (Field Programmable Gate Arrays), sont des composants entièrement reconfigurables, ce qui leur confère la capacité d'être reprogrammés à volonté pour accélérer de manière significative certaines phases de calcul. L'un des avantages majeurs de ce type de circuit réside dans sa grande flexibilité, qui permet de le réutiliser facilement dans différents algorithmes en un laps de temps très court. Les avancées technologiques dans ce domaine ont permis de concevoir des composants de plus en plus rapides et intégrés, ouvrant ainsi la voie à la programmation d'applications d'envergure.

Cette technologie offre la possibilité d'implanter un large éventail d'applications et représente une solution d'implantation matérielle à faible coût pour les entreprises de taille modeste. En effet, le coût de développement d'un circuit intégré spécifique représente un investissement considérable pour ces sociétés. Grâce à sa reconfiguration aisée et illimitée, un circuit FPGA (Field Programmable Gate Array) est capable d'implémenter des systèmes et des circuits complexes dans divers domaines d'applications (Bailey 2011)

1.3.2 Architecture d'un FPGA

L'architecture précise d'un FPGA peut varier d'un fournisseur à l'autre. Dans cette section, nous présentons une structure FPGA standard (Figure 1.11) qui comprend les éléments suivants (Bailey 2011) :

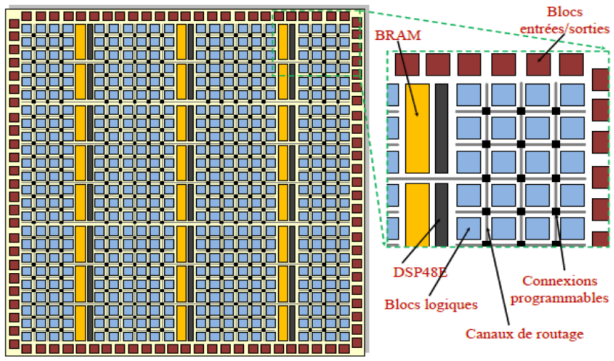


Fig. 1.7 : Architecture d'un FPGA (Bailey 2011)

- **Blocs logiques programmables :** Ces blocs logiques sont composés de milliers à des millions de transistors. Ils implémentent les fonctions logiques nécessaires à la conception et se composent de modules logiques tels que des combinaisons de transistors, des tables de correspondance (look-up tables ou LUT) et des éléments de contrôle et de transmission (bascules et multiplexeurs).

- **Blocs d'E/S programmables :** Ils servent de lien entre les blocs logiques et les composants externes via des broches d'interface.

- **Ressources d'interconnexion programmables** Ce sont des interconnexions électriquement programmables disposées de manière verticale et horizontale pour établir les chemins de routage des blocs logiques programmables. Ces chemins de routage sont composés de segments de fils de longueurs variables, qui peuvent être interconnectés à l'aide de commutateurs électriquement programmables. La densité du FPGA dépend du nombre de segments utilisés pour les chemins de routage (Messaoudi 2012).

- **Circuit d'horloge :** Dans un circuit d'horloge, des blocs d'E/S spéciaux équipés de tampons d'horloge à entraînement élevé, appelés pilotes d'horloge, sont répartis sur la puce. Ces tampons se connectent aux bornes d'entrée d'horloge et génèrent les signaux d'horloge qui sont transmis via des lignes d'horloge globales. Ces lignes d'horloge sont conçues pour minimiser les temps de montée et de propagation. Il est important de souligner que la conception synchrone est indispensable dans les FPGA, car seule la synchronisation des signaux d'horloge garantit l'absence de décalage et de retard.

- **Cœurs de processeur :** Les cœurs de processeur sont des composants essentiels fréquemment disponibles sous forme de cœurs IP (propriété intellectuelle) ou de noyaux intégrés. Ils sont principalement conçus pour les systèmes embarqués, car ces derniers sont généralement programmables par nature. Ainsi, les processeurs embarqués sont particulièrement adaptés aux applications des systèmes embarqués.

- **Cœurs DSP :** Les processeurs de signaux numériques (DSP) sont un autre type courant de cœur proposé sous forme de cœur IP ou de cœur intégré. Ils sont spécifiquement conçus pour traiter des signaux analogiques. Les DSP sont largement utilisés pour des applications telles que le filtrage et la compression des signaux vidéo ou audio. Dans le contexte de la conception d'un réseau CNN basé sur des FPGA, il est crucial de déterminer le type de DSP à utiliser en fonction des tâches à effectuer.

1.3.3 Consommation D'énergie

En ce qui concerne la consommation d'énergie, les FPGA consomment plus d'énergie que les ASIC en raison du grand nombre de transistors nécessaires pour la programmation des composants. Les FPGA à faible consommation d'énergie représentent un défi pour les fabricants. Trois types de consommation d'énergie sont pris en compte pour concevoir un FPGA efficace :

- **Puissance statique :** C'est la puissance consommée par les pertes des transistors lorsqu'aucun signal ne circule.

- **Puissance dynamique :** Il s'agit de la puissance consommée par les changements d'état des transistors pendant le fonctionnement du circuit.

- **Puissance d'entrée/sortie :** C'est la puissance consommée par les portes d'entrée/sortie, y compris les entrées/sorties générales et les émetteurs-récepteurs série à haute vitesse

1.3.4 Les Familles des FPGA

Dans la section précédente, une analyse générale des caractéristiques architecturales des FPGA a été effectuée. Dans cette section, nous examinerons plus en détail les caractéristiques spécifiques de certains dispositifs. Xilinx et Intel FPGA sont les deux principaux fabricants de FPGA en termes de part de marché, bien qu'il existe d'autres fournisseurs qui proposent également des FPGA. Nous présenterons les offres actuelles de chacun de ces fabricants de manière détaillée(Bailey 2011).

- **Xilinx :** Xilinx occupe une position de leader mondial en tant que fournisseur de technologies et de dispositifs programmables. L'entreprise se démarque en offrant bien plus que de la simple logique programmable, en permettant la programmabilité matérielle et logicielle. Elle intègre également des fonctionnalités de signaux mixtes analogiques et numériques, ainsi que des capacités avancées d'interconnexion programmable dans les circuits intégrés 3D monolithiques et multiples. Les produits Xilinx sont accompagnés d'un environnement de conception de nouvelle génération et d'une vaste bibliothèque de propriété intellectuelle (IP) pour répondre aux divers besoins de sa clientèle, allant de la logique programmable à l'intégration de systèmes programmables (Bailey 2011).

- **Zynq :** La famille Zynq-7000, qui repose sur l'architecture Xilinx SoC, regroupe des produits combinant un système de traitement (PS) puissant basé sur l'ARM Cortex-A9 MPCore et une logique programmable (PL) fournie par Xilinx, le tout intégré dans un seul dispositif. Ces appareils sont fabriqués en utilisant une technologie de pointe, le processus haute performance, basse consommation (HPL) de 28 nm et High-k Gate (HKMG). Le système PS est composé de processeurs ARM Cortex-A9 MPCore qui constituent le cœur, accompagnés d'une mémoire intégrée, d'interfaces de mémoire externes et d'un ensemble complet de périphériques d'E/S(Xilinx 2021).

  La famille Zynq-7000 offre la flexibilité et l'évolutivité d'un FPGA, combinant les performances, la puissance et la facilité d'utilisation généralement associées aux ASIC et aux ASSP. Les différents dispositifs de la famille Zynq-7000 permettent aux concepteurs de cibler des applications à la fois économiques et à haute performance à partir d'une plateforme unique, en utilisant des outils standard de l'industrie. Bien que tous les périphériques de la famille Zynq-7000 partagent le même système PS, les ressources PL et les E/S peuvent varier d'un dispositif à l'autre(Xilinx 2021).

  L'architecture Zynq-7000 facilite l'intégration pratique de la logique personnalisée dans la PL et du logiciel dans le PS. Cela permet de réaliser des fonctions système

uniques et différenciées. L'intégration du PS avec la PL offre des niveaux de performance inégalés par les solutions à base de deux puces (par exemple, un ASSP avec FPGA), grâce à une bande passante élevée des E/S, une absence de couplage et des contraintes de puissance limitées(Xilinx 2021).

1.3.5 Les overlays :

Le concept d'overlay est relativement récent et ne possède pas encore de définition précise et largement acceptée. Toutefois, on peut le définir de manière générale comme un design reconfigurable qui est mis en œuvre sur un FPGA, lequel est également reconfigurable. En d'autres termes, un overlay représente une couche d'abstraction matérielle intercalée entre l'application et le FPGA hôte, permettant de séparer l'application de ce dernier. Son rôle est d'offrir une interface simplifiée et cohérente pour l'application, tout en tirant parti de la flexibilité et des capacités de reconfiguration du FPGA pour optimiser les performances et la fonctionnalité (Bollengier 2018).

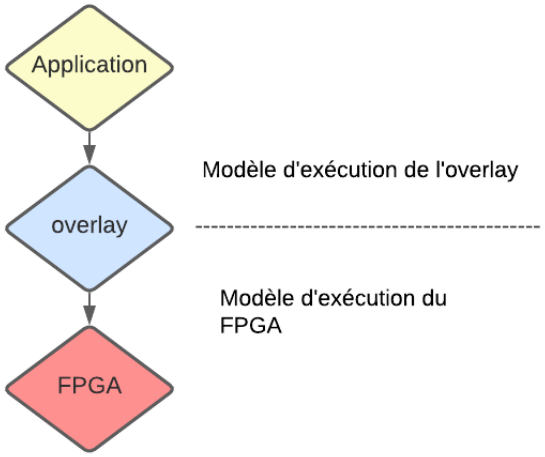


Fig. 1.8 : Modèle d'exécution d'overlay

La figure 1.8 met en évidence le fait que l'overlay introduit son propre modèle d'exécution, qui peut différer de manière significative de celui du FPGA sous-jacent. L'application n'a connaissance que du modèle d'exécution de l'overlay et est ainsi isolée du FPGA hôte. La granularité de l'overlay est déterminée par son modèle d'exécution. Les overlays offrent la possibilité d'explorer l'espace situé entre la facilité d'utilisation des processeurs et les performances des FPGAs. Par conséquent, les overlays sont des candidats prometteurs pour servir d'accélérateurs de calcul reconfigurables.

1.3.6 Comparaison des plates-formes matérielles pour l'inférence des CNN

Les réseaux de neurones convolutionnels (CNN) utilisent des opérations de multiplication et d'accumulation (MAC) avec des nombres à virgule flottante pour classifier les arythmies cardiaques. Les GPU sont souvent préférés pour l'entraînement et l'exécution de ces réseaux en raison de leur capacité à exploiter un parallélisme massif pour les calculs en virgule flottante. Toutefois, lors de l'inférence des réseaux neuronaux, moins de calculs sont nécessaires.

En plus des GPU, l'inférence peut être réalisée sur d'autres plates-formes matérielles telles que les CPU, les FPGA et les ASIC, y compris les TPU. Voici une brève description de ces plates-formes :

- **Les CPU** peuvent effectuer les calculs nécessaires à l'inférence. Bien qu'ils puissent utiliser des instructions SIMD (Single Instruction, Multiple Data) et des cœurs multiples pour exécuter plusieurs calculs en parallèle, leur vitesse de traitement est généralement inférieure par rapport aux autres alternatives.

- **Les GPU** sont des processeurs dotés de plusieurs cœurs conçus pour traiter des blocs de données en parallèle. Des frameworks comme CUDA et OpenGL peuvent être utilisés pour répartir les données sur les cœurs du GPU et accéder rapidement aux données en virgule flottante grâce à une mémoire à large bande passante. Cependant, les GPU consomment beaucoup d'énergie, atteignant parfois jusqu'à 250 W.

- **Les FPGA** offrent une architecture matérielle flexible car différentes unités de calcul peuvent être utilisées. Contrairement aux processeurs ou aux GPU qui sont limités à une seule architecture, les FPGA permettent une adaptation flexible aux avancées rapides de la recherche sur les réseaux neuronaux. Les couches du réseau peuvent être placées sur un FPGA et exécutées en parallèle pour une inférence efficace.

- **Les TPU** sont des ASIC spécialement conçus pour l'inférence des réseaux neuronaux. Ils offrent des performances améliorées par rapport aux GPU en termes de coûts. Les TPU sont optimisés pour effectuer des calculs sur des nombres en 8 bits, adaptés aux couches de base des réseaux neuronaux. Cependant, le rythme rapide du développement des réseaux neuronaux peut rendre la conception des TPU obsolète rapidement. De plus, la conception et la fabrication des TPU sont coûteuses par rapport aux autres alternative.

- **Les DSP** sont des processeurs spécialisés conçus pour traiter des signaux numériques en temps réel. Ils sont largement utilisés dans les applications de traitement du signal telles que la télécommunication, l'audio, la vidéo, la surveillance et les systèmes embarqués. Les DSP sont conçus pour exécuter efficacement des opérations mathématiques complexes et des algorithmes de traitement du signal.

1.3.7 Les systèmes SoC FPGA

Un système SoC FPGA combine un processeur ou un microcontrôleur avec un FPGA (Field Programmable Gate Array) sur une même puce, formant ainsi une plateforme intégrée. Le processeur est responsable des tâches de traitement général, tandis que le FPGA permet la personnalisation matérielle et l'accélération de fonctions spécifiques. Cette combinaison offre une solution complète pour le développement de systèmes embarqués, en tirant parti de la puissance de calcul logiciel du processeur et de la flexibilité matérielle du FPGA. Les systèmes SoC FPGA sont largement utilisés dans divers secteurs, tels que l'informatique embarquée, les télécommunications et l'automobile, pour leurs performances élevées, leur adaptabilité matérielle et leur capacité à réduire les délais de mise sur le marché(Melo et al. 2022).

1.3.8 PYNQ-Z1

Le PYNQ-Z1 [1] est une carte de développement FPGA basée sur le FPGA ZYNQ XC7Z020. Elle a été spécifiquement conçue pour prendre en charge PYNQ, un nouveau cadre open source permettant aux programmeurs intégrés d'explorer les fonctionnalités des SoC Xilinx ZYNQ sans avoir à concevoir des circuits logiques complexes. En tirant parti de la combinaison de la logique programmable et du processeur ARM avancé de ZYNQ, les concepteurs sont en mesure de créer des systèmes embarqués plus puissants. De plus, les SoC peuvent être programmés en Python, offrant ainsi la possibilité de développer et de tester le code directement sur le PYNQ-Z1. Cette approche facilite un processus de développement rapide et efficace.

Fig. 1.9 : La carte PynqZ1

Le PYNQ-Z1 prend en charge nativement les applications multimédias grâce à ses interfaces audio et vidéo intégrées. Il est conçu pour être facilement extensible avec des périphériques Pmod, Arduino et Grove, ainsi qu'avec des broches d'E/S polyvalentes.

La carte PYNQ-Z1 peut également être étendue avec des périphériques USB tels que le Wi-Fi, le Bluetooth et les webcams. Ensuite, nous énumérons les principales interfaces et blocs intégrés dans le PYNQ-Z1. Cela nous permet de comprendre pleinement cette

[1] https://pynq.readthedocs.io/en/latest/getting_started/pynq_z1_setup

carte de développement, même si nous n'avons pas utilisé certains d'entre eux dans les expériences suivantes.

PYNQ XC7Z020-1CLG400C :

- Processeur Cortex-A9 double cœur cadencé à 650 MHz
- Contrôleur de mémoire DDR3 avec 8 canaux DMA et 4 ports esclaves AXI3 haute performance
- Contrôleurs périphériques à large bande passante : Ethernet 1G, USB 2.0, SDIO
- Contrôleur périphérique à faible bande passante : SPI, UART, CAN, I2C
- Programmable depuis JTAG, flash Quad-SPI et carte microSD
- Logique programmable de la famille Artix-7 :
  - 13 300 blocs logiques, chacun avec quatre LUTs (Tables de Vérité à 6 entrées) et 8 bascules
  - 630 Ko de RAM bloc rapide
  - 4 tuiles de gestion d'horloge, chacune avec un boucle à verrouillage de phase (PLL) et un gestionnaire d'horloge mixte (MMCM)
  - 220 blocs DSP
  - Convertisseur analogique-numérique intégré (XADC) sur puce
- Mémoire :
  - 512 Mo de DDR3 avec bus 16 bits à 1050 Mbps
  - Flash Quad-SPI de 16 Mo avec identifiant unique programmé en usine (compatible EUI-48/64TM sur 48 bits)
  - Emplacement pour carte microSD
- Alimentation :
  - Alimenté par USB ou toute source de 7V à 15V
- USB et Ethernet :
  - Circuit de programmation USB-JTAG
  - Pont USB-UART
  - PHY USB OTG (prend en charge uniquement le mode hôte)
  - PHY Ethernet Gigabit
- Audio et vidéo :
  - Microphone électret avec sortie modulée par densité d'impulsion (PDM)

Le framework PYNQ

La carte PYNQ-Z1 est conçue pour être utilisée avec le framework open-source PYNQ. Ce framework permet aux programmeurs embarqués de déployer des plates-formes SoC programmables sans avoir à utiliser des langages de description matérielle tels que Verilog. Au niveau matériel, les circuits logiques programmables du FPGA sont représentés sous forme de bibliothèques appelées overlays, qui peuvent être accessibles via une interface de programmation.

Au niveau logiciel, le langage de programmation Python, associé à l'environnement de conception Jupyter Notebook et au système d'exploitation Linux, est utilisé pour les applications embarquées à haute performance. La carte PYNQ-Z1 combine ces éléments logiciels et matériels pour simplifier et améliorer la conception de systèmes SoC (*Pynq - Python Productivity for Zynq* s. d.).

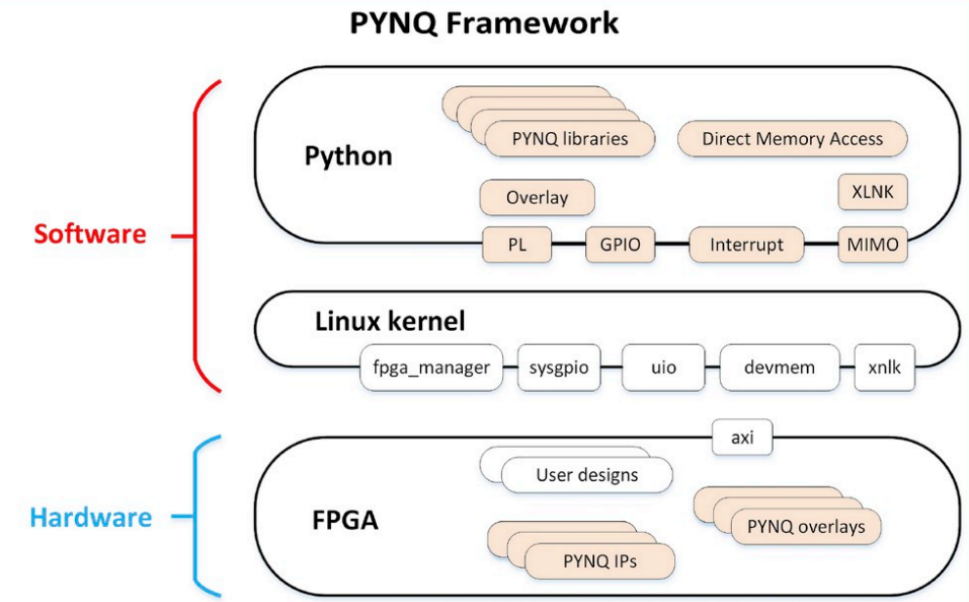


Fig. 1.10 : Le framework PYNQ (*Pynq - Python Productivity for Zynq* s. d.)

Intégration des overlays

Dans le framework PYNQ, la gestion de l'overlay, qui est une bibliothèque logicielle représentant les circuits logiques programmables du FPGA, est réalisée à la fois au niveau matériel et au niveau logiciel. Au niveau matériel, des contrôleurs matériels interagissent avec l'overlay pour permettre sa gestion par un logiciel appelé hyperviseur. L'ensemble de l'overlay et de ses contrôleurs matériels est regroupé en une IP (propriété intellectuelle).

L'hyperviseur est un logiciel qui nécessite l'utilisation d'un processeur pour son exécution. Cependant, sur la carte Pynq Z1, le FPGA est utilisé en mode autonome, ce qui signifie qu'il est utilisé indépendamment d'un processeur externe et qu'il est directement connecté au réseau. Dans cette configuration, pour exécuter l'hyperviseur, il est nécessaire d'utiliser un processeur softcore qui est implémenté à l'aide des ressources reconfigurables du FPGA.

Le processeur softcore est une implémentation de processeur réalisée en utilisant les blocs de logique programmables du FPGA. Il est configuré et programmé pour exécuter l'hyperviseur, gérer les opérations de l'overlay et faciliter la communication avec d'autres composants du système, notamment le réseau.

Overlay de base

L'objectif principal de la conception de l'overlay de base pour la carte Pynq Z1 est de permettre à la plateforme PYNQ d'utiliser les périphériques disponibles sur la carte sans nécessiter de modifications supplémentaires. L'overlay de base comprend des blocs IP matériels qui permettent de contrôler les périphériques présents sur la carte et de les connecter au processeur Zynq PS. Grâce à l'overlay de base, les périphériques peuvent être utilisés directement depuis l'environnement Python dès le démarrage du système.

Les périphériques couramment inclus dans l'overlay de base comprennent des GPIO (entrées/sorties à usage général) pour les voyants, les commutateurs, les boutons, ainsi que des interfaces vidéo, audio et autres interfaces personnalisées.

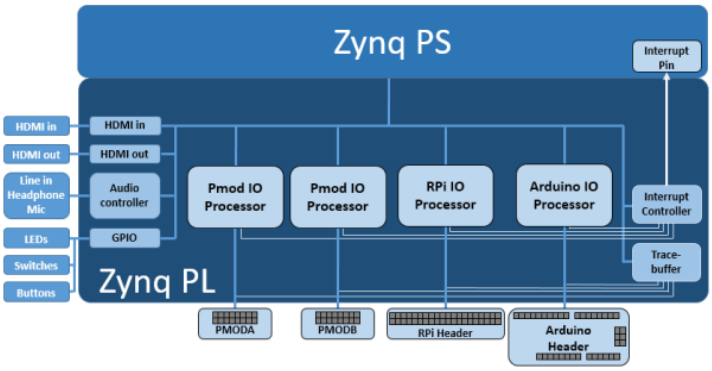


FIG. 1.11 : Overlay de base (*Pynq - Python Productivity for Zynq* s. d.)

1.3.9 Représentation des données

Les formats à virgule flottante et à virgule fixe sont deux représentations courantes des nombres réels en informatique. Le format à virgule flottante permet une représentation flexible des nombres avec une large gamme de valeurs, tandis que le format à virgule fixe utilise des bits entiers et fractionnaires pour une représentation plus précise mais limitée.(POSITRON-LIBRE 2023)

- **Virgule fixe**

  La représentation en virgule fixe est une méthode de codage des nombres réels qui utilise une virgule binaire virtuelle positionnée entre deux emplacements de bits. Elle divise les nombres en une partie entière à gauche de la virgule et une partie fractionnaire à droite. Cette représentation assigne un nombre fixe de bits pour

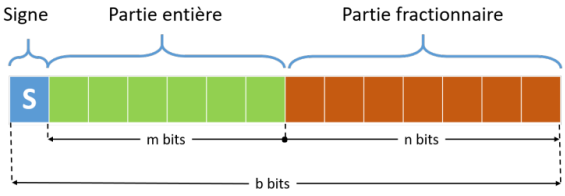


FIG. 1.12 : Représentation en virgule fixe

coder les nombres, comprenant un bit de signe , m bits pour la partie entière et n bits pour la partie fractionnaire (où b = 1 + m + n).

Dans cette méthode, tous les nombres, qu'ils soient entiers ou fractionnaires, sont codés avec le même nombre de bits. Par conséquent, la position de la virgule reste constante pour tous les nombres représentés.

- **virgule flottante**

  La représentation en virgule flottante est utilisée lorsque des niveaux élevés de précision sont nécessaires. Elle représente un nombre réel en utilisant un signe (S), un exposant (E) et une mantisse (M). La valeur associée à un nombre réel x dans la représentation en virgule flottante est donnée par l'expression suivante .

$$x = (-1)^S \times M \times 2^E$$

  La mantisse détermine la précision du nombre, tandis que l'exposant définit le décalage de la puissance de 2.

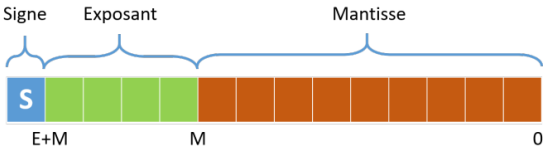


FIG. 1.13 : Représentation en virgule flottante

1.4 Présentation des outils logiciels de développement

1.4.1 Colab

Colab[2] est un produit de type Jupyter Notebook de Google Research. Un développeur de programme Python peut utiliser ce bloc-notes pour écrire et exécuter des codes de programme Python aléatoires simplement à l'aide d'un navigateur Web.

FIG. 1.14 : Logo de Colab.

1.4.2 Pytorch

:

PyTorch[3] est une bibliothèque optimisée de tenseurs pour l'apprentissage profond, basée sur Python et Torch. Elle est principalement utilisée pour les applications utilisant des GPU et des CPU. PyTorch est préféré par rapport à d'autres frameworks d'apprentissage profond tels que TensorFlow et Keras, car il utilise des graphes de calcul dynamiques et est entièrement compatible avec Python. Il permet aux scientif iques, aux développeurs et aux débogueurs de réseaux neuronaux d'exécuter et de tester des parties du code en temps réel. Par conséquent, les utilisateurs n'ont plus besoin d'attendre l'implémentation complète du code pour vérifier si une partie fonctionne correctement ou non.

FIG. 1.15 : Logo de Pytorch.

1.4.3 Keras

Keras [4] est un Framework de haut niveau, convivial et flexible qui permet aux utilisateurs de construire des modèles de Deep Learning et de choisir l'exécution sur Theano ou TensorFlow. Francis Chollet (F. CHOLLET et al. 2015), membre de l'équipe Google Brain, l'a écrit et maintient actuellement.

FIG. 1.16 : Logo de Keras.

[2] https://colab.research.google.com/ (visité le 15/05/2023).
[3] https://pytorch.org/ (visité le 15/05/2023).
[4] https://keras.io/ (visité le 15/05/2023).

1.4.4 Numpy

Numpy [5] une librairie qui permet de réaliser des calculs numériques avec le langage de programmation Python. Elle offre une gestion simplifiée des tableaux de nombres ainsi que des fonctions avancées telles que la diffusion.

FIG. 1.17 : Logo de numpy.

1.4.5 Pandas

Pandas [6]est une bibliothèque Python open source, distribuée sous licence BSD, qui propose des structures de données efficaces et conviviales ainsi que des outils d'analyse de données performants.(PYTHON consulté le mai. 06, 2023)

FIG. 1.18 : Logo de pandas.

1.4.6 Matplotlib

Matplotlib [7] est une bibliothèque de représentation graphique pour le langage de programmation Python, qui s'appuie sur l'extension mathématique numérique NumPy. Elle offre une interface de programmation orientée objet permettant d'intégrer des graphiques dans des applications informatiques.

FIG. 1.19 : Logo de Matplotlib.

1.4.7 Scikit-learn

Scikit-learn [8] est une bibliothèque Python de machine learning destinée à l'apprentissage statistique. Elle offre des fonctionnalités intermédiaires pour la création de modèles de prédiction. Elle est développée en langage de haut niveau Python.

FIG. 1.20 : Logo de Scikit-learn.

[5] https://numpy.org/ (visité le 15/05/2023).
[6] https://realpython.com/pandas-plot-python/ (visité le 15/05/2023).
[7] https://realpython.com/ (visité le 15/05/2023).
[8] https://realpython.com/ (visité le 15/05/2023).

1.4.8 WFDB

WFDB[9] Le package natif de la base de données de formes d'ondes Python (WFDB) est une bibliothèque d'outils conçue pour la lecture, l'écriture et le traitement des signaux et des annotations WFDB. Les principales composantes de ce package sont basées sur les spécifications WFDB originales, bien qu'il ne contienne pas toutes les fonctionnalités du package WFDB d'origine. L'objectif est de mettre en œuvre autant de fonctionnalités principales que possible, tout en offrant des interfaces de programmation conviviales. Des outils de traitement de signaux physiologiques supplémentaires utiles sont ajoutés régulièrement.(Package 2020)

1.4.9 Seaborn

Seaborn[10] est une bibliothèque de visualisation de données pour Python qui se base sur la bibliothèque matplotlib. Elle offre une interface de programmation de haut niveau pour créer des graphiques statistiques attrayants et informatifs.

1.4.10 Python

Python [11] est un langage de programmation interprété largement utilisé dans divers domaines. Il est fréquemment choisi comme langage de programmation d'introduction dans les établissements scolaires en raison de sa facilité d'apprentissage. Cependant, il est également utilisé par des professionnels du développement de logiciels dans des organisations prestigieuses telles que Google, la NASA. En conséquence, Python est devenu le langage le plus couramment utilisé dans le domaine du Machine Learning et de la science des données, bénéficiant de sa richesse de bibliothèques et d'outils spécifiquement conçus pour ces domaines.

Fig. 1.21 : Logo de Python.

le choix de PyTorch, Colab et Python comme matériel et logiciel pour notre implémentation est justifié par leur popularité, leur adaptabilité aux tâches d'apprentissage automatique, leur efficacité en termes de ressources et leur facilité d'utilisation. Ces outils nous permettent de développer et de déployer efficacement notre classificateur neuronal pour la détection précise des arythmies cardiaques, tout en bénéficiant d'une communauté de soutien active et de fonctionnalités avancées pour la formation et l'évaluation du modèle.

[9] https://wfdb.readthedocs.io/en/latest/ (visité le 15/05/2023).
[10] https://seaborn.pydata.org/ (visité le 15/05/2023).
[11] https://www.python.org/ (visité le 15/05/2023).

Les outils logiciels de développement pour la carte PYNQ-Z1 incluent Vitis HLS, Vivado, Jupyter et Overlay.

1.4.11 Vitis HLS

Vitis HLS[12] est un outil logiciel de développement de haut niveau (High-Level Synthesis) proposé par Xilinx. Il offre aux développeurs la possibilité de transformer du code C, C++ ou SystemC en circuits numériques optimisés pour les FPGA Xilinx. Vitis HLS simplifie le processus de conception en utilisant des langages de programmation de haut niveau et en générant automatiquement du code VHDL ou Verilog adapté au FPGA. Ainsi, les développeurs peuvent tirer parti des avantages de la programmation de haut niveau tout en obtenant des performances optimales sur le FPGA.

FIG. 1.22 : Logo de vitis.

1.4.12 Vivado

Vivado est un environnement de développement intégré (IDE) fourni par Xilinx pour la conception et la programmation des FPGA. Il offre une suite complète d'outils permettant de concevoir, de simuler, de synthétiser et de programmer des circuits numériques sur les FPGA Xilinx. Vivado prend en charge différentes étapes de la conception FPGA, de la création du projet à la génération de la configuration finale du FPGA. Il offre également des fonctionnalités avancées telles que l'analyse des contraintes, la vérification fonctionnelle et la génération de rapports détaillés sur les performances et les ressources.

FIG. 1.23 : Logo de Vivado.

1.4.13 Jupyter Notebook

Jupyter Notebook est un environnement informatique interactif basé sur un navigateur. Il permet la création de cahiers Jupyter contenant du code en direct, des widgets interactifs, des graphiques, des explications textuelles, des équations, des images et des vidéos. Dans le contexte d'une carte Zynq compatible PYNQ, il est possible de programmer facilement cette carte dans Jupyter Notebook en utilisant le langage de programmation Python. Les développeurs peuvent alors exploiter les bibliothèques matérielles et les superpositions logicielles pour la logique programmable. Les bibliothèques matérielles, également connues

FIG. 1.24 : Logo de jupyter .

[12] https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html

sous le nom de superpositions, permettent d'accélérer les logiciels s'exécutant sur une carte Zynq et de personnaliser la plate-forme matérielle ainsi que les interfaces. L'image PYNQ est une image Linux amorçable qui inclut le package pynq python ainsi que d'autres packages open source, offrant ainsi un ensemble complet d'outils pour le développement et l'utilisation de la carte Zynq compatible PYNQ.

1.5 Conclusion

En conclusion, ce chapitre "Fondements théoriques" est essentiel pour comprendre les concepts clés de l'implémentation sur FPGA des réseaux de neurones artificiels dans la classification des arythmies cardiaques. En maîtrisant ces bases théoriques, nous serons en mesure de concevoir et de mettre en œuvre efficacement des réseaux de neurones sur FPGA pour une classification précise des arythmies cardiaques, ouvrant ainsi la voie à des avancées significatives dans le domaine de la santé cardiaque.

Chapitre 2

État de l'art

2.1 Introduction

Les réseaux de neurones ont émergé en tant qu'approche prometteuse dans le domaine de la classification des arythmies. Ces réseaux ont la capacité d'apprendre à extraire automatiquement des caractéristiques pertinentes à partir des signaux ECG bruts, ce qui contribue à améliorer la précision des classifications. De plus, l'utilisation de FPGA (Field-Programmable Gate Arrays) en tant que plateforme matérielle pour l'implémentation de ces réseaux de neurones présente des avantages notables en termes de vitesse de traitement et d'efficacité énergétique.

Dans le cadre de ce chapitre, nous entreprendrons une exploration approfondie du système cardiovasculaire, de l'ECG et des arythmies. Nous examinerons en détail les différentes méthodes de classification des arythmies, en mettant particulièrement l'accent sur l'utilisation des réseaux de neurones. De plus, nous aborderons l'état de l'art en matière d'implémentation des réseaux de neurones sur FPGA pour la classification des arythmies

2.2 Système cardiovasculaire et les arythmies cardiques

2.2.1 Système cardiovasculaire

Le coeur est composé de quatre cavités : deux oreillettes et deux ventricules qui se remplissent et se vident de sang. D'un côté le sang qui vient de l'organisme pauvre en O2 et riche en CO2 entre dans l'oreillette droite par les veines caves supérieure et inférieure et passe au ventricule droit où il sera éjecté dans les artères pulmonaires. Au niveau du poumon et à l'aide d'alvéoles, le sang se débarrasse du CO2 et absorbe l'O2 puis retourne au coeur dans l'oreillette gauche par les veines pulmonaires. D'un autre côté et simultanément le sang fraîchement oxygéné qui revient des poumons vers le coeur dans l'oreillette gauche, passe dans le ventricule gauche, qui à son tour le propulse dans l'aorte qui irrigue les organes à travers ses ramifications. Le sang pauvre en oxygène quant à lui retourne au coeur dans l'oreillette droite par les veines caves supérieure et inférieure. Ce cycle se répète continuellement.

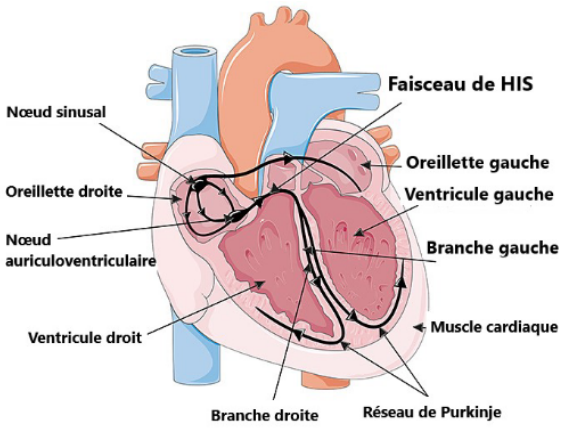


Fig. 2.1 : la disposition du muscle cardiaque (JOHNY 2023)

Dans les prochains paragraphes, nous aborderons l'activité cardiaque sous deux perspectives distinctes : l'activité mécanique et l'activité électrique.

2.2.1.1 Activité mécanique cardiaque

L'activité mécanique cardiaque est assurée par les contractions du myocarde, le tissu musculaire du cœur, qui permettent la propulsion du sang à travers l'organisme.
Ce cycle est constitué de trois phases bien distinctes, à savoir :

- **La systole auriculaire** qui correspond à la contraction des oreillettes et à l'éjection du sang vers les ventricules, suivie de la fermeture des valves auriculo-ventriculaires et du bruit grave du cœur.
- **La systole ventriculaire** correspond à la contraction des ventricules et à l'éjection du sang vers les organes, suivie de la fermeture des valves sigmoïdes ou artérielles, et du bruit aigu du cœur.
- **La diastole** correspond à la phase de relaxation de toutes les parties du cœur, au remplissage passif et elle représente les deux tiers du cycle cardiaque.

2.2.1.2 Activité électrique de coeur

Le cœur génère une activité électrique autonome qui suit l'activité mécanique et ne dépend pas du système nerveux. Cette autonomie est due à un groupe de cellules appelé "noeud sinusal" situé dans l'atrium gauche. Ce groupe produit un signal électrique de quelques millivolts qui est ensuite transmis à un autre groupe de cellules nommé "noeud atrio-ventriculaire". L'influx nerveux se propage alors aux ventricules grâce au faisceau de His et au réseau de Purkinje. En cas de perturbation de cette conduction électrique, on parle d'arythmie ou de trouble de la conduction. L'électrocardiogramme (ECG) permet de mesurer cette activité électrique. Chaque onde de l'ECG est associée à un phénomène électrique spécifique : l'onde P correspond à la contraction des oreillettes (dépolarisation auriculaire), le complexe QRS reflète la contraction des ventricules (dépolarisation ventriculaire), et enfin l'onde T correspond au contrôle électrique de la repolarisation ventriculaire.

Traitons maintenant l'électrocardiogramme.

2.2.2 Électrocardiogramme ECG

L'enregistrement de l'activité électrique cardiaque est possible grâce au signal électrocardiographique, ou ECG (Goldberger et al. 2018). Cette mesure électrophysiologique est caractérisée par une séquence d'ondes se répétant à chaque cycle cardiaque et présentant des formes et des durées spécifiques. Chacune de ces ondes reflète les événements mécaniques et électriques successifs survenant dans le cœur, tels qu'illustrés sur la figure 2.2 .

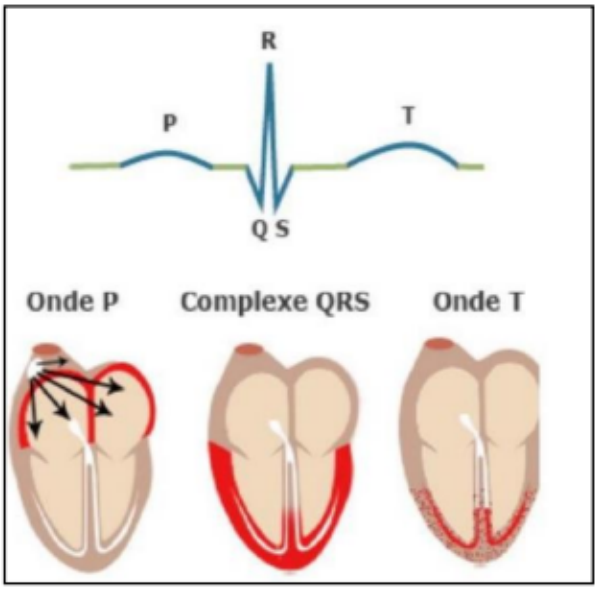


FIG. 2.2 : Ondes du signal ECG (GOLDBERGER et al. 2018)

2.2.2.1 Les ondes du signal ECG

L'enregistrement électro-physiologique de l'activité cardiaque, communément appelé électrocardiogramme ou ECG, se compose de déflexions positives et négatives superposées à une ligne de base représentant l'absence de phénomènes cardiaques, tel que dépeint sur la figure 2.2
Les ondes principales de l'ECG sont désignées par les lettres P, Q, R, S et T, selon leur signification physiologique.

- **L'onde P** traduit la dépolarisation auriculaire, qui se propage à travers toutes les cellules de l'oreillette à partir du nœud sinusal.
- **Le complexe QRS** reflète la propagation du stimulus électrique à travers les ventricules. Il est constitué d'une déviation initiale négative (onde Q), d'une première déviation positive (onde R) et d'une déviation négative suivante (onde S).
- **L'onde Q** représente la dépolarisation septale ventriculaire
- **L'onde R** est la conséquence de la dépolarisation du muscle ventriculaire résultant.
- **L'onde S** correspond à la dépolarisation ventriculaire basale.
- **L'onde T** indique une partie de la repolarisation ventriculaire et succède au complexe QRS.
- **L'onde U** peut apparaître après l'onde T et traduit une activité électrique déphasée du muscle papillaire ventriculaire.

L'analyse de l'ECG inclut l'évaluation de la morphologie et de l'amplitude des ondes P, QRS et T, ainsi que des paramètres temporels tels que les segments PR et ST et les intervalles PR, QT et ST.

2.2.2.2 Dérivations électro-cardiographiques

L'électrocardiographie (*Dérivations de l'électrocardiogramme* s. d.) est une technique qui permet de mesurer l'activité électrique du cœur en utilisant des dérivations. Une dérivation est définie comme la mesure de la différence de potentiel électrique entre deux points d'observation. Les électrocardiographes sont capables d'enregistrer plusieurs dérivations simultanément, en fonction du nombre et de l'emplacement des électrodes placées sur le corps. Ces mesures permettent d'explorer l'ensemble du champ électrique cardiaque résultant de la contraction du myocarde. Le positionnement des électrodes est soigneusement sélectionné pour assurer une couverture optimale du cœur

- **a. Les dérivations périphériques** : L'électrocardiographie utilise les dérivations périphériques pour l'étude de l'activité électrique cardiaque dans le plan frontal. Cette méthode consiste à placer quatre électrodes sur le bras droit, le bras gauche, la jambe gauche et la jambe droite (électrode neutre). Les dérivations bipolaires classiques (DI, DII, DIII) enregistrent la différence de potentiel entre deux électrodes placées à des extrémités différentes. Elles utilisent trois électrodes pour former un triangle d'Einthoven sur le patient.Les trois dérivations sont :

  1. DI (dérivation I) avec DI = VL –VR
  2. DII (dérivation II) avec DII = VF – VR
  3. DIII (dérivation III) avec DIII = VF –VL

  Avec : VL : le potentiel sur le bras gauche, VR : le potentiel sur le bras droit et VF : le potentiel sur la jambe gauche.

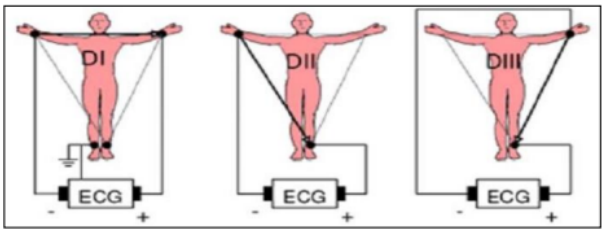


Fig. 2.3 : Dérivations bipolaires (PierreTaboulet 2023)

Les dérivations unipolaires des extrémités enregistrent la différence de potentiel entre un point théorique situé au centre du triangle et l'électrode de chaque extrémité, permettant ainsi de déterminer le potentiel absolu dans chaque électrode. À l'origine désignées VR, VL et VF, ces dérivations ont ensuite été amplifiées et renommées Vra, Vla et Vfa.

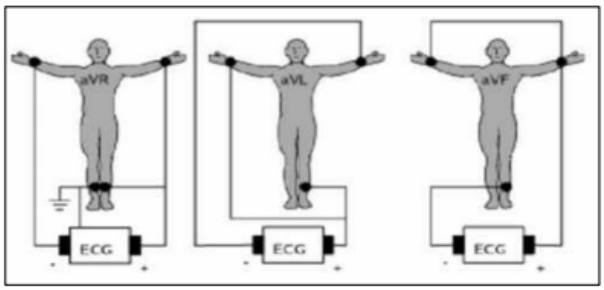


Fig. 2.4 : Dérivations unipolaires (PierreTaboulet 2023)

b.Les dérivations précordiales : En électrocardiographie, les pistes précordiales sont au nombre de six et sont désignées par un V majuscule suivi d'un nombre de 1 à 6. Ce sont des dérivations unipolaires qui enregistrent le potentiel absolu du point où est placée l'électrode portant le même nom. Elles sont particulièrement utiles pour la détection des anomalies du ventricule gauche, en particulier celles des parois antérieure et postérieure. En effet, sur un électrocardiogramme normal, les complexes QRS sont principalement négatifs dans les dérivations V1 et V2, et majoritairement positifs dans les dérivations V4 à V6, ce qui correspond à un profil Rs.

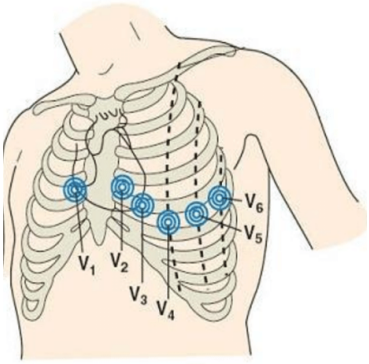


Fig. 2.5 : Dérivations thoraciques.(Tedjini et al. 2014)

Ces dérivations sont positionnées comme suit :

- V1 : 4ème espace intercostal, bord droit du sternum (ligne parasternale).
- V2 : 4ème espace intercostal, bord gauche du sternum (ligne parasternale).
- V3 : à mi-distance entre V2 et V4.
- V4 : 5ème espace intercostal, ligne médio-claviculaire gauche
- V5 : à mi-distance entre V4 et V6, sur la ligne axillaire antérieure.
- V6 : même niveau horizontal que V4 et V5, ligne axillaire moyenne.

2.2.3 Les arythmies cardiques

L'arythmie cardiaque désigne une perturbation du rythme normal du cœur. Bien que certaines formes soient bénignes, d'autres peuvent entraîner des symptômes tels que des douleurs thoraciques ou des étourdissements, voire même mettre en danger la vie du patient en perturbant l'irrigation sanguine. En fonction de la vitesse à laquelle le cœur bat, l'arythmie peut être qualifiée de tachycardie si elle accélère le rythme, ou de bradycardie si elle le ralentit.(KASTOR 2000)

Les Causes d'Arythmies

Les arythmies cardiaques peuvent être déclenchées par divers facteurs, notamment

- Des problèmes de circulation sanguine dans l'artère coronaire,
- Des déséquilibres des concentrations d'électrolytes dans le sang tels que le sodium ou le potassium,
- Des altérations dans les tissus musculaires du coeur, des lésions après une crise cardiaque ou encore
- Des perturbations survenant durant le processus de guérison après une intervention chirurgicale cardiaque. De plus, il est possible que des troubles du rythme cardiaque se produisent même chez des individus présentant un coeur en bonne santé et normalement fonctionnel.(Wedro et al. 2019)

Les Types d'Arythmies

Les arythmies peuvent être classées de différentes manières,

Rythmes cardiaques rapides

2.2.3.1 Tachycardie supraventriculaire (TSV)

Il s'agit d'un type de tachycardie anormale commun chez les jeunes adultes, caractérisé par une fréquence cardiaque rapide qui dépasse souvent 150 battements par minute. Cette accélération du rythme cardiaque se produit dans les cavités supérieures du cœur ou dans la partie supérieure du système de conduction électrique. Les symptômes associés à cette condition incluent des palpitations, des douleurs thoraciques, des maux d'estomac.

2.2.3.2 Tachycardie auriculaire

La tachycardie auriculaire, aussi connue sous les appellations flutter auriculaire ou fibrillation auriculaire, est une forme de trouble du rythme cardiaque qui se caractérise par une fréquence cardiaque élevée initiée dans les cavités supérieures du cœur et transmise

vers les cavités inférieures. Elle survient souvent à la suite d'une intervention chirurgicale impliquant les oreillettes, ainsi qu'en présence d'une dilatation de ces dernières, généralement causée par une fuite ou une obstruction des valves mitrale ou tricuspide. Les symptômes de cette condition incluent une accélération du rythme cardiaque, une fatigue, des vertiges, des étourdissements et des évanouissements.

2.2.3.3 Tachycardie ventriculaire

La tachycardie ventriculaire est une manifestation de l'arythmie cardiaque qui se caractérise par un rythme cardiaque rapide débutant dans les cavités inférieures du cœur. Elle découle souvent d'une pathologie cardiaque sévère et requiert fréquemment une intervention thérapeutique rapide ou urgente. Bien que certains symptômes puissent être bénins, ils sont le plus souvent graves et incluent des sensations vertigineuses, des étourdissements et des évanouissements.

Rythmes cardiaques lents

2.2.3.4 Dysfonctionnement du nœud sinusal

Le nœud sinusal, point de départ des contractions cardiaques(le battement du cœur), peut être altéré à la suite d'une intervention chirurgicale, conduisant à un syndrome des sinus. Dans ce cas, le rythme cardiaque est caractérisé par une lenteur et une difficulté à augmenter en réponse à l'effort physique. Les patients peuvent être asymptomatiques ou souffrir de fatigue, d'intolérance à l'exercice, de vertiges ou d'épisodes de syncope.

2.2.3.5 Bloc auriculo-ventriculaire complet (bloc cardiaque complet)

Lorsqu'un signal électrique ne peut pas circuler normalement des cavités supérieures du cœur, un blocage cardiaque complet peut se produire. Si le nœud A-V subit des dommages pendant une intervention chirurgicale, cela peut entraîner un blocage cardiaque complet. Dans certains cas, ce blocage se produit spontanément, sans intervention chirurgicale. L'utilisation d'un stimulateur cardiaque artificiel peut rétablir le rythme et la fréquence cardiaques normaux.

2.2.3.6 Les catégories d'arythmies cardiaques

Les arythmies cardiaques sont des troubles du rythme cardiaque qui peuvent être classifiés en différentes catégories. Parmi les cinq classes d'arythmies cardiaques couramment identifiées, on retrouve les classes N, S, V, F et Q.

La classe N fait référence aux arythmies normales ou sinusales, où le rythme cardiaque est régulier et provient du nœud sinusal, qui est le stimulateur cardiaque naturel. Ce type d'arythmie est considéré comme normal et sain.

La classe S représente les arythmies supraventriculaires, qui se produisent au-dessus des ventricules, les principales chambres de pompage du cœur. Ces arythmies peuvent inclure des anomalies du nœud sino-atrial ou des voies de conduction supraventriculaires.

La classe V désigne les arythmies ventriculaires, qui se produisent dans les ventricules du cœur. Ces arythmies peuvent être plus graves, car elles affectent directement les chambres de pompage du cœur. Des exemples courants d'arythmies ventriculaires comprennent la fibrillation ventriculaire et la tachycardie ventriculaire.

La classe F se rapporte aux arythmies auriculaires, qui se produisent dans les oreillettes du cœur. Ces arythmies peuvent inclure des irrégularités dans la contraction des oreillettes, telles que la fibrillation auriculaire.
Lorsqu'une arythmie cardiaque ne peut être clairement identifiée dans l'une des classes précédemment mentionnées (N, S, V, F), elle peut être classée dans **la catégorie Q** pour indiquer une classe inconnue ou non spécifiée. Cela peut se produire lorsqu'il y a des difficultés à interpréter l'arythmie en raison de ses caractéristiques particulières ou lorsque des informations supplémentaires sont nécessaires pour établir une classification précise.

2.2.4 Méthodes de Détection des Arythmies

En présence de troubles de conduction, un battement cardiaque anormal peut être observé. Si ces problèmes persistent, la forme anormale se répète, conduisant à une arythmie ou à une dégradation de l'ECG rendant sa reconnaissance difficile. Pour analyser l'arythmie, trois approches générales sont utilisées. La première méthode consiste à détecter les ondes QRS et à classifier les battements, en qualifiant une série de battements d'un type particulier comme arythmiques. La deuxième approche consiste à analyser une section de l'ECG couvrant plusieurs intervalles de battement, à calculer une statistique (par exemple, la variance ou le rapport de puissance à différentes fréquences) et à utiliser cette information pour classifier l'arythmie. La troisième option consiste à créer un modèle de la dynamique attendue pour différents rythmes et à comparer le signal observé (ou ses caractéristiques dérivées) à ce modèle. Les méthodes basées sur des modèles peuvent être soit des méthodes basées sur l'ECG, soit des méthodes basées sur des statistiques d'intervalle RR.

2.3 Méthodes de classification des arythmies cardiaques

2.3.1 Classification

La classification implique le regroupement d'observations similaires en classes ou en catégories distinctes. Son objectif fondamental est d'attribuer une classe spécifique à une nouvelle observation dont la classe est inconnue. Le nombre de classes établies dépend étroitement de l'application en question.

Principe

La classification est représentée par des modèles, des méthodes ou des algorithmes spécifiques qui sont utilisés pour organiser et regrouper les données en classes distinctes Cette formule peut représenter le classifieur souhaité.(Izeboudjen 1999)

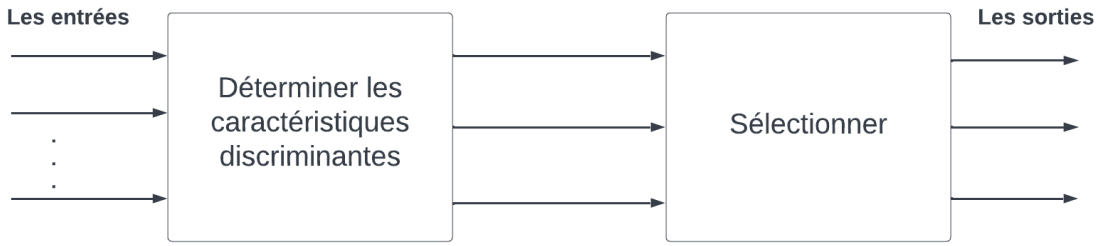


Fig. 2.6 : Classifieur

La classification se déroule généralement en deux phases, conformément à la Figure 2.6.

La phase d'apprentissage (Entraînement), a pour objectif d'inférer un modèle à partir de données d'entraînement. Pour ce faire, des paramètres sont extraits des observations afin de réduire la dimension du signal. Dans le contexte des signaux biologiques, les observations sont généralement des signaux temporels obtenus par le biais d'un capteur. Les paramètres extraits dépendent fortement de l'application de classification et du signal considéré, et ils peuvent avoir un impact significatif sur les performances.

Après l'extraction des paramètres, l'algorithme d'apprentissage analyse ces derniers afin de configurer le modèle du classifieur. Cette étape peut être réalisée de manière supervisée ou non-supervisée.
Dans le cas de **l'apprentissage non-supervisé**, l'algorithme n'a pas connaissance des classes ni de leur nombre. Son objectif est de trouver des similarités entre les différentes observations afin de les regrouper en classes.
En revanche, dans le cas de **l'apprentissage supervisé** , chaque observation d'entraînement est associée à un label. L'algorithme tente alors d'ajuster le modèle de manière à ce que la décision de classe pour chaque observation corresponde à celle donnée en entrée. Cela permet à l'algorithme de s'orienter vers la bonne réponse, connaissant à l'avance la classe correcte, et ainsi de modifier les paramètres du modèle pour obtenir la bonne sortie. L'étape d'apprentissage est généralement effectuée hors ligne.

La phase de prédiction, consiste à extraire un vecteur de paramètres à partir d'une nouvelle observation et à déterminer sa classe d'appartenance. Les paramètres extraits lors de cette étape sont les mêmes que ceux utilisés lors de l'apprentissage. Cette phase est également appelée étape de généralisation, car elle évalue la capacité du modèle à classifier des observations qui n'ont aucun lien avec l'apprentissage initial.

En résumé, la classification se déroule en deux phases : l'apprentissage, où le modèle est inféré à partir des données d'entraînement en extrayant des paramètres significatifs, et la prédiction, où de nouveaux paramètres sont extraits et utilisés pour déterminer la classe d'appartenance.(P. Chollet 2017)

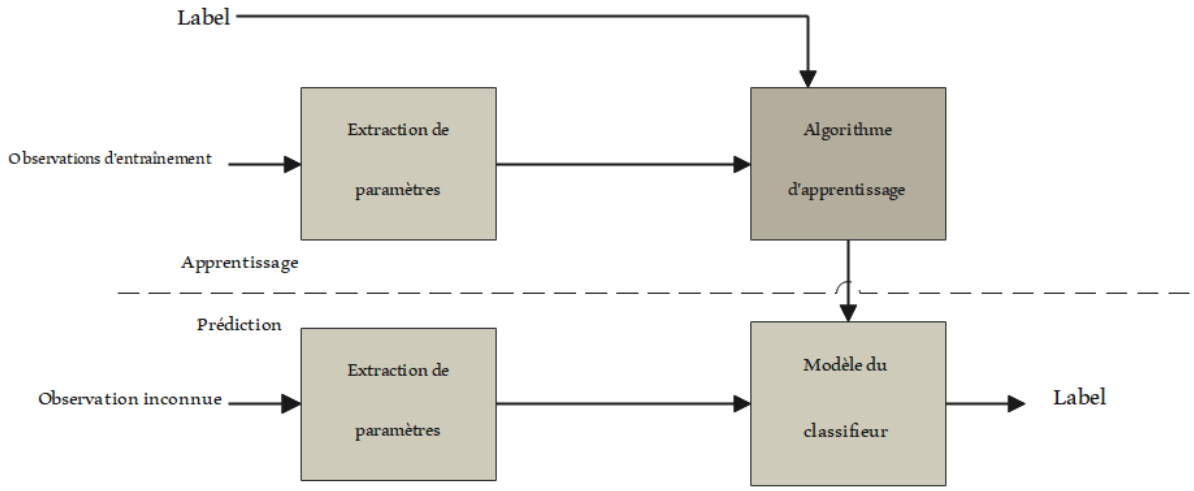


Fig. 2.7 : Schéma de principe de la classification supervisée. L'apprentissage (a) permet de fixer les paramètres du modèle grâce aux observations d'entraînement. La phase de prédiction (b) utilise le modèle pour déterminer la classe ou le label.

2.3.1.1 Ensembles de données

On distingue 3 partitions au niveau du dataset :

1. **L'ensemble d'entraînement** : avec lequel on fixe nos paramètres (les poids et les biais).
2. **L'ensemble de validation** : avec lequel on fixe nos hyperparamètres (le nombre d'itérations, le taux d'apprentissage, ...etc).
3. **L'ensemble de test.**

Fig. 2.8 : Exemple de répartition d'un dataset

2.3.2 Les modèles de classification des arythmies cardiaques

Dans le domaine de la reconnaissance des battements cardiaques à partir des signaux ECG, diverses méthodes de classification sont utilisées. Parmi elles, on retrouve les méthodes basées sur l'apprentissage automatique telle que SVM, KNN et celles basées sur l'apprentissage profond telle que CNN .

2.3.2.1 Réseaux de neurones convolutifs (CNN)

Un réseau de neurones convolutif CNN se compose de plusieurs couches connectées de manière anticipée.(Ebrahimi et al. 2020) Les couches principales comprennent la couche convolutive, couche de mise en commun et couche entièrement connectée. Les deux premières couches sont responsables de l'extraction des caractéristiques, tandis que les couches entièrement connectées sont chargées de la classification. La figure 2.9, illustre

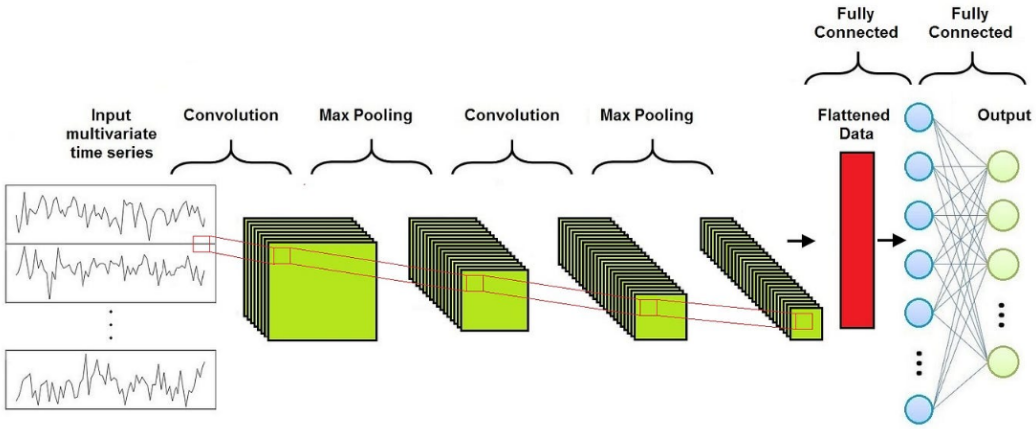


Fig. 2.9 : Architecture de CNN (Ebrahimi et al. 2020)

l'architecture du réseau de neurones convolutifs.

Couche convolutive (Convolutional Layer) : la convolution est définie par un ensemble de filtres (des matrices de taille fixe) qui sont appliquées à une sous-matrice de la carte des caractéristiques (features map) en entrée pour donner en sortie la somme du produit de chaque élément du filtre avec l'élément dans la même position de la sousmatrice afin d'extraire les caractéristiques de haut niveau de données. Les valeurs des filtres sont considérées comme des poids qui peuvent être entraînés et ensuite apprises pendant l'entraînement. Deux paramètres importants qui doivent être choisis pour la couche convolutive sont le stride et le Padding.

1. **Stride** :il contrôle la façon dont le filtre s'articule autour d'une carte des caractéristiques en entrée. En particulier, la valeur de foulée indique combien d'unités doivent être décalées à la fois.

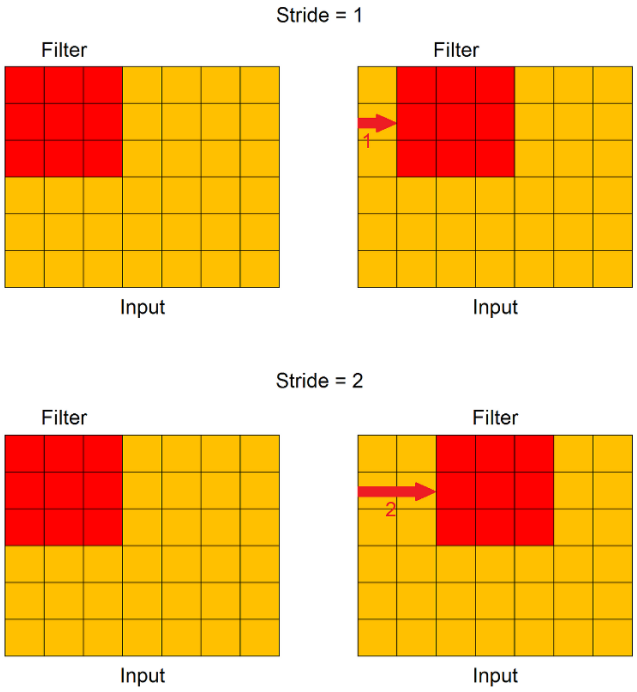


FIG. 2.10 : Mécanisme de stride

2. **Padding** :indique le nombre de colonnes et de lignes supplémentaires à ajouter autour d'une carte des caractéristiques en entrée, avant d'appliquer un filtre de convolution. Toutes les cellules des nouvelles colonnes et lignes ont une valeur fictive, généralement 0. Le Padding est utilisé car lorsqu'un filtre de convolution est appliqué à une carte des caractéristiques en entrée de petite taille. Ensuite, après l'application de nombreux filtres, la taille peut devenir trop petite. En ajoutant des lignes et des colonnes supplémentaires, nous pouvons conserver la taille d'origine ou la ralentir.

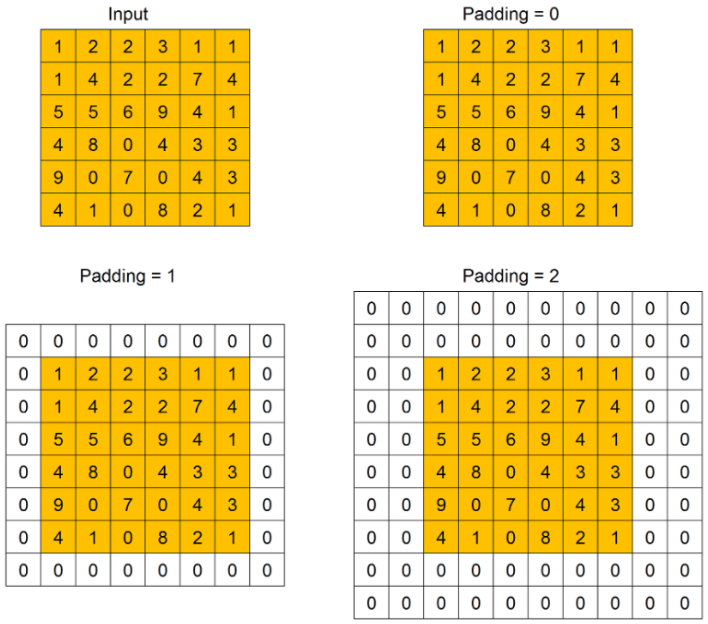


Fig. 2.11 : Mécanisme de Padding

Couche de mise en commun (Pooling Layer) : le but de l'opération de pooling est d'obtenir une réduction de dimension des cartes de caractéristiques, en préservant autant d'informations que possible. Il est également utile pour extraire les caractéristiques dominantes qui sont invariantes en rotation et en position. Son entrée est une série de cartes de caractéristiques et sa sortie est une série différente de cartes de caractéristiques, avec une dimension inférieure. Le pooling est appliquée aux fenêtres coulissantes de taille fixe sur la largeur et la hauteur de chaque carte d'entités en entrée. Il existe deux types de pooling : Max Pooling and Average Pooling Max Pooling fonctionne également comme un suppresseur de bruit, éliminant complètement les activations bruyantes. Par conséquent, il fonctionne généralement mieux que Average Pooling. Aussi pour la couche de pooling, le stride et le pading doivent être spécifiés.

Couche entièrement connectée (Fully-Connected Layer FC) : L'objectif de la FC est d'apprendre des combinaisons non linéaires des caractéristiques de haut niveau représentées par la sortie de la couche convolutive et de la couche de mise en commun. Après plusieurs opérations de convolution et de regroupement, la TS d'origine est représentée par une série de cartes de caractéristiques. Toutes ces cartes de caractéristiques sont aplaties dans un vecteur de colonne, c'est-à-dire la représentation finale de la série chronologique multivariée d'entrée d'origine. La colonne aplatie est connectée au Perceptron multicouche, dont la sortie a un nombre de neurones égal au nombre de classes possibles de séries temporelles. La rétro propagation est appliquée à chaque itération de l'entrainement. Sur une série d'époques, le modèle est capable de distinguer les séries temporelles d'entrée grâce à leurs caractéristiques dominantes de haut niveau et de les classer.

Hyperparamètres

Pour les réseaux de neurones convolutionnels (CNN), il existe de nombreux hyperparamètres à spécifier. Les plus importants sont les suivants :

Nombre de filtres de convolution : Évidemment, trop peu de filtres ne peuvent pas extraire suffisamment de caractéristiques pour parvenir à une classification. Cependant, un nombre excessif de filtres est inutile lorsque les filtres sont déjà suffisants pour représenter les caractéristiques pertinentes, et cela rend l'entraînement plus coûteux en termes de puissance de calcul.

Taille et valeurs initiales des filtres de convolution : Les filtres plus petits collectent autant d'informations locales que possible, tandis que les filtres plus grands représentent des informations plus globales, de haut niveau et représentatives. Les filtres sont généralement initialisés avec des valeurs aléatoires.

Initialisation des poids : Les poids sont généralement initialisés avec de petits nombres aléatoires pour éviter les neurones inactifs.

Fonction d'activation : La fonction d'activation introduit de la non-linéarité dans le modèle. La fonction rectifieur (Rectifier), sigmoïde ou tangente hyperbolique sont généralement choisies.

Nombre d'époques : Le nombre d'époques correspond au nombre de fois où l'ensemble d'entraînement complet passe à travers le modèle. Ce nombre devrait être augmenté jusqu'à ce qu'il y ait une petite différence entre l'erreur de test et l'erreur d'entraînement, si les performances computationnelles le permettent.

2.3.2.2 k-plus proches voisins (KNN)

L'algorithme des k-plus proches voisins (kNN) est un algorithme d'apprentissage automatique simple mais efficace. Il est utilisé à la fois pour la classification et la régression, mais il est principalement utilisé pour la prédiction de classification. Le kNN regroupe les données en clusters cohérents ou sous-ensembles et classe les nouvelles données d'entrée en fonction de leur similarité avec les données précédemment entraînées. L'entrée est attribuée à la classe avec laquelle elle partage le plus de voisins les plus proches.(Laaksonen et al. 1996).
Lorsque le kNN est utilisé pour la classification, la prédiction est effectuée en déterminant la classe majoritaire parmi les k échantillons les plus proches du nouvel échantillon. La distance entre les échantillons est généralement calculée à l'aide de mesures telles que la distance euclidienne ou la distance de Manhattan. Le choix de la valeur k est important, car une valeur trop petite peut entraîner une sensibilité excessive au bruit, tandis qu'une valeur trop grande peut entraîner une perte de détails locaux. Bien que le kNN soit efficace, il présente de nombreuses faiblesses.

2.3.2.3 Machines à vecteurs de support (SVM) :

SVM (Support Vector Machine) est un algorithme d'apprentissage automatique supervisé utilisé à la fois pour la classification et la régression. Il est principalement utilisé pour la classification de données dans des problèmes de classification binaire et multi-classes.

L'objectif de SVM est de trouver un hyperplan dans un espace de dimension supérieure (ou dans le même espace de dimension) qui sépare les échantillons de différentes classes de manière optimale. Cet hyperplan est déterminé en utilisant les échantillons d'entraînement les plus proches, appelés vecteurs de support.

1. Formation : L'algorithme SVM utilise un ensemble d'échantillons d'entraînement étiquetés pour apprendre à séparer les classes. Il cherche à trouver l'hyperplan qui maximise la marge entre les vecteurs de support des différentes classes.

2. Transformation des données : Dans certains cas, lorsque les classes ne sont pas linéairement séparables, les données d'entrée peuvent être transformées dans un espace de dimension supérieure à l'aide de techniques telles que le noyau (kernel) pour rendre la séparation linéaire possible.

3. Prédiction : Une fois l'hyperplan optimal trouvé, il peut être utilisé pour prédire la classe des nouveaux échantillons. L'échantillon est attribué à la classe correspondante en fonction de son emplacement par rapport à l'hyperplan.

Il convient de noter que les explications fournies ci-dessus sont une description générale de l'algorithme SVM. Il existe différentes variantes et paramètres qui peuvent être utilisés pour ajuster et améliorer les performances de l'algorithme dans différents scénarios d'apprentissage automatique.(Yang et al. 2015)

2.3.2.4 La régression logistique

La régression logistique est un algorithme d'apprentissage automatique utilisé pour la classification de données. Il est principalement utilisé dans des problèmes de classification binaire, où l'objectif est de prédire une variable binaire (par exemple, vrai/faux, positif/négatif) en fonction d'un ensemble de variables prédictives.
L'algorithme de régression logistique utilise une fonction mathématique appelée fonction sigmoïde pour modéliser la probabilité qu'un échantillon appartienne à une classe spécifique. La fonction sigmoïde transforme les valeurs d'entrée en une probabilité comprise entre 0 et 1.
Voici comment fonctionne l'algorithme de régression logistique :

1. Formation : L'algorithme utilise un ensemble d'échantillons d'entraînement étiquetés pour apprendre les poids ou les coefficients associés à chaque variable prédictive. Ces coefficients sont utilisés pour combiner les variables prédictives et calculer la probabilité d'appartenance à une classe.

2. Transformation des données : Les variables prédictives peuvent être transformées ou prétraitées si nécessaire pour améliorer les performances du modèle. Cela peut inclure des étapes telles que la normalisation des données ou la création de variables supplémentaires à partir des variables existantes.

3. Prédiction : Une fois que le modèle est entraîné, il peut être utilisé pour prédire la classe d'un nouvel échantillon. La probabilité prédite est comparée à un seuil (par exemple, 0,5) pour attribuer l'échantillon à la classe correspondante.(Kirasich et al. 2018)

2.3.3 Méthodes de classification des arythmies cardiaques

Il existe diverses approches de classification qui continuent de progresser grâce au développement technologique. Elles comprennent des techniques telles que les voisins les plus proches (KNN), les séparateurs à vaste marge (SVM), les réseaux de neurones, ainsi que des combinaisons de ces méthodes (ANN, SVM, KNN, etc.) et l'apprentissage profond (deep learning). Bien que ces méthodes puissent différer dans les outils utilisés à chaque étape, elles partagent généralement une architecture globale illustrée dans la Figure 2.12. La plupart des méthodes de classification débutent par une étape de prétraitement, où les données sont préparées avant d'être soumises au processus de classification. Afin d'améliorer l'utilisabilité du signal, une étape de prétraitement est effectuée pour le rendre plus exploitable. Ensuite, des paramètres pertinents sont extraits du signal, qui seront utilisés dans la phase finale de classification (apprentissage et test) pour attribuer la classe appropriée au battement cardiaque.(MEDDAH 2021).

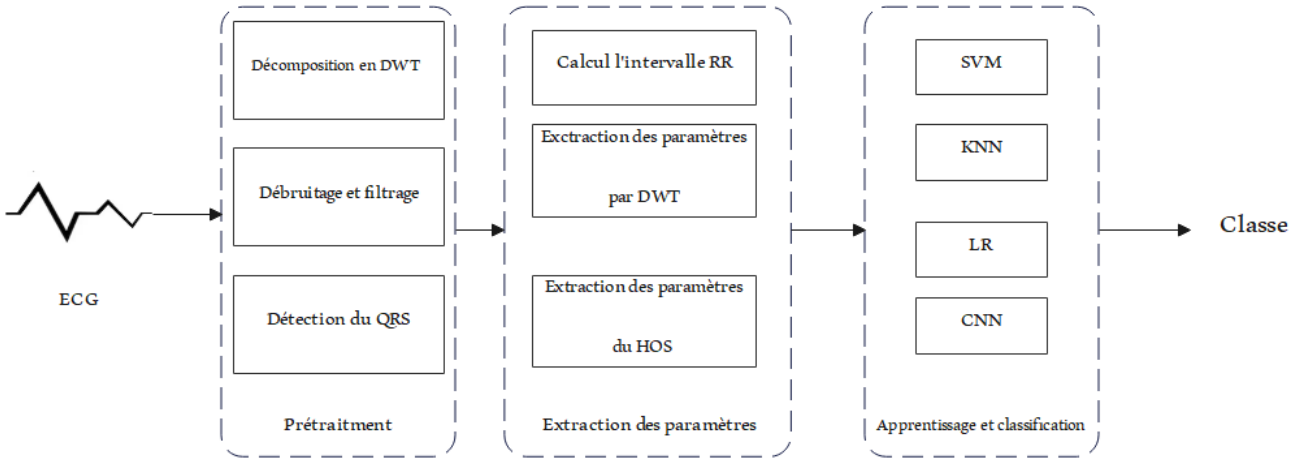


Fig. 2.12 : Organigrame globale d'une architecture de classification (MEDDAH 2021)

Prétraitment

Pour préparer le signal ECG à la classification, une étape de prétraitement est essentielle pour conditionner le signal et limiter les sources d'erreurs telles que les bruits

introduits lors de l'acquisition. Cette étape peut impliquer l'utilisation de filtres numériques,(Xia et al. 2018)(Afkhami et al. 2016)(Thomas et al. 2015) ou d'autres outils tels que les ondelettes (Alqudah et al. 2019)(Elhaj et al. 2016). Une opération importante du prétraitement est la segmentation du signal en battements , qui peut être réalisée à l'aide d'un détecteur de complexe QRS. Cela permet de sélectionner les segments individuels (battements) pour la classification ultérieure. Dans une étude spécifique, différentes étapes de prétraitement sont utilisées (He et al. 2020), notamment la calibration de la fréquence d'échantillonnage, l'élimination de la ligne de base à l'aide de filtres médians, le débruitage du signal à l'aide de la transformée en ondelette discrète (DWT) et la segmentation du signal en utilisant le détecteur du complexe QRS de Pan (Pan et al. 1985). Il est important de souligner que ces phases de prétraitement permettent une meilleure extraction des caractéristiques du signal ECG, ce qui conduit à une meilleure précision de classification.

Extraction des paramètres

Les performances des techniques de détection automatique des battements cardiaques à partir de l'électrocardiogramme (ECG) et de classification dépendent essentiellement des caractéristiques extraites du signal. Différentes approches sont utilisées pour extraire ces paramètres, qu'ils soient temporels, fréquentiels ou obtenus à partir d'une analyse temps-fréquence.

Une méthode proposée par Thomas et ses collaborateurs (Thomas et al. 2015) consiste à utiliser une analyse temps-fréquence basée sur les ondelettes pour extraire les caractéristiques pertinentes. Ils calculent également des paramètres statistiques d'ordre supérieur, tels que le Skewness et le Kurtosis, pour décrire la forme des battements. De plus, l'information sur l'énergie des battements et l'intervalle RR, des mesures couramment utilisées dans ce domaine, est exploitée pour différencier les battements normaux des battements prématurés ventriculaires (PVC) (Zhang et al. 2014).

Étant donné le nombre élevé de paramètres extraits à cette étape, il est nécessaire de réduire la dimension de ces vecteurs de caractéristiques. Plusieurs techniques de réduction de dimension sont couramment utilisées, notamment l'analyse en composantes principales (ACP) (Alqudah et al. 2019), l'analyse en composantes indépendantes (ICA) (Ye et al. 2012), ou encore une combinaison des deux approches [(Li et al. 2016)(Elhaj et al. 2016)(Martis et al. 2013)].

Une étude comparative menée par Chawla (Chawla 2009) a examiné l'utilisation de l'ACP et de l'ICA pour réduire le bruit et les artefacts présents dans le signal ECG. Les résultats ont montré que l'ACP est plus efficace pour atténuer le bruit, tandis que l'ICA est plus performante pour extraire les caractéristiques pertinentes. Basée sur cette étude, Ye(Ye et al. 2012) a utilisé l'ICA pour extraire les caractéristiques du signal, initialement composées de 114 paramètres, qui ont ensuite été réduits à seulement 18 grâce à l'application de l'analyse en composantes principales.

Classification

Dans cette étape, il est essentiel d'utiliser les caractéristiques extraites lors de l'étape précédente afin d'identifier le meilleur classifieur ou les meilleurs paramètres pour différencier efficacement les différents types de battements. Une fois cela réalisé, la base de données de test est utilisée pour évaluer les performances du système mis en place.

Un CNN individuel a été entraîné pour chaque ECG de patient dans (Kiranyaz et al. 2016). Il est vérifié par des expériences que ce système présente des performances supérieures dans la détection des battements ventriculaires ectopiques et des battements ectopiques supraventriculaires. Dans (Acharya et al. 2018), des CNN sont utilisés pour détecter la fibrillation auriculaire (A(fib)), la flutter auriculaire (A(fl)), la fibrillation ventriculaire (V-fib), la maladie coronarienne (CAD), l'infarctus du myocarde (MI) et cinq principales classes d'arythmies non mortelles. La même structure CNN profonde de 11 couches a été utilisée. Dans (Acharya et al. 2017), une structure CNN profonde de 9 couches a été utilisée. Le bruit du signal ECG est éliminé, puis alimenté dans le réseau CNN. Les travaux de (Acharya et al. 2018)et (Acharya et al. 2017) ont validé la fiabilité de l'utilisation de différents ensembles d'entraînement pour détecter plusieurs maladies dans le même cadre CNN.Jun et al.(Jun et al. 2018) ont décrit une méthode utilisant un réseau de neurones convolutionnels en 2D pour classer les anomalies dans les ECG. Ils ont comparé leur méthode avec des modèles CNN bien connus tels qu'AlexNet et VGGNet, et ont constaté que leur approche était plus efficace. Pour Séparateurs à vaste marge (SVM) Mondejar-Guerra et al.(Mondéjar-Guerra et al. 2019) ont proposé une méthode de classification automatique des ECG basée sur une combinaison de plusieurs machines à vecteurs de support (SVM). Ils ont utilisé les intervalles de temps entre les battements cardiaques consécutifs ainsi que leur morphologie pour caractériser les ECG, une combinaison de SVM et régression logistique : Dans une approche en deux étapes de fusion de classificateurs en série, Saini et al. (année) ont proposé d'utiliser le rejet des SVM comme mesure de confiance pour rejeter les échantillons ambigus. . Ribeiro et al. (A. H. Ribeiro et al. 2020) ont décrit une méthode de diagnostic automatique des ECG en utilisant un réseau neuronal profond (DNN). Leurs résultats ont montré que le DNN surpassait les médecins résidents en cardiologie dans la reconnaissance des six types d'anomalies ECG Les principales méthodes et les paramètres d'évaluation des performances correspondants sont résumés dans le Tableau I pour permettre une comparaison.

Tab. 2.1 : Résumé des méthodes et des paramètres d'évaluation des performances

Méthode	Ref	Accuracy	sensibilité
CNN	(Kiranyaz et al. 2016)	94.90%	99.13%
CNN	(Acharya et al. 2018)	93.18%	95.32%
CNN	(Acharya et al. 2017)	89.07%	
CNN	(Jun et al. 2018)	99,05%	97,85%
SVM	(Mondéjar-Guerra et al. 2019)	98,84%	
DNN	(A. H. Ribeiro et al. 2020)	99%.	

2.3.4 État de l'art dans l'implémentation de réseaux de neurones sur FPGA pour la classification des arythmies cardiaques

Les FPGA peuvent être un choix approprié pour réaliser une analyse ECG à haute vitesse. Ils fournissent diverses unités matérielles configurables telles que des éléments logiques, de la mémoire vive intégrée (RAM, qui peut également être configurée en mémoire morte en lecture seule, ROM), des broches d'entrée/sortie à haute vitesse, des registres et des blocs de traitement du signal numérique (DSP). En programmant ces unités matérielles, les algorithmes peuvent être mis en œuvre de manière parallèle et exécutés de manière séquentielle. Cela permet aux FPGA d'accélérer efficacement les algorithmes.

Bien que les unités de traitement graphique (GPU) fournissent également une accélération parallèle pour les algorithmes, les FPGA se sont révélés beaucoup plus économes en énergie (Ma et al. 2018), ce qui permet de satisfaire les contraintes de puissance des appareils périphériques.

Dans l'étude de Panigraphy et al.(Panigrahy et al. 2016), ils ont utilisé le FPGA Xilinx Virtex-5 pour développer un système de surveillance de la fréquence cardiaque. Ils ont conçu un nouvel algorithme de détection des pics R et en ont évalué les performances sur la base de données MIT-BIH. De même, Agrawal et Gawali (Agrawal et al. 2017) ont également mis en œuvre un algorithme de détection des pics sur le FPGA Xilinx Virtex-5. Leur approche peut détecter non seulement les pics R, mais aussi les pics P et T. Pour faciliter l'analyse de la variabilité de la fréquence cardiaque (HRV),

Abdullah (Abdullah et al. 2016) ont proposé un système FPGA simple pour détecter les intervalles RR des ECG. Néanmoins, ces études ont simplement utilisé les FPGA pour exécuter leurs algorithmes, sans révéler d'avantage spécifique des FPGA (haute vitesse, grande flexibilité, etc.)

Dans (A. Chen et al. 2020), Chen et al. ont conçu un algorithme de détection des pics R à haute performance. Ils l'ont implémenté sur le FPGA Intel Cyclone-V et ont évalué ses performances d'accélération, ce qui constitue une étude plus complète que celles mentionnées précédemment. Cependant, leur implémentation FPGA n'a accéléré l'algorithme que de 13% par rapport à l'implémentation logicielle utilisant Matlab.

Une accélération FPGA plus efficace pour l'analyse ECG a été proposée par (Gu et al. 2016). Comparée à l'implémentation logicielle utilisant Matlab, elle a atteint une accélération de 6,69 fois sur l'extraction des caractéristiques, y compris la détection des pics R et d'autres points fiduciaires. Cependant, des évaluations quantitatives des performances des algorithmes n'ont pas été réalisées (Gu et al. 2016).

Outre les applications mentionnées précédemment, les FPGA peuvent également être utilisés pour déployer et accélérer des algorithmes basés sur l'intelligence artificielle (en particulier différents réseaux neuronaux tels que MLP et CNN) pour l'analyse ECG.

Wess et al.(Wess et al. 2017) ont utilisé l'analyse en composantes principales (PCA) et un MLP à 3 couches pour détecter les arythmies. Pour déployer le MLP sur un FPGA Xilinx Zynq, ils ont optimisé les fonctions d'activation non linéaires par une approximation linéaire par morceaux. Cette méthode permet de réduire de plus de 80% la quantité de ressources requises. De même,(Zairi et al. 2020) et (Wang et al. 2017) Wang et al. ont également utilisé des MLP à 3 couches pour classifier les arythmies basées sur les battements ECG, mais ils ont utilisé la transformée en ondelettes discrète (DWT) pour l'extraction des caractéristiques. De plus, la DWT et le MLP ont été conjointement déployés sur des FPGA pour mettre en œuvre des systèmes de surveillance complets. Bien que des précisions impressionnantes dans la détection des arythmies aient été rapportées dans ces trois études représentatives basées sur les MLP, seuls des ensembles de données à petite échelle ont été utilisés pour évaluer et optimiser les algorithmes. Cela peut rendre difficile la révélation de la faisabilité réelle de leurs systèmes. De plus, les réseaux neuronaux peu profonds comme les MLP à 3 couches présentent une généralisation inférieure aux réseaux neuronaux profonds en théorie (LeCun et al. 2015).

Pour les modèles plus profonds, Wei et al. (Wei et al. 2021) ont conçu un CNN 1-D à 15 couches pour classifier les battements ECG en cinq classes, et un modèle similaire a également été exploré(Lu et al. 2021). En détail, ils ont développé des unités de traitement génériques de convolution et de pooling, et ont réalisé ces unités de manière multiplexée pour effectuer l'inférence CNN. Cette approche peut être inspirée de la conception d'accélérateurs FPGA pour les CNN en vision par ordinateur(Ma et al. 2018) . Bien qu'elle puisse réduire la consommation de ressources, elle n'est pas suffisamment flexible pour exploiter pleinement les avantages spécifiques des FPGA (Gong et al. 2018). De plus, les communications entre les unités de calcul et les mémoires intégrées ou externes sont trop fréquentes de cette manière, ce qui peut entraîner une consommation d'énergie considérable (Ma et al. 2018).

2.3.5 Analyse des résultats du système de classification

Afin d'évaluer les performances de notre classificateur, on utilise des mesures de performances communes aux systèmes de classificationen général et qui son

Matrice de confusion

Également connue sous le nom de matrice d'erreur, est une représentation sous forme de tableau de dimensions N x N, où N représente le nombre de classes cibles, utilisée pour évaluer les performances d'un modèle de classification. Son rôle est de comparer les valeurs réelles des classes cibles avec les prédictions effectuées par le modèle d'apprentissage automatique. Elle se compose de quatre valeurs clés qui permettent de quantifier les différentes combinaisons de prédictions correctes et incorrectes du modèle. Les vrais positifs (True Positif : TP) correspondent aux cas où le modèle prédit correctement une valeur positive (par exemple, la personne est malade) et que la valeur réelle est effectivement positive. Cela signifie que la prédiction du modèle correspond à la réalité.

Les vrais négatifs (True Négatif : TN) correspondent aux cas où le modèle prédit correctement une valeur négative (par exemple, la personne n'est pas malade) et que la valeur réelle est effectivement négative. Cela indique que le modèle a correctement identifié les cas négatifs.

Les faux positifs (False positif : FP) correspondent aux cas où le modèle prédit à tort une valeur positive alors que la valeur réelle est négative. Il s'agit d'une erreur de type 1, où le modèle a erronément identifié un cas comme positif alors qu'il ne l'était pas réellement.

Les faux négatifs (False Négatif : FN) correspondent aux cas où le modèle prédit à tort une valeur négative alors que la valeur réelle est positive. Il s'agit d'une erreur de type 2, où le modèle a erronément identifié un cas comme négatif alors qu'il était réellement positif.

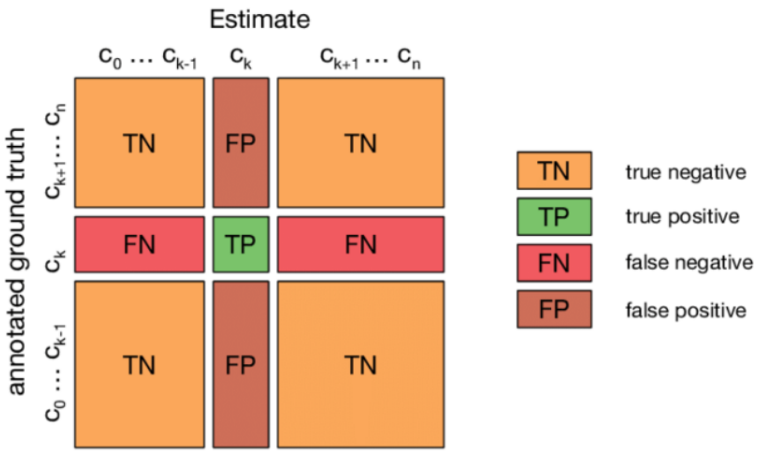


Fig. 2.13 : Matrice de confusion

Exactitude (Accuracy)

Est une métrique que l'on peut utiliser pour évaluer la qualité d'un classiffcateur.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Précision

Le rapport entre le nombre d'observations positives correctement prédites et le total des observations positives prédites par le modèle.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Également appelé sensibilité. Il est calculé en prenant le rapport entre le nombre d'observations positives correctement prédites et le nombre total d'observations réelles de la classe positive.

$$Recall = \frac{TP}{TP + FN}$$

F1 Score

F1 Score représente la moyenne pondérée de la précision et du rappel. Il est calculé en prenant en compte à la fois les faux positifs et les faux négatifs.

$$F1{-}score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

2.3.6 les diverses approches d'implémentation

Les classifieurs CNN (Convolutional Neural Networks) peuvent être implémentés de différentes manières sur le plan logiciel et matériel. Voici quelques approches courantes d'implémentation software et hardware des classifieurs CNN

Implémentation software

- **Utilisation de bibliothèques de deep learning :** Des bibliothèques populaires telles que TensorFlow, Keras, PyTorch et Caffe offrent des fonctionnalités complètes pour implémenter des classifieurs CNN. Elles fournissent des API conviviales pour la définition du modèle, l'entraînement, l'évaluation et l'inférence des classifieurs CNN. Ces bibliothèques prennent en charge des opérations efficaces sur les tenseurs et des optimisations pour accélérer les calculs.

- **Utilisation de frameworks de deep learning :** Les frameworks de deep learning tels que TensorFlow et PyTorch fournissent des fonctionnalités avancées pour la construction et l'entraînement des classifieurs CNN. Ils permettent également d'utiliser des techniques telles que le transfert d'apprentissage (transfer learning) et l'optimisation automatique du graphe de calcul pour accélérer les performances. Ces frameworks offrent également des outils pour la visualisation et l'interprétation des résultats.

- **Utilisation de langages de programmation scientifique :** Des langages de programmation tels que Python avec des bibliothèques scientifiques comme NumPy et SciPy offrent des fonctionnalités pour manipuler les données, effectuer des calculs matriciels

- **Programmation parallèle :** Les CNN peuvent bénéficier de la parallélisation pour accélérer les calculs. Des outils tels que CUDA (Compute Unified Device Architecture) permettent d'exploiter les capacités de calcul parallèle des GPU (Graphics

Processing Units) pour accélérer les opérations de convolution et d'autres calculs intensifs des CNN.

Implémentation hardware

- **Utilisation de processeurs spécialisés :** Certains fabricants de puces développent des processeurs spécialement conçus pour l'accélération des réseaux de neurones, tels que les GPU (Graphics Processing Units) et les TPU (Tensor Processing Units). Ces processeurs fournissent des instructions et des architectures optimisées pour les calculs matriciels et les opérations sur les tenseurs, ce qui permet d'accélérer considérablement les performances des CNN (Y.-H. Chen et al. 2017) (Sze et al. 2017) .
- **Systèmes sur puce (SoC) dédiés à l'IA :** Certains SoC intègrent des unités de traitement spécialisées pour l'intelligence artificielle, telles que les Neural Processing Units (NPU). Ces SoC sont conçus pour offrir des performances élevées tout en réduisant la consommation d'énergie, ce qui les rend adaptés à l'implémentation de CNN sur des appareils mobiles et embarqués (Saha 2021) .
- **Accélérateurs FPGA :** Les Field-Programmable Gate Arrays (FPGA) permettent de créer des circuits personnalisés pour l'accélération des CNN. Les FPGA peuvent être programmés pour exécuter spécifiquement les opérations de convolution et d'autres opérations fréquentes des CNN, offrant ainsi une accélération matérielle sur mesure pour les réseaux de neurones (Philip et al. 2022)(Goodfellow et al. 2016).

Ces approches d'implémentation software et hardware des CNN offrent différentes trade-offs entre performances, flexibilité, consommation d'énergie et coût, ce qui permet d'adapter les choix d'implémentation en fonction des besoins spécifiques de l'application.

Conclusion

La classification des arythmies cardiaques revêt une importance cruciale dans le domaine médical, offrant des outils précieux pour la détection et le diagnostic précoce des anomalies du rythme cardiaque. Dans ce chapitre, nous avons exploré différentes méthodes de classification utilisées dans ce contexte. Ce chapitre a fourni une base solide pour comprendre le système cardiovasculaire, l'ECG, les arythmies et les méthodes de classification des arythmies. Les réseaux de neurones sur FPGA émergent comme une solution prometteuse pour la classification des arythmies, ouvrant de nouvelles perspectives pour des systèmes plus précis, rapides et économes en énergie dans le domaine médical. Cependant, des travaux supplémentaires sont nécessaires pour relever les défis techniques et améliorer davantage les performances de cette approche.

Chapitre 3

Conception et implémentation

3.1 Introduction

Dans ce chapitre, nous allons aborder les différentes étapes de conception et d'implémentation de notre modèle de réseau de neurones convolutifs (CNN). Nous utiliserons plusieurs langages de programmation, tels que Python et C/C++, ainsi que des outils matériels tels que Vivado, Vitis(HLS) et la carte Pynq Z1. L'objectif de ce chapitre est de fournir une vue d'ensemble complète de notre approche, en démontrant sa faisabilité et son efficacité pour relever le défi qui nous est présenté.

La première étape de notre processus consiste à concevoir l'architecture du CNN. Cela comprend le choix du nombre de couches, la taille des filtres, les fonctions d'activation et les autres hyperparamètres. Nous pouvons utiliser des bibliothèques de deep learning telles que PyTorch en Python pour faciliter cette étape.

Après avoir implémenté le modèle en Python, nous pouvons passer à l'étape de l'optimisation et de l'accélération matérielle. Cela implique la traduction du modèle Python en un langage de bas niveau tel que C/C++ pour une exécution plus efficace. Cette étape est cruciale pour obtenir des performances optimales sur des systèmes embarqués ou des dispositifs à ressources limitées.

Une fois que nous avons optimisé notre modèle, nous pouvons passer à l'implémentation matérielle. Vivado est un outil populaire utilisé pour la conception et la mise en œuvre de circuits logiques programmables tels que des FPGA (Field-Programmable Gate Arrays). Nous pouvons utiliser Vivado pour synthétiser notre modèle CNN en utilisant le langage de description matériel VHDL. Cette étape permet de créer un circuit matériel dédié qui exécute le modèle CNN de manière beaucoup plus rapide et efficace que l'exécution logicielle traditionnelle.

Enfin, nous pouvons déployer notre modèle CNN sur la carte Pynq Z1, qui est une plate-forme matérielle utilisant un FPGA. La carte Pynq Z1 offre une interface conviviale pour le développement et l'exécution de modèles de deep learning sur du matériel FPGA. Nous pouvons transférer notre circuit matériel sur la carte Pynq Z1 et l'utiliser pour effectuer des inférences en temps réel avec notre modèle CNN.

3.2 Conception en Python du modèle CNN

3.2.1 Base des données utlisées

Nous utilisons les bases de données PhysioNet MIT-BIH Arrhythmia pour les enregistrements d'ECG étiquetés (Moody et al. 2001)(Bousseljot et al. 1995).

Cet ensemble de données contient les signaux ECG de 47 individus souffrant de divers types d'arythmies. Les enregistrements ont été numérisés à une fréquence d'échantillonnage de 360 Hz, avec une résolution de 11 bits, couvrant une plage de signal de 10 mV. Chaque battement a été annoté par au moins deux cardiologues. Les annotations de cet ensemble de données ont été utilisées pour créer cinq catégories distinctes de battements, conformément aux normes de l'Association pour l'amélioration de la sécurité des patients (AASP) et de l'Association for the Advancement of Medical Instrumentation (AAMI)

EC57.

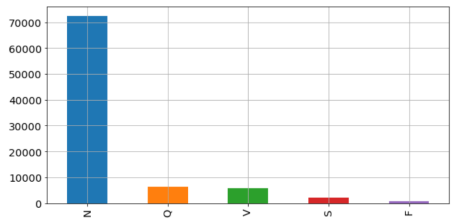
Ces catégories de battements cardiaques sont représentées par les symboles N, S, V, F et Q. Les échantillons significatifs sont respectivement indexés par 0, 1, 2, 3 et 4. Les battements normaux sont symbolisés par N (indice 0), les battements ectopiques supraventriculaires par S (indice 1), les battements ectopiques ventriculaires par V (indice 2), les battements de fusion par F (indice 3), et les battements inconnus par Q (indice 4).

Tab. 3.1 : Un tableau récapitulatif des classes de battements.

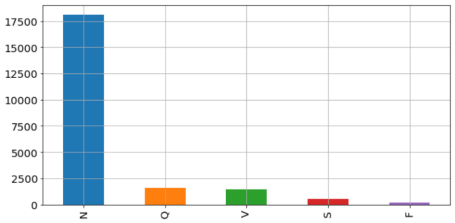
Type	Annotation	Nombre de battements
Battements normaux	N	90589
Battements ectopiques supra ventriculaires	S	2779
Battements ectopiques ventriculaires	V	7236
Battements de fusion	F	803
Battements inconnus	Q	8039

Aperçu sur la base de données

L'ensemble de données sur lequel nous avons travaillé est alors disponible sur Kaggle et est déjà subdivisé en 2 ensembles ; entraînement et test. Ainsi, il se compose de

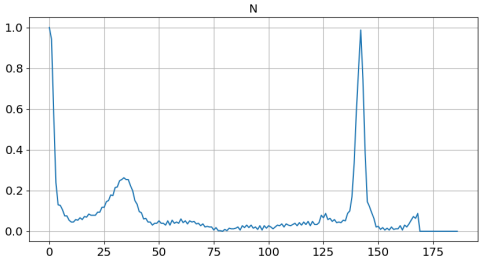


(a) L'ensemble d'entraînement.

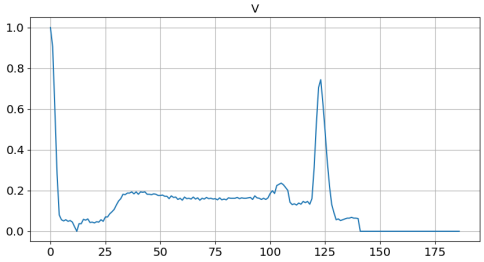


(b) L'ensemble de test.

Présentation de quelques données



(a) Battement normale.



(b) Battement ectopique ventriculaire

3.2.2 Algorithme utilisé

3.2.2.1 Prétraitement ECG et extraction des battements cardiaques

Comme les battements de l'ECG sont les entrées de la méthode proposée, nous suggérons une méthode simple mais efficace pour le prétraitement des signaux ECG et l'extraction des battements. Les étapes utilisées pour extraire les battements à partir d'un signal ECG sont les suivantes (Kachuee et al. 2018) :

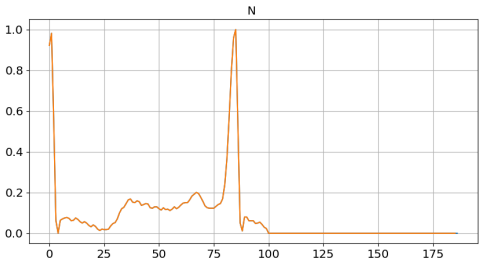
1. Diviser le signal ECG continu en fenêtres de 10 secondes et sélectionner une fenêtre de 10 secondes à partir du signal ECG.
2. Normaliser les valeurs d'amplitude pour qu'elles se situent entre zéro et un.
3. Trouver l'ensemble de tous les maximums locaux en se basant sur les passages par zéro de la première dérivée.
4. Trouver l'ensemble des candidats aux pics R de l'ECG en appliquant un seuil de 0,9 à la valeur normalisée des maximums locaux.
5. Trouver la médiane des intervalles de temps R-R comme période cardiaque nominale de cette fenêtre (T).
6. Pour chaque pic R, sélectionner une partie du signal d'une longueur égale à 1,2T.
7. Remplir chaque partie sélectionnée avec des zéros pour atteindre une longueur prédéfinie fixe.

Il convient de mentionner que cette méthode d'extraction des battements est simple et efficace pour extraire les intervalles R-R à partir de signaux de différentes morphologies. Par exemple, aucune forme de filtrage ou de traitement n'est utilisée, ce qui signifie qu'aucune hypothèse n'est faite sur la morphologie ou le spectre du signal. De plus, tous les battements extraits ont des longueurs identiques, ce qui est essentiel pour les utiliser en tant qu'entrées dans les parties de traitement ultérieures.

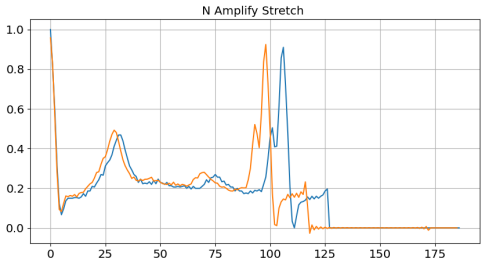
3.2.2.2 Augmentation des données

L'augmentation de données est une technique utilisée pour générer de nouvelles données d'entraînement à partir de données existantes. Elle est largement utilisée dans les réseaux de neurones convolutifs (CNN) pour obtenir une plus grande variabilité des données en entrée. Cette technique agit comme un régularisateur et contribue à réduire le surapprentissage lors de la formation d'un modèle d'apprentissage automatique.

L'augmentation de données est étroitement liée au suréchantillonnage dans l'analyse des données. Elle consiste à appliquer différentes transformations aux échantillons de données, ce qui crée de nouvelles variations et augmente la diversité des données d'entraînement. Les classes d'augmentation de données, telles que les classes "Stretch" et "Amplify" dans le code, encapsulent ces transformations spécifiques. Elles sont utilisées pour augmenter la variabilité des données existantes, ce qui permet d'améliorer la capacité du modèle à généraliser et à traiter différentes variations des données.



(a) Avant l'augmentation de donnée



(b) Aprés l'augmentation de donnée

3.2.2.3 Données d'entraînement

Cet ensemble de données comprend 87554 échantillons et est étiqueté en fonction du type de battement cardiaque (normal ou irrégulier). Étant donné un déséquilibre des classes dans l'ensemble de données, avec un grand nombre d'échantillons normaux, on a sélectionné aléatoirement seulement 5000 échantillons de la classe sur-représentée pour l'ensemble d'entraînement afin de prévenir un biais extrême du modèle.
Nous avons divisé l'ensemble de données en un ensemble de formation, un ensemble de validation et un ensemble de test. et nous avons veillé à ce que chacun d'entre eux présente une répartition similaire des classes.

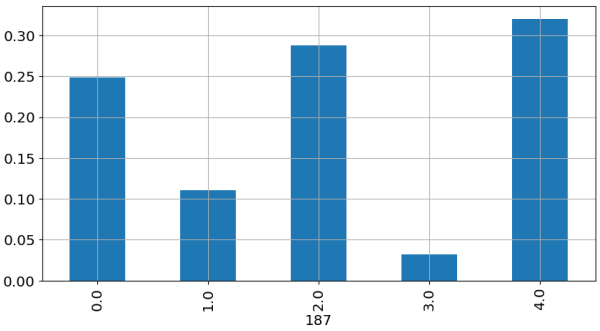


Fig. 3.4 : Distribution des classes de l'ensemble de données amélioré

3.2.2.4 Architecture du classifieur

L'objectif principal de ce réseau formé est la classification des battements cardiaques. la figure 3.5 illustre Organigramme de la méthode proposée :

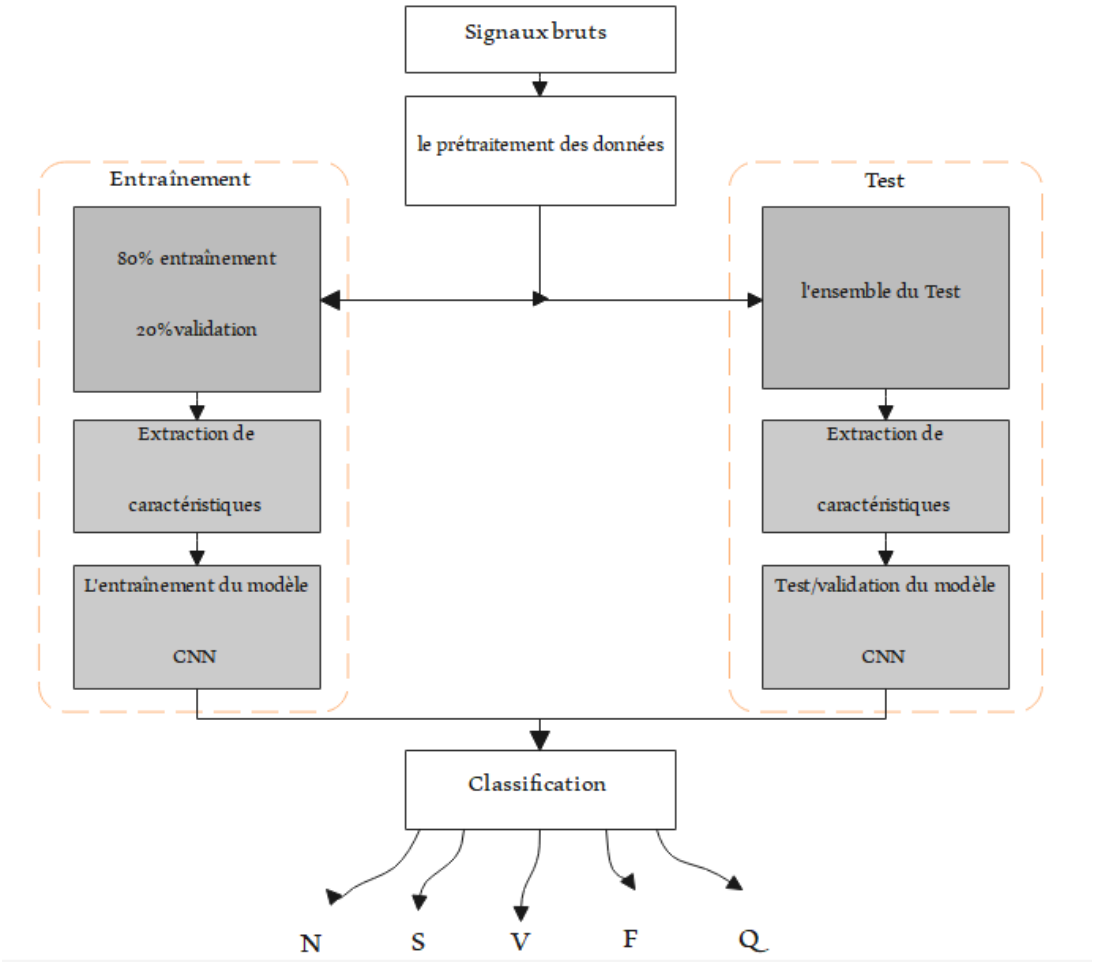


FIG. 3.5 : Organigramme de la méthode proposée

La figure 3.6 illustre l'architecture du réseau proposée pour la tâche de classification des battements cardiaques. Les battements extraits, comme expliqué dans le prétraitment, sont utilisés comme entrées du réseau.
Dans cette architecture, toutes les couches de convolution appliquent une convolution 1D dans le temps et ont chacune 32 noyaux de taille 5. Des couches de max pooling de taille 5 avec un pas (stride) de 2 sont également utilisées dans toutes les couches de pooling.
Le réseau prédicteur se compose de cinq blocs résiduels (residual blocks), suivis de deux couches entièrement connectées avec 32 neurones chacune. Une couche softmax est utilisée en sortie pour prédire la classe de sortie.

Le tableau récapitulatif fournit une vue d'ensemble des différentes couches de l'architecture utilisée pour notre model .

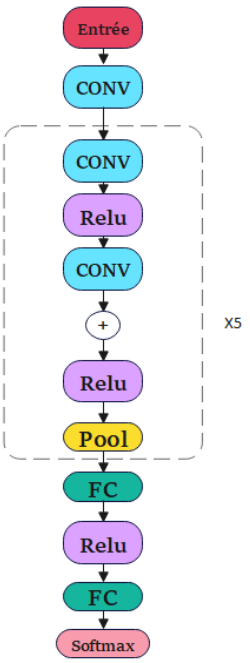


FIG. 3.6 : Architecture du réseau proposé

Modèles de couches	les paramètres des couches
Conv1D	entrée=1, sortie=32, noyau=5, padding=2
MaxPool1D	noyau=5, stride=2
Conv1D	entrée=32, sortie=32, noyau=5, padding=2
MaxPool1D	noyau=5, stride=2
Conv1D	entrée=32, sortie=32, noyau=5, padding=2
MaxPool1D	noyau=5, stride=2
Conv1D	entrée=32, sortie=32, noyau=5, padding=2
MaxPool1D	noyau=5, stride=2
MaxPool1D	noyau=5, stride=2
linéaire	caractéristiques d'entrée=64,caractéristiques de sortie=32
linéaire	caractéristiques d'entrée=32,caractéristiques de sortie=5

3.2.3 Les méthodes d'optimisation

3.2.3.1 Hyperparamètres

Les hyperparamètres sont des paramètres qui régulent le processus d'apprentissage automatique et doivent être définis avant le début de l'apprentissage. Leur sélection appropriée est essentielle pour obtenir de bonnes performances et une généralisation adéquate du modèle.

les hyperparamètres du réseau proposé

Les hyperparamètres	valeurs
Taille de la couche d'entrée	(186×1)
Optimiseur utilisé	Adam
Taux d'apprentissage	$1 * 10^{-3}$
Taille du lot	256
Nombre d'époques	10
le nombre de plis (folds)	5

3.2.3.2 la validation croisée k-fold

La validation croisée divise les données en plusieurs ensembles d'entraînement et de validation, et évalue le modèle sur chaque ensemble de validation pour obtenir une estimation plus fiable des performances du modèle. n-splits indique le nombre de plis utilisés dans cette division.
On remarque que le taux d'apprentissage a été réduit tout au long de la formation. La modulation du taux d'apprentissage est un domaine de recherche actif dans le domaine de l'apprentissage automatique. La perte devrait converger. La perte de l'ensemble d'apprentissage doit être monotone (bien qu'elle puisse ne pas l'être). Une certaine suradaptation est acceptable, mais il est parfois nécessaire d'utiliser une certaine régularisation .

3.2.3.3 Le sous-échantillonnage

le sous-échantillonnage aléatoire est une technique qui consiste à supprimer aléatoirement des exemples de la classe majoritaire afin de réduire le déséquilibre de classe dans l'ensemble de données. Son objectif est d'obtenir une distribution plus équilibrée des classes, ce qui peut améliorer les performances du modèle d'apprentissage automatique.

3.2.4 Le choix de Pytorch

En utilisant le jeu de données et l'architecture de modèle décrits précédemment, nous avons choisi d'utiliser PyTorch en Python pour créer notre classifieur ECG. En exploitant la bibliothèque complète de fonctions prédéfinies et de couches fournies par PyTorch,

nous avons pu construire un modèle performant. Une fois le modèle entraîné, nous avons exporté son fichier pth pour ensuite exporter ses fichiers binaires. Ce modèle PyTorch nous a permis d'établir une base solide pour notre classifieur ECG et nous a fourni un point de départ pour une optimisation ultérieure et une implémentation matérielle.

3.3 Description HDL

Au début de notre projet, nous avons opté pour une approche modulaire dans l'implémentation VHDL de notre modèle CNN. Nous avons divisé le modèle en modules distincts pour chaque couche, notamment la convolution 1D, le max pooling 1D, la transformation linéaire (Dense), l'activation ReLU et la classification finale. Chaque module a été configuré avec des paramètres spécifiques et a interagi en utilisant des signaux d'entrée et de sortie appropriés, ainsi que des signaux de contrôle pour coordonner les opérations entre les différentes couches.

Cependant, lors de l'utilisation de Vivado, nous avons rencontré des problèmes de compatibilité entre les entrées de certaines couches, ce qui a entraîné des erreurs dans la transmission des données et a affecté les performances globales du modèle. Pour remédier à ces problèmes, nous avons décidé d'explorer une autre approche en utilisant le langage de programmation C++. Cette décision nous permettra de continuer nos travaux et de rechercher des solutions plus efficaces pour résoudre les problèmes de compatibilité rencontrés précédemment.

Malgré cette transition vers le langage C++, nous considérons toujours l'approche modulaire initiale comme une perspective prometteuse pour des travaux futurs. Nous espérons pouvoir trouver des solutions appropriées aux problèmes de compatibilité et ainsi améliorer l'efficacité et la réutilisabilité des modules individuels dans d'autres projets ou pour des modifications ultérieures du modèle.

3.4 Implémentation en C/C++

La méthodologie de développement hardware se concentre sur l'implémentation du modèle en C/C++ et son déploiement sur la carte PYNQ-Z1. Cette section se concentre sur les aspects spécifiques liés au matériel, tels que l'optimisation des performances, l'intégration avec la carte.

3.4.1 Choix de l'élément de traitement (PE) pour notre modèle

Un Processing Element (PE) est une unité de calcul utilisée dans les architectures matérielles pour effectuer des opérations spécifiques sur les données. Les PEs sont des composants fondamentaux qui contribuent au traitement et à l'analyse des données dans diverses applications, y compris les réseaux de neurones convolutionnels (CNN)(**bollengier2018**). Différents types de PEs existent, chacun ayant ses propres caractéristiques et avantages (Venkatesh et al. 2019) :

- **Les PEs de calcul général** sont polyvalents et peuvent effectuer une variété d'opérations de calcul, telles que les multiplications, les additions et les opérations arithmétiques. Ils sont adaptés à différentes étapes de traitement du modèle et offrent une flexibilité dans les opérations qu'ils peuvent exécuter.
- **Les PEs spécialisés** sont conçus pour des tâches spécifiques telles que les convolutions, les activations ou d'autres opérations spécifiques à un domaine. Ils sont optimisés pour ces opérations particulières, offrant ainsi des performances supérieures et une efficacité accrue.
- **Les PEs de mémoire** sont responsables de la gestion des données en les stockant dans des tampons d'entrée et de sortie. Ils permettent également les échanges de résultats intermédiaires entre les PEs, favorisant ainsi la communication et la coordination des opérations.
- **les PEs de contrôle** jouent un rôle crucial dans la coordination globale du modèle. Ils supervisent le flux de données entre les différentes couches du modèle et assurent la synchronisation des opérations pour un traitement harmonieux.
- **les PEs polyvalents** mettent l'accent sur la flexibilité et la polyvalence des opérations qu'ils peuvent exécuter. Ils sont capables de s'adapter à différentes étapes de traitement du modèle et peuvent effectuer une variété d'opérations, ce qui les rend polyvalents dans leur utilisation.

Dans notre modèle, nous avons choisi d'utiliser un Processing Element (PE) polyvalent qui combine plusieurs fonctionnalités essentielles pour le traitement des données. Ce PE est capable d'effectuer des opérations de convolution, de traitement des données, de gestion des poids et des biais, d'activation avec la fonction ReLU, ainsi que des opérations de pooling. En regroupant ces différentes fonctionnalités dans un seul PE, nous simplifions l'architecture matérielle et optimisons les performances en réduisant les transferts de données entre les éléments. Ce choix de PE polyvalent nous permet d'exploiter de manière efficace les ressources matérielles disponibles et d'accélérer le traitement des données dans notre modèle.

3.4.2 L'architecture typique d'un accélérateur CNN

L'architecture globale d'un accélérateur CNN sur une plate-forme FPGA embarquée se compose de deux parties principales : le Processing System (PS) et la Programmable Logic (PL). Le PS comprend un CPU et une mémoire externe pour stocker les données et les poids et les bais . Le CPU configure le module de contrôle de l'accélérateur et gère les opérations autres que les couches de convolution. La PL est une puce FPGA programmable qui comprend un tableau de PE pour les calculs de convolution, un module de contrôle et une mémoire interne pour le stockage des données. Les données sont échangées entre les PE pour une réutilisation efficace, tandis que le module de contrôle gère le flux de données. Le fonctionnement de l'accélérateur implique le stockage des données dans la mémoire externe, la configuration du module de contrôle par le CPU, le transfert des données vers la mémoire interne, les calculs effectués par les PE, puis le retour des résultats dans la mémoire externe.

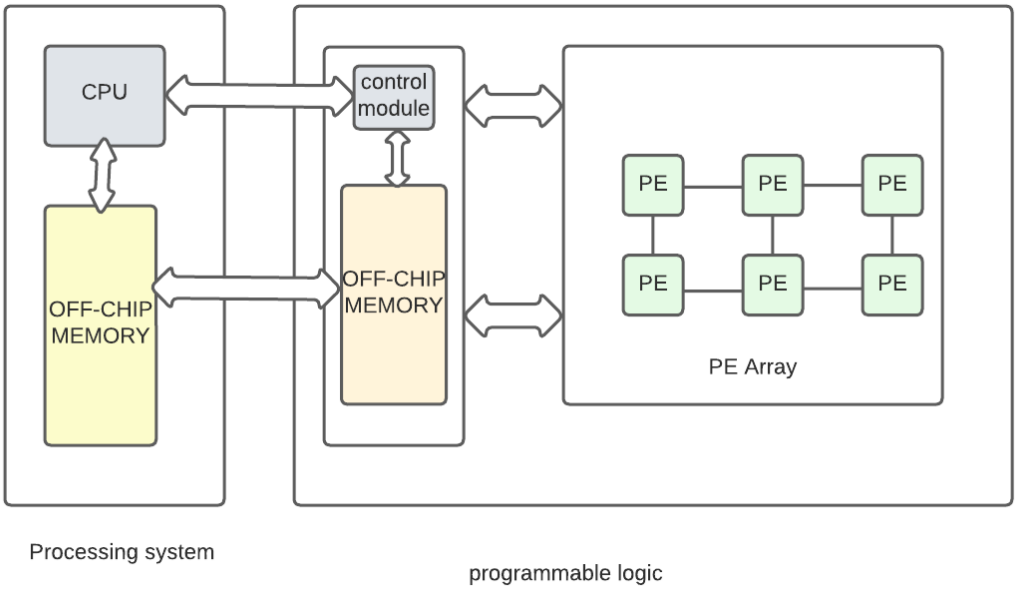


FIG. 3.7 : Architecture typique d'un accélérateur CNN

3.4.3 L'architecture des éléments de traitement (PE)

L'architecture matérielle du modèle utilise des éléments de traitement appelés Process Elements (PE). Chaque PE est une unité de calcul qui effectue différentes étapes de traitement telles que l'extraction des poids et des biais, la convolution, l'activation et les opérations de pooling. Les PEs fonctionnent de manière parallèle et indépendante, ce qui permet de traiter les données simultanément et efficacement à travers les couches du modèle. Les données sont stockées dans des tampons d'entrée et de sortie, ce qui permet les échanges de résultats intermédiaires entre les PEs. Cette architecture parallélisée permet d'accélérer considérablement le traitement des données en exploitant pleinement les ressources matérielles disponibles. Une fois que le traitement est effectué dans les PEs, les résultats finaux sont stockés dans les tampons de sortie des PEs. Ces résultats peuvent être ensuite utilisés pour d'autres étapes du modèle ou pour l'obtention de la prédiction finale.

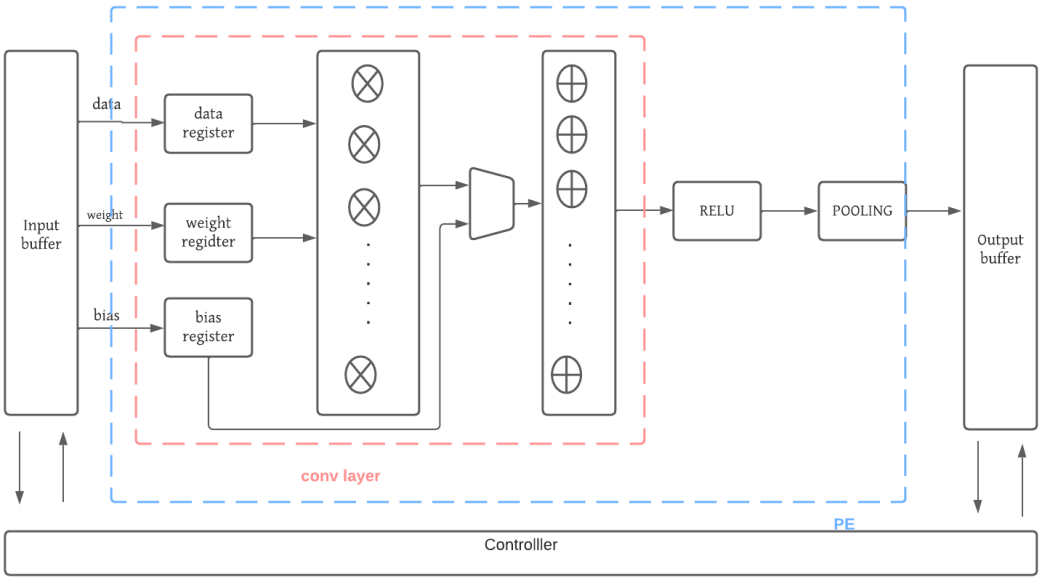


FIG. 3.8 : Architecture matérielle parallélisée avec le Process Element

3.4.4 Implementation en VITIS HLS

L'implémentation du modèle se déroule en plusieurs étapes, en combinant le langage de programmation C/C++ et la plate-forme Vitis HLS.

La description du modèle

Tout d'abord, nous exportons les poids, les biais et les sorties attendues de chaque couche du modèle à partir d'un fichier pth, en les stockant sous forme de fichiers binaires. Ensuite nous créons un fichier de simulation en C++ qui lit les poids, les biais et les sorties attendues de chaque couche à partir du fichier pth, puis nous les stockons dans des tableaux. nous implémentons les fonctions de convolution 1D, de MaxPooling1D et linéaires (dense) en utilisant les poids et les biais extraits des fichiers binaires. Ce processus est répété pour toutes les couches du modèle. En parallèle, nous utilisons la plate-forme Vitis HLS pour ajouter des interfaces AXI en utilisant des pragmas HLS, permettant ainsi l'exécution du modèle sur des FPGA.

- **Les fonctions de convolution 1D**

  Dans le cadre de l'implémentation en C++ de notre modèle, nous avons développé des fonctions de convolution 1D. Nous disposions de quatre de ces fonctions, toutes avec des paramètres similaires : une taille de noyau de 5, un padding de 2, un stride de 1 et 32 canaux de sortie. La première fonction de convolution avait un seul canal d'entrée, tandis que les trois suivantes en avaient 32. Après chaque opération Conv1D, nous avons appliqué une opération MaxPool1D, permettant de réduire les dimensions de l'entrée à chaque étape, ce qui a abouti à une longueur de signal de 187 réduite à 20.

- **Les fonctions de MaxPooling1D**

  Notre approche comprenait quatre couches de max pooling 1D et une couche de max pooling adaptatif à la fin. Cependant, les FPGA ne disposent généralement pas de matériel intégré pour effectuer des opérations de max pooling global, et l'utilisation des ressources FPGA est nécessaire pour activer cette opération. Cela peut entraîner un goulot d'étranglement au niveau de la couche de max pooling adaptatif en cas de contraintes de ressources matérielles, ce qui affecte l'efficacité et la vitesse du réseau neuronal sur le FPGA. Par conséquent, nous avons pris la décision de remplacer la couche de max pooling adaptatif par une couche de MaxPool1D classique. Les cinq fonctions MaxPool1D utilisent toutes une taille de noyau de 5 et un stride de 2.

- **Les fonctions linéaires (Dense)**

  À la fin de notre modèle C++, nous avons utilisé deux fonctions linéaires pour transformer le tableau d'entrée. Cette transformation linéaire implique la multiplication du tableau d'entrée par des poids spécifiques, suivie de l'ajout de biais. En effectuant cette opération, nous avons cherché à reproduire la fonctionnalité de la fonction linéaire de PyTorch. En d'autres termes, nous avons utilisé les poids et les biais des deux couches pour ajuster les valeurs du tableau d'entrée de manière linéaire, afin de produire un tableau de cinq valeurs correspondant à chaque étiquette.

- **Les fonctions d'activation ReLU (Rectified Linear Unit)**

  Afin d'appliquer la fonction d'activation ReLU après chaque couche de convolution et la première couche dense, nous avons utilisé une fonction qui met à zéro toutes les valeurs négatives. Nous avons également envisagé d'utiliser une fonction d'activation softmax à la sortie de la dernière couche entièrement connectée.

3.4.4.1 Validation du modèle

Dans la partie du conception en CNN, nous avons effectué des tests en utilisant un vecteur constant égal à 1, ce qui nous a donné des sorties (output) que nous appelons "expected output" (sortie attendue).
Ainsi, Afin de garantir la fiabilité de notre modèle, nous avons reproduit cette même méthode de test dans C++. Nous avons effectué le test en utilisant le même vecteur d'entrée égal à 1, puis comparé la sortie avec l'output attendu obtenu lors de la phase de conception , comme illustré dans la figure 3.9.

De cette manière, nous avons pu évaluer la cohérence entre les résultats du modèle et les sorties attendues, ce qui a renforcé la fiabilité. Cette approche de validation nous permet de garantir que notre modèle peut produire les résultats souhaités et nous permet de poursuivre notre travail.

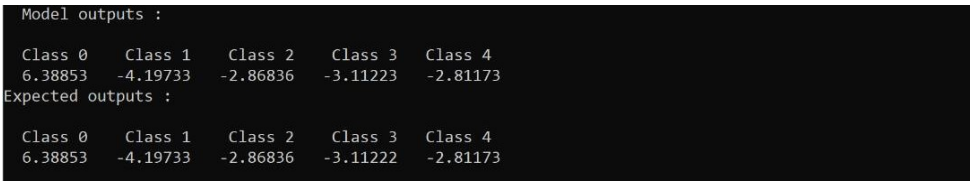


FIG. 3.9 : La fiabilité du modèle

3.4.4.2 La synthèse dans Vitis HLS

Nous avons commencé l'implémentation en C avec Vitis HLS en utilisant le script TCL et le Makefile comme point de départ. La fonction principale, appelée "tiled conv", a été conservée. Elle effectue une opération de convolution en accumulant les résultats partiels des différentes couches pour générer la sortie finale, en respectant le fonctionnement de notre modèle PyTorch.
Dans le but de faciliter l'intégration de notre code avec d'autres composants, nous avons ajouté l'interface AXI en utilisant les pragmas HLS. Ces interfaces AXI jouent un rôle essentiel dans la gestion des transferts de données entre les différents composants du système.
Étant donné le nombre important de fichiers binaires requis pour les données d'entrée, les poids et les biais du modèle, nous avons utilisé les 4 bundles disponibles, chacun étant composé de 4 portsà l'exception d'un seul.

. 3.4.5 La Conception de notre architecture sur vivado suite designe

Nous avons conçu notre architecture à l'aide de Vivado Suite Design, et voici notre conception finale.

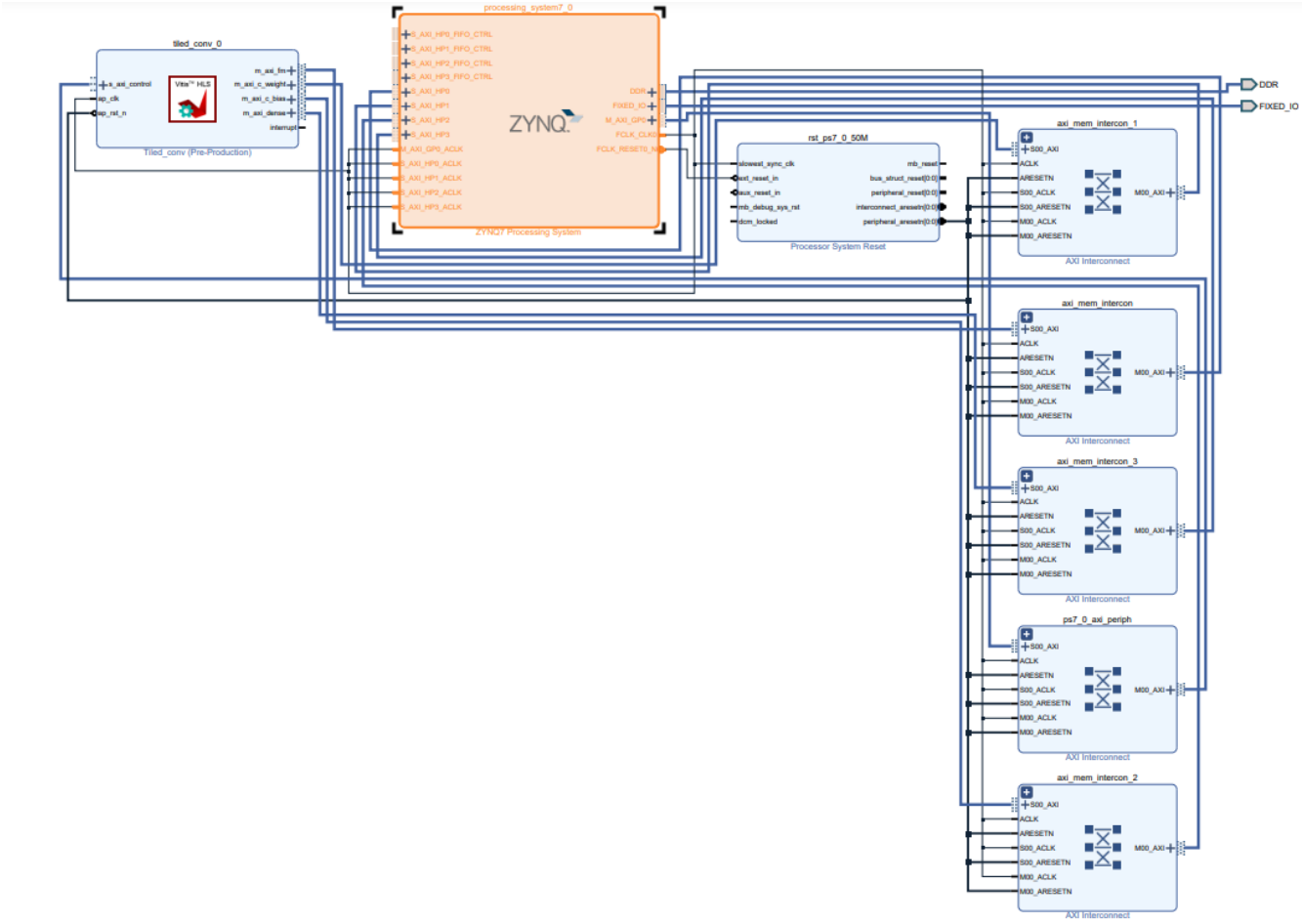


FIG. 3.10 : Conception de notre architecture sur vivado suite designe

Le réinitialisation du système de processeur Le module de réinitialisation du système de processeur (Processor System Reset) Xilinx nous permet d'adapter la conception à nos applications en configurant certains paramètres pour activer ou désactiver des fonctionnalités. Les caractéristiques paramétrables de la conception sont décrites dans les Paramètres de conception du module de réinitialisation du système de processeur(*Processor System Reset* s. d.)

Le composant AXI Interconnect IP Le composant AXI Interconnect IP permet de connecter un ou plusieurs périphériques maîtres AXI, qui sont mappés en mémoire, à un ou plusieurs périphériques esclaves également mappés en mémoire. Les interfaces AXI sont conformes aux spécifications AMBA® AXI version 4 d'ARM, incluant le sous-ensemble d'interfaces de registre de contrôle AXI4-Lite. Cependant, cet IP d'interconnexion ne prend pas en charge les transferts AXI4-Stream. L'AXI Interconnect IP est disponible dans le catalogue IP de Vivado® en tant que noyau pour l'EDK (Embedded Development Toolkit) ou en tant que noyau autonome dans le catalogue IP de CORE Generator™(XILINX INC. 2023a)

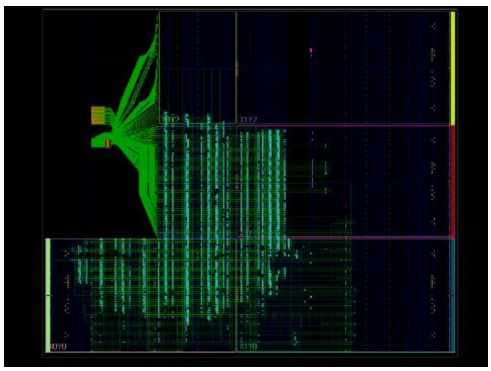
Le système de traitement Zynq-7000 Xilinx Le système de traitement Zynq-7000 Xilinx propose le wrapper IP du système de traitement Zynq®-7000 afin de faciliter la conception et la configuration de vos produits intégrés. Le système de traitement IP représente l'interface logicielle entourant le système de traitement

Zynq-7000. La famille Zynq-7000 comprend un système de traitement intégré (PS) de type système sur puce (SoC) ainsi qu'une unité de logique programmable (PL), offrant une solution SoC polyvalente et extensible sur une seule puce. Le wrapper IP du système de traitement joue le rôle de connexion logique entre le PS et le PL, tout en facilitant l'intégration d'IP personnalisées et intégrées au système de traitement à l'aide de l'outil d'intégration IP Vivado²®(Xilinx Inc. 2023b).

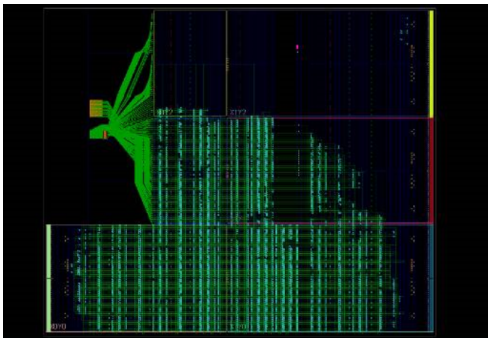
Le bloc Tiled-conv (Pre-Production) Le bloc Tiled-conv (Pre-Production) est un IP core développé à l'aide de Vitis HLS. Il représente une implémentation matérielle optimisée de la fonction de convolution adaptée à notre modèle PyTorch. Grâce à l'utilisation de pragmas HLS et à l'ajout d'interfaces AXI, nous avons pu exploiter efficacement les ressources matérielles disponibles et améliorer les performances du bloc. Cette optimisation nous a permis d'obtenir une latence réduite et une utilisation efficace des ressources, contribuant ainsi à l'accélération du traitement des données.

3.4.6 Analyse de la conception physique sur le Plateau Logique (PL)

Deux modèles de conception physique sur le Plateau Logique (PL) sont présentés, illustrant comment l'espace sur le PL a été utilisé dans ces deux conceptions. Chaque point turquoise représente l'utilisation d'une ressource spécifique. Les deux figures correspondent aux modèles a : non optimisé et b : optimisé. Elles montrent la répartition des conceptions sur le PL du Zynq-7000. Les parties colorées indiquent quels composants du PL ont été utilisés dans chaque modèle respectif.



(a) Conception non optimisée sur PL



(b) Conception optimisée sur PL

3.5 Implémentation en PYNQ Z1

3.5.1 Choix de la carte PYNQ Z1

La carte Pynq-Z1 se distingue par ses avantages significatifs par rapport aux FPGA traditionnels grâce à son SoC Zynq-7000 intégrant un processeur ARM Cortex-A9

et une logique programmable FPGA. Cette combinaison offre une grande flexibilité pour le développement de systèmes embarqués, permettant l'exécution de code logiciel et matériel sur une même plateforme. La compatibilité avec le framework PYNQ simplifie la programmation avec une interface conviviale basée sur Python. La Pynq-Z1 offre ainsi une solution complète pour le développement rapide d'applications matérielles et logicielles, en exploitant la puissance du FPGA et la simplicité de la programmation en Python.

3.5.2 Préparation de la carte Pynq-Z1

Nous utilisons le logiciel Win32 Disk Image pour graver l'image v2.4 téléchargeable sur internet, spécialement conçue pour la carte FPGA Pynq-Z1, sur une carte SD. Cette image contient tous les packages et les informations nécessaires pour garantir le bon fonctionnement de la carte.

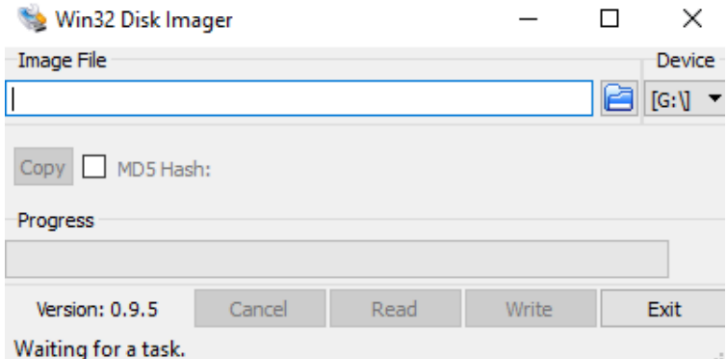


Fig. 3.12 : Win 32 Disk Imager

Ensuite, nous connectons le PYNQ Z2 au routeur avec le câble Ethernet, puis à l'ordinateur avec le câble Micro-USB.

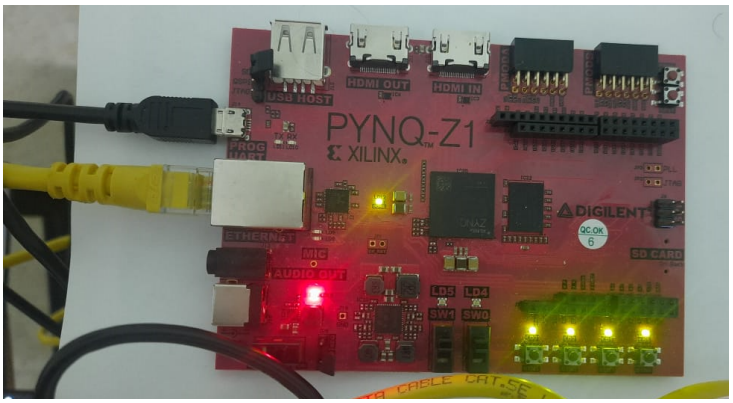


Fig. 3.13 : Carte PYNQZ1 connectée au pc

Après avoir gravé l'image sur la carte SD, nous utilisons la plateforme Jupyter pour accéder au contenu de la carte SD et programmer le FPGA Pynq-Z1. Nous saisissons le mot de passe "XILINX" pour accéder à la plateforme.

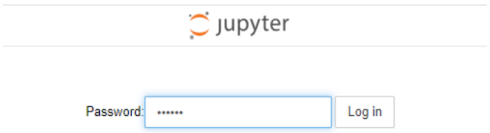


FIG. 3.14 : JUPYTER

Après avoir accédé à la plateforme, nous avons démarré notre implémentation en utilisant le bitstream généré avec Vivado, qui fournit des rapports détaillés sur l'utilisation des ressources. En combinant les fichiers bitstream et de configuration matérielle (hwh) avec le notebook Jupyter, nous avons pu exécuter notre implémentation sur la carte FPGA. Pour préparer les données, nous avons utilisé un code existant pour convertir les valeurs en virgule flottante en valeurs en virgule fixe. Ensuite, à l'aide du notebook Jupyter, nous avons lu les fichiers binaires contenant les poids et les biais nécessaires à notre modèle. Après avoir chargé ces données dans les tampons, nous avons lancé notre implémentation sur la carte Pynq-Z1 et mesuré la latence en temps réel. Cette approche nous a permis d'intégrer efficacement notre modèle sur la carte FPGA et d'évaluer ses performances.

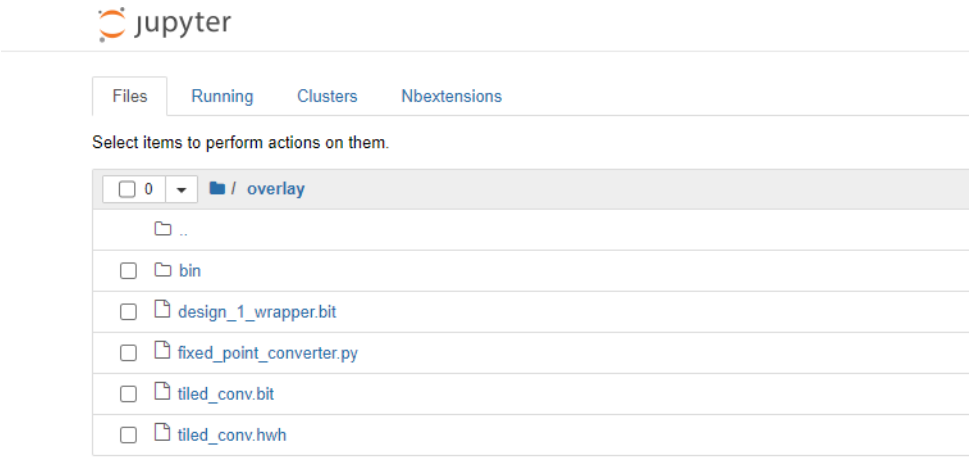
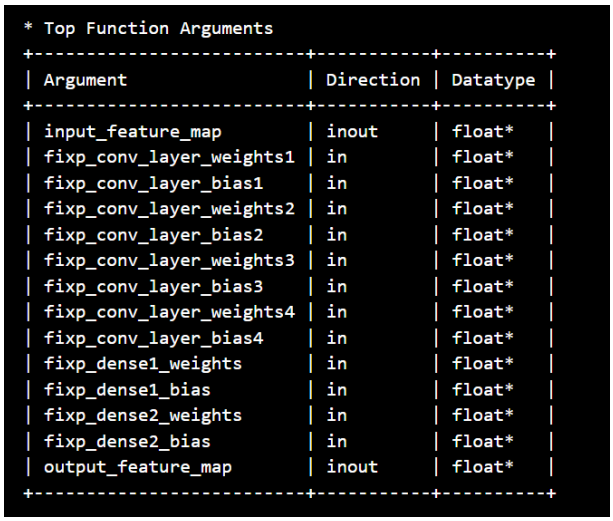


FIG. 3.15 : Capture des fichiers ajoutés

3.5.3 La représentation en Virgule Fixe

Dans notre modèle, les données sont principalement représentées sous forme de pointeurs vers des tableaux de nombres flottants (float*). Ces pointeurs permettent d'accéder efficacement aux tableaux sans avoir à les copier intégralement. Les nombres flottants sont utilisés pour représenter des valeurs décimales, ce qui est courant dans les calculs et la manipulation de données au sein des réseaux neuronaux. L'utilisation de pointeurs dans notre modèle améliore l'efficacité du traitement des données en évitant les opérations de copie coûteuses et en permettant une manipulation directe des valeurs dans les tableaux. Cela contribue à optimiser les performances

globales du modèle.



```
* Top Function Arguments
+--------------------------+-----------+----------+
| Argument                 | Direction | Datatype |
+--------------------------+-----------+----------+
| input_feature_map        | inout     | float*   |
| fixp_conv_layer_weights1 | in        | float*   |
| fixp_conv_layer_bias1    | in        | float*   |
| fixp_conv_layer_weights2 | in        | float*   |
| fixp_conv_layer_bias2    | in        | float*   |
| fixp_conv_layer_weights3 | in        | float*   |
| fixp_conv_layer_bias3    | in        | float*   |
| fixp_conv_layer_weights4 | in        | float*   |
| fixp_conv_layer_bias4    | in        | float*   |
| fixp_dense1_weights      | in        | float*   |
| fixp_dense1_bias         | in        | float*   |
| fixp_dense2_weights      | in        | float*   |
| fixp_dense2_bias         | in        | float*   |
| output_feature_map       | inout     | float*   |
+--------------------------+-----------+----------+
```

FIG. 3.16 : Types de données

En outre, une fonction de conversion en virgule fixe a été développée pour l'implémentation sur la carte pynq-Z1. Nous avons créé une fonction appelée "fixed point conversion" qui permet de convertir les tableaux de données en une représentation en virgule fixe. Cette fonction prend en compte les paramètres spécifiés tels que la largeur de la partie entière et la présence de nombres signés. Elle ajuste les valeurs en fonction des limites spécifiées et les arrondit avant de les stocker dans le tableau de destination.

La fonction de conversion en virgule fixe est bénéfique pour plusieurs raisons. Tout d'abord, elle permet de représenter des nombres réels avec une précision définie pour la partie entière et la partie fractionnaire, ce qui est important pour les calculs dans les réseaux neuronaux. De plus, cette représentation en virgule fixe permet de réduire l'utilisation des ressources matérielles, la consommation d'énergie et la latence, ce qui est particulièrement avantageux dans les environnements FPGA où les ressources sont limitées.

3.6 Conclusion

Ce chapitre a démontré la faisabilité et l'efficacité de notre approche pour relever le défi qui nous a été présenté. La combinaison des langages de programmation, des outils matériels et des techniques d'optimisation nous a permis de concevoir, implémenter et déployer avec succès notre modèle CNN.

```python
import numpy as np

def to_fixed_point(dst, src, *, width=None, iwidth, signed=True):
    if width is None:
        width = dst.dtype.itemsize * 8

    fwidth = width - iwidth
    epsilon = 1.0 / (2.0 ** fwidth)
    min_ = -1.0 * (2.0 ** (iwidth - 1)) if signed else 0.0
    max_ = (2.0 ** (iwidth - (1 if signed else 0))) - epsilon

    src = np.copy(src)
    src = src.reshape(dst.shape)
    src[src < min_] = min_
    src[src > max_] = max_
    if signed:
        src[src < 0] += (2 ** iwidth)
    dst[:] = np.around(src * (2.0 ** fwidth)).astype(dst.dtype)
```

FIG. 3.17 : fonction de conversion en virgule fixe

Chapitre 4

Résultats et évaluation

4.1 Introduction

Dans ce chapitre , nous présentons les résultats obtenus et l'évaluation de notre modèle de classification des arythmies cardiaques. Nous décrivons la conception du modèle CNN en utilisant Python, en ajustant les hyperparamètres pour éviter le surajustement. La courbe de perte d'entraînement et de validation du modèle est également présentée.

Nous évaluons les performances du modèle à l'aide de mesures telles que la matrice de confusion, l'exactitude, la précision, le rappel et le score F1. Nous comparons également notre modèle avec d'autres méthodes d'apprentissage automatique telles que les SVM, les KNN et la régression logistique. Nous constatons que notre modèle CNN dépasse ces méthodes en termes de précision et d'efficacité dans la classification des arythmies cardiaques.

Enfin, nous présentons les résultats de la synthèse dans Vitis HLS et l'implémentation sur Vivado. Nous mettons en évidence les améliorations apportées par les optimisations, tout en tenant compte des contraintes de ressources .

4.2 Conception en Python du modèle CNN

Pour l'entraînement, nous avons utilisé plus de 5 plis (séparations) en utilisant la méthode KFold avec 10 époques chacun et un taux d'apprentissage de 1e-3. L'optimiseur Adam a été choisi pour sa convergence plus rapide par rapport à la descente de gradient stochastique. Nous avons effectué un ajustement des hyperparamètres en ce qui concerne le nombre de plis et d'époques. Lorsque le nombre de plis est trop faible ou le nombre d'époques est trop élevé, le modèle a tendance à surajuster l'ensemble d'entraînement et à avoir de mauvaises performances sur l'ensemble de test.

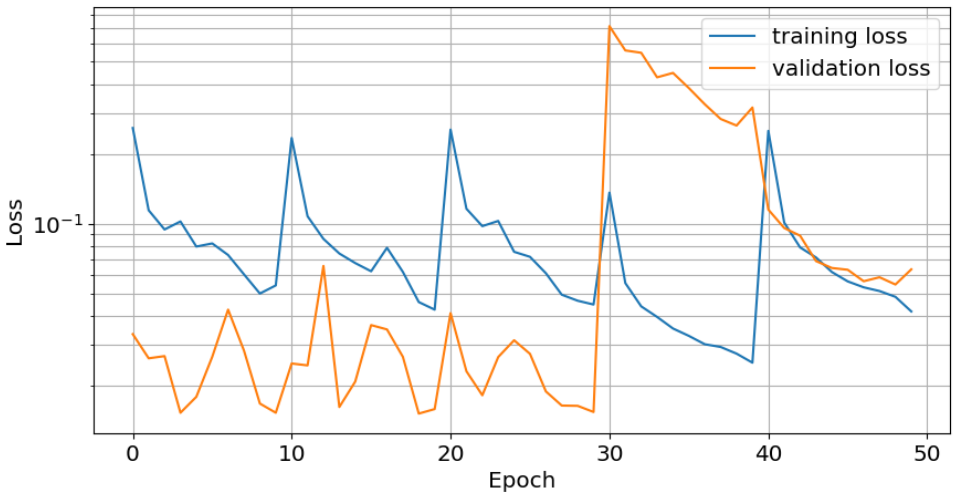


Fig. 4.1 : Perte d'entraînement et de validation du modèle.

La figure 4.15 présente à la fois la perte d'entraînement et la perte de validation de

notre modèle, on utilise la fonction de perte CrossEntropyLoss, qui est couramment utilisée pour les problèmes de classification multiclasse. Notre objectif en créant et en entraînant ce modèle n'était pas d'optimiser sa précision, mais plutôt de le faire fonctionner sur une FPGA avec une latence réduite et sans utiliser toutes les ressources de la carte.

4.2.1 Les mesures de performances

4.2.1.1 Matrice de confusion

Prediction / Label	0.0	1.0	2.0	3.0	4.0
0.0	18102.0	13.0	2.0	1.0	NaN
1.0	116.0	433.0	7.0	NaN	NaN
2.0	29.0	2.0	1407.0	9.0	1.0
3.0	20.0	NaN	16.0	126.0	NaN
4.0	8.0	NaN	2.0	NaN	1598.0

(a) Matrice de confusion sur les données du test.

Prediction / Label	0.0	1.0	2.0	3.0	4.0
0.0	18027.0	26.0	23.0	2.0	37.0
1.0	181.0	358.0	14.0	NaN	3.0
2.0	35.0	1.0	1396.0	12.0	4.0
3.0	25.0	NaN	13.0	124.0	NaN
4.0	12.0	NaN	6.0	NaN	1590.0

(b) Matrice de confusion sur les données de validation.

Fig. 4.2 : matrices de confusion du modèle

4.2.1.2 Métriques de mesure

Nous avons exécuté ce modèle sur notre ensemble de test afin d'évaluer sa précision, sa précision, son rappel et son score F1. Bien que nous n'ayons pas cherché à obtenir un modèle parfait, la figure 4 montre l'exactitude, le rappel et le score F1 qui étaient suffisamment bons pour atteindre l'objectif du projet.

	precision	recall	f1-score	support
0.0	1.00	0.99	0.99	18275
1.0	0.78	0.97	0.86	448
2.0	0.97	0.98	0.98	1434
3.0	0.78	0.93	0.85	136
4.0	0.99	1.00	1.00	1599
accuracy			0.99	21892
macro avg	0.90	0.97	0.94	21892
weighted avg	0.99	0.99	0.99	21892

(a) Exactitude du modèle, précision, rappel et score F1 pour l'ensemble de test.

	precision	recall	f1-score	support
0	1.00	0.99	0.99	18280
1	0.64	0.93	0.76	385
2	0.96	0.96	0.96	1452
3	0.77	0.90	0.83	138
4	0.99	0.97	0.98	1634
accuracy			0.98	21889
macro avg	0.87	0.95	0.90	21889
weighted avg	0.98	0.98	0.98	21889

(b) Exactitude du modèle, précision, rappel et score F1 pour l'ensemble de validation.

Comparaison entre les résultats du test et de la validation :

En analysant les résultats du test et de la validation, nous pouvons observer des performances similaires et élevées pour la classification des arythmies cardiaques.

Pour l'ensemble de validation, nous avons obtenu une précision moyenne pondérée de 0.99, un rappel moyen pondéré de 0.99 et un score F1 moyen pondéré de 0.99. Ces résultats indiquent que notre modèle est capable de classifier de manière précise et équilibrée les différentes classes d'arythmies cardiaques lors de la phase de validation.

De même, pour l'ensemble de test, nous avons obtenu une précision moyenne pondérée de 0.99, un rappel moyen pondéré de 0.99 et un score F1 moyen pondéré de 0.99. Cela démontre que notre modèle maintient des performances cohérentes lorsqu'il est évalué sur des données indépendantes.

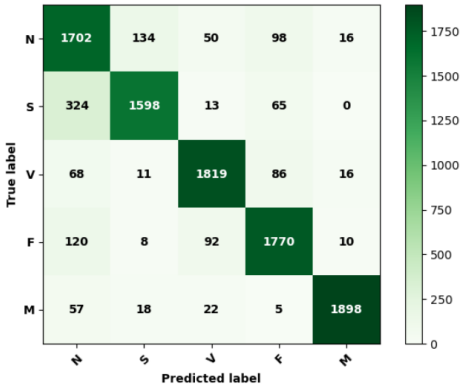
L'exactitude globale pour le test était de 0.99, tandis que pour la validation, elle était également de 0.98. Ces résultats soulignent la capacité de notre modèle à généraliser et à classifier avec précision les arythmies cardiaques, quel que soit l'ensemble de données utilisé.

En résumé, notre modèle a démontré des performances remarquables et cohérentes lors de la classification des arythmies cardiaques, que ce soit lors de la phase de test ou de validation. Ces résultats renforcent la fiabilité et l'efficacité de notre modèle pour la détection précise des différents types d'arythmies cardiaques.

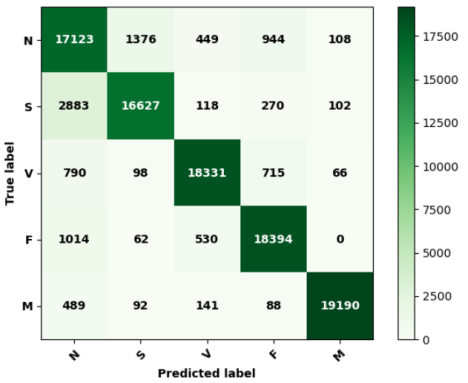
4.2.2 Comparaison entre les diffrentes modèles de classification

Dans le but de classer les arythmies cardiaques, diverses méthodes sont mises en œuvre, parmi lesquelles figurent des techniques d'apprentissage automatique telles que les Machines à Vecteurs de Support (SVM), les K-Nearest Neighbors (KNN) et la régression logistique, ainsi que des méthodes d'apprentissage profond telles que le Réseau de Neurones Convolutif (CNN) présenté dans notre étude. Par conséquent, cette section vise à effectuer une comparaison entre ces deux approches en ce qui concerne la classification des arythmies cardiaques. Nous débuterons en abordant les méthodes d'apprentissage automatique, lesquelles ont été entraînées sur le même ensemble de données, nous permettant ainsi d'obtenir les différentes performances associées à chaque méthode.

SVM



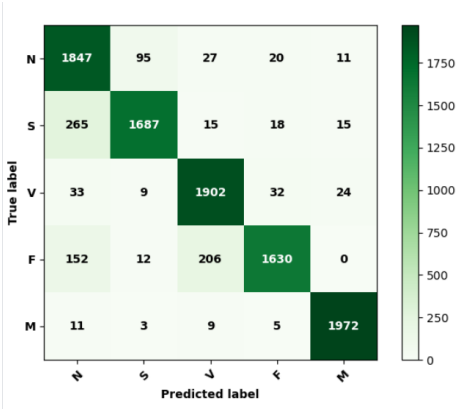
(a) Matrice de confusion sur les données du test.



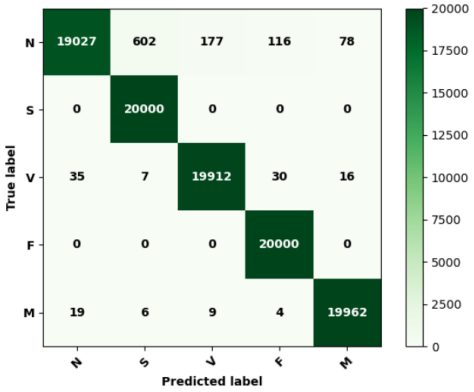
(b) Matrice de confusion sur les données d'entraînement.

FIG. 4.4 : Comparaison des matrices de confusion pour les données d'entraînement et de test(SVM).

KNN



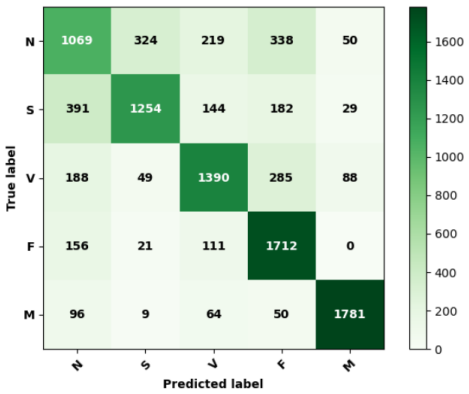
(a) Matrice de confusion sur les données du test.



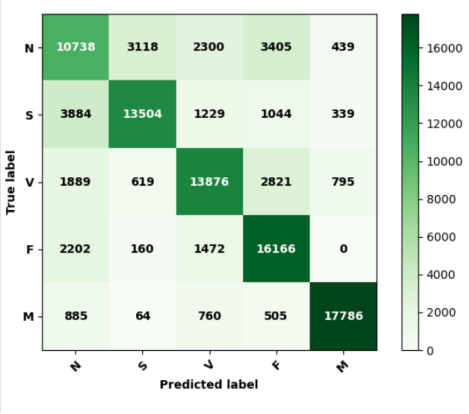
(b) Matrice de confusion sur les données d'entraînement.

FIG. 4.5 : Comparaison des matrices de confusion pour les données d'entraînement et de test(KNN).

Régression logistique



(a) Matrice de confusion sur les données du test.



(b) Matrice de confusion sur les données d'entraînement.

FIG. 4.6 : Comparaison des matrices de confusion pour les données d'entraînement et de test(LR).

	Model	Accuracy (Train)	Precision (Train)	Recall (Train)	F1 Score (Train)	Accuracy (Test)	Precision (Test)	Recall (Test)	F1 Score (Test)
0	SVM	89.665	90.048795	89.665	89.765029	87.87	88.339278	87.87	87.971144
1	KNN	98.901	98.919032	98.901	98.894288	90.38	90.932313	90.38	90.382222
2	Logistic Regression	72.070	72.434619	72.070	72.059573	72.06	72.431455	72.06	71.874155

FIG. 4.7 : la comparaison entre les trois méthodes

À la suite de la comparaison de ces méthodes avec notre modèle, il est évident que les méthodes d'apprentissage profond surpassent les méthodes d'apprentissage automatique. Par conséquent, notre choix s'est orienté vers l'utilisation du CNN pour la classification des arythmies cardiaques. Cette décision est motivée par les performances supérieures observées avec cette approche en termes de précision et d'efficacité dans la catégorisation des différentes anomalies cardiaques.

4.3 Test et simulation

Pour assurer la précision de notre modèle en C++ dans la prédiction des cinq classes d'arythmies cardiaques, nous avons effectué des tests en utilisant des vecteurs d'entrée provenant de la base de données dont les classes étaient connues. Nous avons ensuite comparé les résultats obtenus.
En ce qui concerne la classe N, étiquetée 0, on observe que la valeur maximale est associée à la classe 0, ce qui confirme que le modèle effectue correctement la prédiction , ainsi que pour les autres classes .

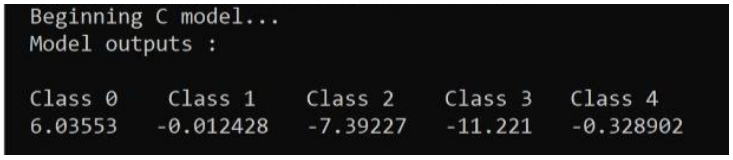


FIG. 4.8 : Prediction obtenue par le modèle sur un signal d'entrée de la classe 0

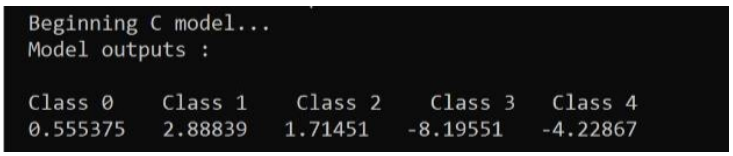


FIG. 4.9 : Prediction obtenue par le modèle sur un signal d'entrée de la classe 1

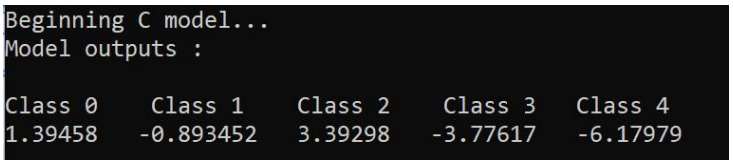


FIG. 4.10 : Prediction obtenue par le modèle sur un signal d'entrée de la classe 2

```
Beginning C model...
Model outputs :

Class 0    Class 1    Class 2    Class 3    Class 4
-4.07091   -2.03966   2.94878    -6.76132   3.86464
```

FIG. 4.11 : Prediction obtenue par le modèle sur un signal d'entrée de la classe 4

4.4 Résultats de la synthèse dans Vitis HLS et l'implémentation sur Vivado :

4.4.1 La synthèse pour le modèle non optimisé

L'utilisation des ressources dans Vitis HLS Selon les estimations fournies par Vitis HLS, la latence prévue était de 114 ms, avec une utilisation des ressources de 36% pour les blocs RAM (BRAM), 2% pour les blocs de traitement numérique du signal (DSP), 8% pour les bascules (FF) et 28% pour les tables de recherche (LUT).

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+---------+---------+---------+
    |   Latency (cycles)  |   Latency (absolute)  |       Interval      | Pipeline|
    |    min  |    max  |    min   |    max   |    min  |    max  |   Type  |
    +---------+---------+----------+----------+---------+---------+---------+
    | 11388579| 11388579| 0.114 sec| 0.114 sec| 11388580| 11388580|       no|
    +---------+---------+----------+----------+---------+---------+---------+
```

FIG. 4.12 : la latence non optimisée

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-----+--------+-------+-----+
|       Name      | BRAM_18K| DSP |   FF   |  LUT  | URAM|
+-----------------+---------+-----+--------+-------+-----+
|DSP              |        -|    -|       -|      -|    -|
|Expression       |        -|    -|       0|    639|    -|
|FIFO             |        -|    -|       -|      -|    -|
|Instance         |        0|    5|    7230|  12460|    -|
|Memory           |      103|    -|      32|      3|    0|
|Multiplexer      |        -|    -|       -|   2251|    -|
|Register         |        -|    -|    2045|      -|    -|
+-----------------+---------+-----+--------+-------+-----+
|Total            |      103|    5|    9307|  15353|    0|
+-----------------+---------+-----+--------+-------+-----+
|Available        |      280|  220|  106400|  53200|    0|
+-----------------+---------+-----+--------+-------+-----+
|Utilization (%)  |       36|    2|       8|     28|    0|
+-----------------+---------+-----+--------+-------+-----+
```

FIG. 4.13 : L'utilisation des ressources dans Vitis HLS

4.4.2 La synthèse pour le modèle optimisé

L'optimisation de la synthèse a apporté des améliorations notables en termes de performances et d'efficacité. En utilisant des techniques telles que le pipelining et

la partition des tableaux, nous avons pu réduire la latence et optimiser l'utilisation des ressources matérielles disponibles. Ces optimisations ont permis d'accélérer l'exécution du modèle et de maximiser l'efficacité des ressources utilisées. En conséquence, nous avons obtenu des résultats améliorés en termes de temps d'exécution et d'utilisation des ressources.

4.4.2.1 L'optimisation des fonctions de convolution

Dans l'objectif d'améliorer les performances des couches de convolution, qui étaient les parties les plus lentes du système, nous avons entrepris une série d'actions pour les optimiser. Suite à l'optimisation des fonctions Conv1D, nous avons obtenu une latence de 4,28 ms avec Vitis, correspondant à une accélération significative de 26.6 fois. De plus, nous avons veillé à une utilisation efficace des ressources, avec seulement 46 % de BRAM, 23% de DSP, 23 % de FF et 52 % de LUT. en utilisant le pipelining et en partitionnant les tableaux, nous avons considérablement accéléré les calculs de convolution.

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+--------+--------+---------+
    |  Latency (cycles) |  Latency (absolute) |     Interval    | Pipeline|
    |   min   |   max   |    min   |    max   |   min  |   max  |   Type  |
    +---------+---------+----------+----------+--------+--------+---------+
    |   428533|   428533|  4.285 ms|  4.285 ms|  428534|  428534|       no|
    +---------+---------+----------+----------+--------+--------+---------+
```

Fig. 4.14 : La latence de la première optimisation

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-----+--------+-------+-----+
|       Name      | BRAM_18K| DSP |   FF   |  LUT  | URAM|
+-----------------+---------+-----+--------+-------+-----+
|DSP              |        -|    -|       -|      -|    -|
|Expression       |        -|    -|       0|     12|    -|
|FIFO             |        -|    -|       -|      -|    -|
|Instance         |        0|   52|   19873|  23870|    -|
|Memory           |      129|    -|      32|      3|    0|
|Multiplexer      |        -|    -|       -|   4045|    -|
|Register         |        -|    -|    5122|      -|    -|
+-----------------+---------+-----+--------+-------+-----+
|Total            |      129|   52|   25027|  27930|    0|
+-----------------+---------+-----+--------+-------+-----+
|Available        |      280|  220|  106400|  53200|    0|
+-----------------+---------+-----+--------+-------+-----+
|Utilization (%)  |       46|   23|      23|     52|    0|
+-----------------+---------+-----+--------+-------+-----+
```

Fig. 4.15 : L'utilisation des ressources pour conv optimisation

4.4.2.2 L'optimisation des fonctions de convolution et de la fonction Dense1

Lorsque nous avons optimisé simultanément la première fonction dense avec les fonctions de convolution, nous avons réussi à réduire la latence de 4,28 ms à 4,06 ms, ce qui représente une amélioration significative. Cependant, cette amélioration de la latence s'est accompagnée d'un dépassement notable des ressources DSP et LUT. Le dépassement des ressources matérielles se produit lorsque nous utilisons plus de ressources que ce qui était initialement prévu ou disponible. Cela peut avoir des répercussions sur d'autres parties du système, telles que la consommation d'énergie et la capacité de traitement des autres tâches.

Dans notre cas, le dépassement des ressources DSP et LUT peut influencer d'autres fonctions du système. Il est donc important de prendre en compte ces contraintes lors de l'évaluation des performances globales. En prenant en considération ces facteurs, nous avons décidé de nous concentrer principalement sur la première optimisation, qui offre une meilleure latence tout en maintenant une utilisation raisonnable des ressources. Ainsi, nous avons privilégié l'équilibre entre la latence et l'utilisation des ressources dans notre approche d'optimisation.

```
+ Latency:
    * Summary:
    +---------+---------+----------+----------+--------+--------+---------+
    |  Latency (cycles) |  Latency (absolute) |     Interval    | Pipeline|
    |   min   |   max   |    min   |    max   |   min  |   max  |   Type  |
    +---------+---------+----------+----------+--------+--------+---------+
    |   406236|   406236|  4.062 ms|  4.062 ms|  406237|  406237|       no|
    +---------+---------+----------+----------+--------+--------+---------+
```

Fig. 4.16 : La latence de la deuxiéme optimisation

4.4.3 résultats de l'implémentation sur Vivado

En examinant l'utilisation des ressources, on observe des différences entre les estimations de Vitis et les résultats réels de Vivado. Les figures a et b présentent le rapport de Vivado sur l'utilisation des ressources. Pour l'implémentation de référence, les ressources embarquées ont été utilisées à hauteur de 13,45% pour les LUT, 10,75% pour les FF, 38,57% pour les BRAM et 2,27% pour les DSP. Globalement, les estimations étaient généralement proches des valeurs réelles. On constate une similarité pour l'utilisation des ressources embarquées de l'implémentation optimisée, avec des taux respectifs de 24,51% pour les LUT, 20,86% pour les FF, 42,50% pour les BRAM et 27,73% pour les DSP. Cette proximité entre l'utilisation réelle des ressources et les estimations suggère que l'exploration de l'espace de conception réalisée avec Vitis HLS a fourni une bonne représentation de l'utilisation des ressources.En ce qui concerne la consommation d'énergie, la conception non optimisée

* Summary:

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	12	-
FIFO	-	-	-	-	-
Instance	0	345	47008	65396	-
Memory	189	-	32	3	0
Multiplexer	-	-	-	6562	-
Register	-	-	5122	-	-
Total	189	345	52162	71973	0
Available	280	220	106400	53200	0
Utilization (%)	67	156	49	135	0

FIG. 4.17 : L'utilisation des ressources pour la deuxiéme optimisation

avait une consommation de 1,765 W, répartie entre le système de processeur et la logique programmable. Dans la conception optimisée, la consommation d'énergie a légèrement augmenté à 1,815 W avec une répartition similaire. Cette augmentation peut être attribuée aux ajustements de configuration et aux fonctionnalités supplémentaires ajoutées lors de l'optimisation, ainsi qu'à l'utilisation légèrement accrue des ressources matérielles. Les techniques d'optimisation telles que le pipelining et la partition des tableaux ont permis d'améliorer les performances et l'efficacité du système, ce qui a entraîné une augmentation légère mais nécessaire de la consommation d'énergie pour maximiser les résultats obtenus.

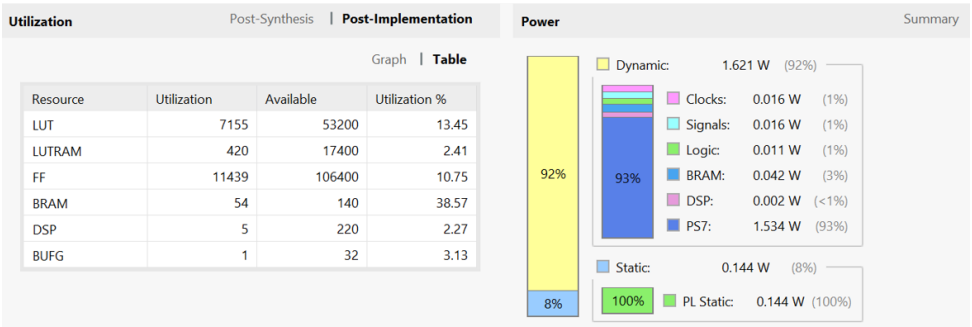


FIG. 4.18 : Utilisation des ressources non optimisée

4.5 Résultats de l'implémentation sur pynq Z1 :

Lors de l'implémentation du classificateur ECG sur la carte PYNQ-Z1, l'estimation initiale non optimisée de la latence par Vitis était de 114 ms. Cependant, la latence réelle embarquée s'est avérée être légèrement plus lente, atteignant 209,89 ms, comme indiqué dans la figure a. Cette différence peut être attribuée au fait que Vitis ne peut pas prédire avec une précision parfaite la durée de fonctionnement du

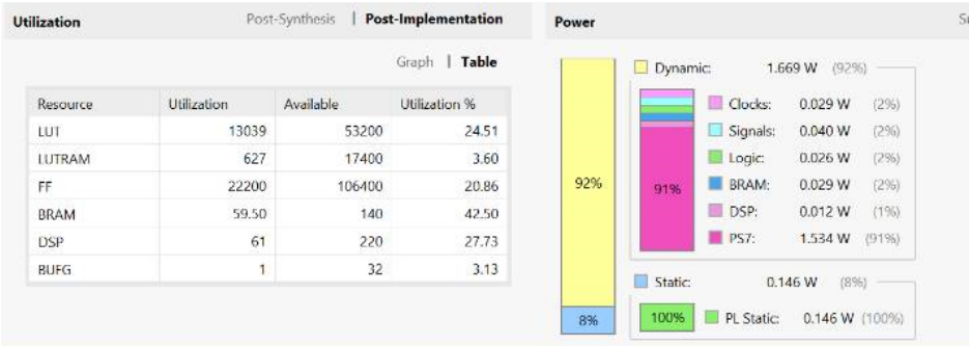


Fig. 4.19 : Utilisation des ressources optimisée

classificateur ECG embarqué, car il ne tient pas compte de divers facteurs réels et de l'entropie qui peuvent provoquer des variations dans la durée de chaque exécution embarquée.

Il est important de noter que l'estimation de la latence par Vitis est généralement inférieure à la latence réelle embarquée, que ce soit pour l'implémentation non optimisée ou optimisée. De plus , Vitis a estimé une latence optimisée de 4,28 ms, mais la latence embarquée mesurée s'est avérée être de 8,97 ms, comme illustré dans la figure 4.17.

```
In [14]: start_time = time.time()

         dut.register_map.CTRL.AP_START = 1
         dut.register_map.CTRL[4] = 1
         while not dut.register_map.CTRL.AP_DONE: pass

         end_time = time.time()
         duration = end_time - start_time
         print(f'Kernel completed in {duration * 1000:.2f}ms')

         Kernel completed in 209.89ms
```

Fig. 4.20 : La latence réelle non optimisée

```
In [54]: start_time = time.time()

         dut.register_map.CTRL.AP_START = 1
         dut.register_map.CTRL[4] = 1
         while not dut.register_map.CTRL.AP_DONE: pass

         end_time = time.time()
         duration = end_time - start_time
         print(f'Kernel completed in {duration * 1000:.2f}ms')

         Kernel completed in 8.97ms
```

Fig. 4.21 : La latence réelle optimisée

4.6 Conclusion

En conclusion, notre étude a démontré que le modèle CNN développé en Python, avec des ajustements d'hyperparamètres, surpassait d'autres méthodes de classification dans la détection des arythmies cardiaques. Les optimisations matérielles réalisées sur FPGA ont permis d'améliorer les performances et l'efficacité, malgré certains compromis en termes de dépassement des ressources. Ces résultats confirment l'efficacité de l'apprentissage profond dans ce domaine et ouvrent de nouvelles perspectives pour l'application de ces technologies dans le domaine de la santé cardiovasculaire.

Conclusion Générale

Les arythmies cardiaques sont des affections graves qui affectent un grand nombre de personnes dans le monde, et leur prévalence ne cesse d'augmenter. Dans ce mémoire, nous avons contribué à la lutte contre ces arythmies en explorant l'utilisation des réseaux de neurones artificiels et des circuits FPGA pour leur classification dans les électrocardiogrammes (ECG) portables. Nous avons adopté une approche basée sur les réseaux de neurones convolutifs (CNN) et optimisé l'utilisation des ressources limitées des FPGA afin d'améliorer la vitesse de traitement, la gestion de l'énergie et la précision de la classification des battements cardiaques.

Les résultats obtenus ont démontré l'efficacité des méthodes basées sur les réseaux de neurones, en particulier les CNN, pour la classification précise des battements cardiaques dans les ECG. En combinant les capacités d'apprentissage profond des CNN avec l'optimisation des ressources des FPGA, nous avons pu obtenir des performances remarquables. Ces résultats encourageants ouvrent de nouvelles perspectives pour la recherche future dans ce domaine.

L'utilisation de la plate-forme FPGA Pynq-Z1 a permis de réduire considérablement la latence du modèle de classification, ce qui permet une détection rapide des événements cardiaques potentiellement mortels. Cela ouvre la voie à des applications pratiques de surveillance cardiaque en temps réel, permettant ainsi une intervention médicale précoce et ciblée.

Cependant, notre travail ne s'arrête pas là. Il ouvre également de nombreuses perspectives intéressantes pour de futures recherches. Par exemple :

- L'amélioration du modèle de classification des arythmies cardiaques en explorant de nouvelles architectures de réseaux neuronaux et des techniques d'apprentissage plus avancées.
- L'intégration du classifieur ECG dans des dispositifs médicaux existants, permettant une surveillance cardiaque continue et en temps réel pour un plus grand nombre de personnes.
- L'utilisation de différentes cartes FPGA, telles que les FPGA Lattice, pour optimiser la consommation d'énergie et la taille du dispositif.

- L'intégration d'un algorithme d'intelligence artificielle explicatif (XAI) pour rendre le modèle plus transparent et faciliter son interprétation par les professionnels de la santé.

Ces perspectives offrent des opportunités d'amélioration technique, de compréhension des prédictions du modèle et d'application étendue dans le domaine de la santé. Elles contribueront à renforcer l'efficacité des dispositifs portables de surveillance cardiaque et à améliorer la prise en charge des patients atteints d'arythmies cardiaques, offrant ainsi de nouvelles opportunités pour prévenir les complications et sauver des vies.

Bibliographie

Abdullah, H. N. et B. H. Abd (2016). "A simple FPGA system for ECG R-R interval detection". In : *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, p. 1379-1382.

Acharya, U. Rajendra et et al. (oct. 2017). "A deep convolutional neural network model to classify heartbeats". In : *Computers in Biology and Medicine* 89, p. 389-396.

— (fév. 2018). "Automated identification of shockable and non-shockable life-threatening ventricular arrhythmias using convolutional neural network". In : *Future Generation Computer Systems* 79, p. 952-959.

Afkhami, Rashid Ghorbani, Ghanbar Azarnia et Mohammad Ali Tinati (2016). "Cardiac arrhythmia classification using statistical and mixture modeling features of ECG signals". In : *Pattern Recognition Letters* 70, p. 45-51.

Agrawal, A. et D. H. Gawali (2017). "FPGA-based peak detection of ECG signal using histogram approach". In : *2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*. IEEE, p. 463-468.

Alqudah, Ali Mohammad et al. (2019). "Developing of robust and high accurate ECG beat classification by combining Gaussian mixtures and wavelets features". In : *Australasian physical & engineering sciences in medicine* 42.1, p. 149-157.

Bai, Lin, Yiming Zhao et Xinming Huang (2018). "A CNN accelerator on FPGA using depthwise separable convolution". In : *IEEE Transactions on Circuits and Systems* 65.10, p. 1415-1419.

Bailey, Donald G. (2011). "Design for Embedded Image Processing on FPGAs". In : *Massey University*.

Bollengier, Heotime (2018). "Du prototypage à l'exploitation d'overlays FPGA". Français. In : *Systèmes embarqués*. NNT : 2018ENTA0003.

Bousseljot, R., D. Kreiseler et A. Schnabel (1995). "Nutzung der EKG-Signaldatenbank Cardiodat der PTB über das Internet". In : *Biomedizinische Technik/Biomedical Engineering* 40.s1, p. 317-318.

Chawla, MPS (2009). "A comparative analysis of principal component and independent component techniques for electrocardiograms". In : *Neural Computing and Applications* 18.6, p. 539-556.

Chen, A. et al. (2020). "A real time QRS detection algorithm based on ET and PD controlled threshold strategy". In : *Sensors* 20.14, p. 4003.

Chen, Yu-Hsin et al. (2017). "Eyeriss : An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In : *IEEE Journal of Solid-State Circuits* 52.1, p. 127-138.

Chollet, P. (2017). "Traitement parcimonieux de signaux biologiques". Thèse de doct. THÈSE DE DOCTORAT / IMT Atlantique sous le sceau, l'Université Bretagne en Electronique École Doctorale Mathématiques et STIC.

Ebrahimi, Zahra et al. (2020). "A review on deep learning methods for ECG arrhythmia classification". In : *Expert Systems with Applications : X* 7, p. 100033.

Elhaj, Fatin A et al. (2016). "Arrhythmia recognition and classification using combined linear and nonlinear features of ECG signals". In : *Computer methods and programs in biomedicine* 127, p. 52-63.

Goldberger, Ary Louis et al. (2018). *Goldberger's Clinical Electrocardiography : a Simplified Approach*. Ninth. Philadelphia, PA : Elsevier.

Gong, L. et al. (2018). "Maloc : A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip". In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.12, p. 2601-2612.

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Gu, Xiaoqi et al. (fév. 2016). "A Real-Time FPGA-Based Accelerator for ECG Analysis and Diagnosis Using Association-Rule Mining". In : *ACM Transactions on Embedded Computing Systems* 15, p. 1-23.

He, Jinyuan et al. (2020). "A framework for cardiac arrhythmia detection from IoT-based ECGs". In : *World Wide Web*, p. 1-16.

Izeboudjen, N. (1999). "Conception et implémentation en FPGA d'un classificateur neuronal des arythmies cardiaques". Magister thesis. Algiers, Algeria : Ecole Nationale Polytechnique.

JOHNY (2023). *la disposition du muscle cardiaque*.

Kachuee, Mohammad, Shayan Fazeli et Majid Sarrafzadeh (avr. 2018). "ECG Heartbeat Classification : A Deep Transferable Representation". In.

KASTOR, John A. (ed.) (2000). *Arrhythmias*. WB Saunders Company.

Kiranyaz, Serkan, Turker Ince et Moncef Gabbouj (mars 2016). "Real-time patient-specific ECG classification by 1-D convolutional neural networks". In : *IEEE Transactions on Biomedical Engineering* 63.3, p. 664-675.

Kirasich, Kaitlin, Trace Smith et Bivin Sadler (2018). "Random forest vs logistic regression : binary classification for heterogeneous datasets". In : *SMU Data Science Review* 1.3, p. 9.

Laaksonen, J. et E. Oja (1996). "Classification with learning k-nearest neighbors". In : *Proceedings of International Conference on Neural Networks (ICNN'96)*. T. 3, 1480-1483 vol.3.

LeCun, Y., Y. Bengio et G. Hinton (2015). "Deep learning". In : *Nature* 521.7553, p. 436-444.

Li, Hongqiang et al. (2016). "Novel ECG signal classification based on KICA nonlinear feature extraction". In : *Circuits, Systems, and Signal Processing* 35.4, p. 1187-1197.

Lu, J. et al. (2021). "Efficient hardware architecture of convolutional neural network for ECG classification in wearable healthcare device". In : *IEEE Transactions on Circuits and Systems I : Regular Papers* 68.8, p. 2976-2985.

Ma, Yufei et al. (juill. 2018). "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA". In : *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, p. 1354-1367.

Martis, Roshan Joy et al. (2013). "Cardiac decision making using higher order spectra". In : *Biomedical Signal Processing and Control* 8.2, p. 193-203.

Maurice, Bastien (2023). *Fonctionnement du neurone artificiel*.

MEDDAH, Karim (2021). "Implémentation sur FPGA d'un système intelligent pour la surveillance de l'état de santé des patients souffrants d'arythmie cardiaque". Thèse de doct. Université des Sciences et de la Technologie Houari Boumediene.

Melo, Francisco de, Horácio C. Neto et Hugo Plácido da Silva (2022). "System on Chip (SoC) for Invisible Electrocardiography (ECG) Biometrics". In : *Sensors* 22.1, p. 348.

Messaoudi, Kamel (2012). "Traitement des signaux et images en temps réel : "implantation de H.264 sur MPSoC"". Instrumentation et informatique de l'image. Thèse de doctorat. Annaba, Algérie : Université Badji Mokhtar-Annaba.

Mondéjar-Guerra, V. et al. (2019). "Heartbeat classification fusing temporal and morphological information of ECGs via ensemble of classifiers". In : *Biomedical Signal Processing and Control* 47, p. 41-48.

Moody, G.B. et R.G. Mark (2001). "The impact of the MIT-BIH Arrhythmia Database". In : *IEEE Engineering in Medicine and Biology Magazine* 20.3, p. 45-50.

natural-solutions.world (April 9, 2018). "Histoire du deep learning". In : avril 2023.

Package, The WFDB Software (2020). *wfdb - documentation wfdb 3.1.1*.

Pan, Jiapu et Willis J Tompkins (1985). "A Real-Time QRS Detection Algorithm". In : *IEEE Transactions on Biomedical Engineering* 3, p. 230-236.

Panigrahy, D., M. Rakshit et P. K. Sahu (mars 2016). "FPGA Implementation of Heart Rate Monitoring System". In : *J. Med. Syst.* 40.3, p. 1-12.

Philip, Nimmy M et N M Sivamangai (2022). "Review of FPGA-Based Accelerators of Deep Convolutional Neural Networks". In : *2022 6th International Conference on Devices, Circuits and Systems (ICDCS)*, p. 183-189.

PierreTaboulet (2023). *Les dérivation*.

Positron-Libre (2023). *Nombre en virgule fixe*.

Python, Real (consulté le mai. 06, 2023). *Plot With Pandas : Python Data Visualization for Beginners*.

Ribeiro, Antônio H et al. (2020). "Automatic diagnosis of the 12-lead ECG using a deep neural network". In : *Nature communications* 11.1, p. 1760.

Rossi, Fabrice (2023). *CNN*.

Saha, Sumit (2021). "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way". In : *Towards Data Science*.

Saptono, Debyo (2011). "Conception d'un outil de prototypage rapide sur le FPGA pour des applications de traitement d'images". Thèse de doct. Dijon.

Sze, Vivienne et al. (2017). "Efficient Processing of Deep Neural Networks : A Tutorial and Survey". In : *Proceedings of the IEEE* 105.12, p. 2295-2329.

Tedjini, Mohammed, Benabadji Noureddine et Ahmed Belbachir (juin 2014). "Conception et Réalisation D'un électrocardiographe ECG numérique." Thèse de doct.

Thomas, Manu, Manab Kr Das et Samit Ari (2015). "Automatic ECG arrhythmia classification using dual tree complex wavelet based features". In : *AEU-International Journal of Electronics and Communications* 69.4, p. 715-721.

Venkatesh, Gowri et al. (2019). "Evaluating the Processing Elements for Deep Learning in FPGAs". In : *IEEE Access* 7, p. 148455-148469.

Wang, X. et al. (2017). "An FPGA-based cloud system for massive ECG data analysis". In : *IEEE Transactions on Circuits and Systems II : Express Briefs* 64.3, p. 309-313.

Wedro, Benjamin et Absalom D. Hepner (2019). "Arrhythmias (Abnormal Heart Rhythms) : Types, Triggers, Warning Signs, and Treatment". In : *MedicineNet*.

Wei, L. et al. (2021). "A low-cost hardware architecture of convolutional neural network for ECG classification". In : *2021 9th International Symposium on Next Generation Electronics (ISNE)*. IEEE, p. 1-4.

Wess, M., P. D. Sai Manoj et A. Jantsch (2017). "Neural network based ECG anomaly detection on FPGA and trade-off analysis". In : *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, p. 1-4.

Xia, Yufa et al. (2018). "An automatic cardiac arrhythmia classification system with wearable electrocardiogram". In : *IEEE Access* 6, p. 16529-16538.

Xilinx, Inc. (2021). *Zynq-7000 SoC Technical Reference Manual*. Version X.X. Xilinx, Inc.

Y.Djeriri (2017). "Les réseaux neurones artificiels". In : *UDL-SBA*.

Yang, Yujun, Jianping Li et Yimei Yang (2015). "The research of the fast SVM classifier method". In : *2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, p. 121-124.

Ye, Can, BVK Vijaya Kumar et Miguel Tavares Coimbra (2012). "Heartbeat classification using morphological and dynamic features of ECG signals". In : *IEEE Transactions on Biomedical Engineering* 59.10, p. 2930-2941.

Zairi, H. et al. (2020). "FPGA-based system for artificial neural network arrhythmia classification". In : *Neural Computing and Applications* 32.17, p. 4105-4120.

Zhang, Zhancheng et al. (2014). "Heartbeat classification using disease-specific feature selection". In : *Computers in Biology and Medicine* 46, p. 79-89.

Webographie

Chollet, François et al. (2015). *keras*.

Dérivations de l'électrocardiogramme (s. d.). https://fr.my-ekg.com/generalites-ecg/derivations-ecg.html. 13/05/2023.

Jun, Tae Joon et al. (2018). *ECG arrhythmia classification using a 2-D convolutional neural network*.

neurone, Le (2015). *Le neurone biologique*. https://iatpe2015.wordpress.com/le-fonctionnement/le-reseau-de-neurones-artificiel/les-neurones-biologiques-et-artificiels/.

Neurone biologique (2021). https://datafranca.org/wiki/Neurone_biologique. 2023.

Processor System Reset (s. d.). https://www.xilinx.com/products/intellectual-property/proc_sys_reset.html. Consulté le jour mois année.

Pynq - Python Productivity for Zynq (s. d.). Archived on 04/02/2019. http://webcitation.org/77K5UFLoz.

Xilinx Inc. (2023a). *AXI Interconnect*. https://www.xilinx.com/products/intellectual-property/axi_interconnect.html.

— (2023b). *Zynq-7000 Processing System*. https://www.xilinx.com/products/intellectual-property/processing_system7.html.