Etude et conception de circuits innovants exploitant les caractéristiques des nouvelles technologies mémoires résistives

Vincent Lorrain

▶ **To cite this version:**

Vincent Lorrain. Etude et conception de circuits innovants exploitant les caractéristiques des nouvelles technologies mémoires résistives. Traitement du signal et de l'image [eess.SP]. Université Paris Saclay (COmUE), 2018. Français. NNT : 2018SACLS182 . tel-01827575

HAL Id: tel-01827575

https://theses.hal.science/tel-01827575v1

Submitted on 2 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université PARIS-SACLAY

NNT : 2018SACLS182

Thèse de doctorat

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

Etude et conception de circuits innovants exploitant les caractéristiques des nouvelles technologies mémoires résistives

Thèse de doctorat de l'Université Paris-Saclay
Préparée à l'Université Paris-Sud

École doctorale n°575
electrical, optical, bio : physics and engineering (EOBE)
Spécialité de doctorat : physique

Thèse présentée et soutenue à Gif-sur-Yvette, le 09/01/18, par

Vincent Lorrain

Composition du Jury :

Alain Merigot Professeur des universités, Université de Paris Sud	Président de jury
Benoit Miramond Professeur des universités, Université Côte d'Azur	Rapporteur
Sylvain Saighi Maître de Conférences, Université de Bordeaux	Rapporteur
Bernabé Linares-Barranco Professeur des universités, Instituto de Microelectronica de Sevilla	Examinateur
Tobi Delbruck Professeur des universités, UZH-ETH Zurich	Examinateur
Antoine Dupret Ingénieur chercheur, CEA	Directeur de thèse
Olivier Bichler Ingénieur chercheur, CEA	Encadrant/ Examinateur

université PARIS-SACLAY

ÉCOLE DOCTORALE
Physique et ingénierie : électrons, photons, sciences du vivant (EOBE)

Titre : Etude et conception de circuits innovants exploitant les caractéristiques des nouvelles technologies mémoires résistives

Mots clés : Réseau de neurones impulsionnels, Réseaux de neurones convolutionnels, FDSOI 28nm, Système neuromorphique, Architecture dédiée

Résumé : Dans cette thèse, nous étudions les approches calculatoires dédiées des réseaux de neurones profonds et plus particulièrement des réseaux de neurones convolutionnel (CNN).
En effet, l'efficacité des réseaux de neurones convolutionnels en font des structures calculatoires intéressantes dans de nombreuses applications.
Nous étudions les différentes possibilités d'implémentation de ce type de réseaux pour en déduire leur complexité calculatoire.

Nous montrons que la complexité calculatoire de ce type de structure peut rapidement devenir incompatible avec les ressources de l'embarqué.
Pour résoudre cette problématique, nous avons fait une exploration des différents modèles de neurones et architectures susceptibles de minimiser les ressources nécessaires à l'application.
Dans un premier temps, notre approche a consisté à explorer les possibles gains par changement de modèle de neurones.
Nous montrons que les modèles dits impulsionnels permettent en théorie de réduire la complexité calculatoire tout en offrant des propriétés dynamiques intéressantes, mais nécessitent de repenser entièrement l'architecture matérielle de calcul.
Nous avons alors proposé notre approche impulsionnelle du calcul des réseaux de neurones convolutionnels avec une architecture associée.
Nous avons mis en place une chaîne logicielle et de simulation matérielle dans le but d'explorer les différents paradigmes de calcul et implémentation matérielle et évaluer leur adéquation avec les environnements embarqués.
Cette chaîne nous permet de valider les aspects calculatoires, mais aussi d'évaluer la pertinence de nos choix architecturaux.

Notre approche théorique a été validée par notre chaîne et notre architecture a fait l'objet d'une simulation en FDSOI 28 nm.
Ainsi nous avons montré que cette approche est relativement efficace avec des propriétés intéressantes en termes de passage à l'échelle, de précision dynamique et de performance calculatoire.
Au final, l'implémentation des réseaux de neurones convolutionnels en utilisant des modèles impulsionnels semble être prometteuse pour améliorer l'efficacité des réseaux.
De plus, cela permet d'envisager des améliorations par l'ajout d'un apprentissage non supervisé type STDP, l'amélioration du codage impulsionnel ou encore l'intégration efficace de mémoire de type RRAM.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Title : Study and design of an innovative chip leveraging the characteristics of resistive memory technologies

Keywords : Spiking neural network, Neuromorphic system, CNN, FDSOI 28nm, Dedicated architecture

Abstract : In this thesis, we study the dedicated computational approaches of deep neural networks and more particularly the convolutional neural network (CNN).
We highlight that the efficiency of convolutional neural networks makes them an interesting choice for many applications.
We study the different implementation possibilities of this type of networks in order to deduce their computational complexity.

We show that the computational complexity of this type of structure can quickly become incompatible with embedded resources.
To address this issue, we explored different models of neurons and architectures that could minimize the resources required for embedded applications.
In a first step, our approach consisted in exploring the possible gains by changing the model of the neurons.
We show that the so-called spiking models theoretically reduce the computational complexity while offering interesting dynamic properties but require a complete rethinking of the hardware architecture.
We proposed our spiking approach to compute the convolutional neural networks with an associated architecture.
We have set up a software and hardware simulation chain in order to explore the different paradigms of computation and hardware implementation and evaluate their suitability with embedded environments.
This chain allows us to validate the computational aspects but also to evaluate the relevance of our architectural choices.

Our theoretical approach has been validated by our chain and our architecture has been simulated in 28 nm FDSOI.
Thus we have shown that this approach is relatively efficient with interesting scaling, dynamic precision and computational performance properties.
In the end, the implementation of convolutional neural networks using spiking models seems to be promising for improving the networks efficiency.

Moreover, can be improved by the addition of non-supervised learning (STDP), the improvement of the spike coding or the efficient integration of RRAM memory.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Table des matières

h

Table des figures

p

Liste des tableaux

r

Remerciements

Cette thèse a été effectuée au Laboratoire Calcul Embarqué (LCE) du Département Architectures Conception et Logiciels Embarqués (DACLE) du Commissariat à l'énergie atomique et aux énergies alternatives (CEA) (CEA LIST), sur le plateau de Saclay. Je remercie, Thierry Collette, ancien chef du DACLE, pour son accueil et les moyens qu'il m'a fourni pour accomplir cette thèse dans les meilleures conditions. Je tiens à remercier Nicolas Ventroux, chef du LCE et son prédécesseur Raphaël David pour la confiance et le soutien qu'ils m'ont accordé tout au long de ces trois ans.

Je remercie Antoine Dupret, ingénieur chercheur au CEA, pour avoir accepté de diriger cette thèse et pour avoir participé à mon jury. Je tiens à le remercier pour son enthousiasme et son soutien sans faille tout au long de ma thèse. Je remercie, Olivier Bichler, ingénieur chercheur au CEA, pour avoir effectué l'encadrement de ma thèse et pour avoir participé à mon jury. Je tiens à le remercier pour la confiance qu'il m'a accordé et pour la grande autonomie qu'il m'a laissé, ainsi que pour son expertise de grande valeur. Nos échanges m'ont permis de prendre du recul sur mes travaux et de conduire efficacement mes travaux de thèse.

Je remercie Benoit Miramond, de l'Université Côte d'Azur, et Sylvain Saighi, de l'Université de Bordeaux, pour avoir accepté de juger mon travail en tant que rapporteur ainsi que pour leur participation à mon jury de thèse. Je remercie également Bernabé Linares-Barranco , professeur à l'Instituto de Microelectrónica de Sevilla et Tobi Delbruck , professeur à Institute for Neuroinformatics UZH-ETH, pour leur participation à mon jury. Je remercie Alain Merigo, professeur à l'Université de Paris Sud, pour avoir accepté de présider mon jury de thèse.

Je tiens à remercier l'ensemble des membres du LCE, L3A, L3S et du LFIC que j'ai pu côtoyer pendant ma thèse. La bonne ambiance de travail qui règne au sein de ces équipes a sans aucun doute contribué à la réussite de cette expérience. J'ai une pensée particulière pour les collègues/amis ayant partagé mon bureau et particulièrement Julien Collet, David Briand, Joël Cathebras.

Je remercie ma femme, Charlotte Lorrain et mes parents Sylvie Lorrain et Xavier Lorrain pour m'avoir encouragé, supporté et pour l'intérêt qu'ils ont porté à mes travaux. Je remercie enfin l'ensemble de ma famille et de mes amis pour leur soutien.

t

Chapitre 1

Introduction générale

On peut définir un système embarqué comme un composant logiciel et matériel dédié à une tache spécifique dans un contexte contraint. Les systèmes embarqués sont soumis à plusieurs contraintes, en matière de réactivité, de consommation/autonomie, mais aussi de coût. De plus les systèmes embarqués sont assignés à une tâche spécifique, ils prennent place dans un ensemble plus grand pour fournir un service. Cet ensemble peut prendre plusieurs formes en fonction de l'application traitée avec des réseaux du système embarqué autonome ou centralisé, etc.

Depuis quelques années les thématiques industrielles et les recherches sur les systèmes embarqués sont en expansion. Cette expansion s'explique par l'évolution et l'augmentation des services proposés par nos objets, ainsi on remarque une croissance des thématiques de l'IOT, de la voiture autonome, smart devise, du smartphone, etc.

L'expansion des solutions embarquées pose de nouveaux défis en matière de traitement des données et de communication. Les modèles classiques de traitement centralisé par un serveur, avec plusieurs systèmes embarqués faisant office plus ou moins de capteurs vont rapidement être limités avec l'augmentation du nombre de systèmes embarqués et de la bande passante disponible. Ainsi, pour limiter cette problématique de la communication, une solution possible est d'embarquer au moins une partie du traitement pour minimiser les communications vers une centrale de traitement, ainsi les données seront acquises et traitées par le système embarqué et potentiellement seuls les résultats seront transmis vers une unité centralisée. Ce traitement local a pour conséquence l'augmentation de la complexification des tâches à effectuer par les systèmes embarqués tout en maintenant les mêmes contraintes.

Si l'on s'intéresse aux tâches classiquement effectuées actuellement côté serveur on peut identifier la tache de classification. Cette tâche est utilisée dans de nombreuses problématiques telles que la reconnaissance de motif, le partitionnement de scènes, etc. L'approche actuelle est de fournir au serveur les données à classifier puis il fournit en retour les classes, ce qui peut représenter une grande quantité de données. Parallèlement, les applications de la classification dans les systèmes embarqués sont nombreuses, de la caméra de surveillance intelligente au partitionnement

de scène pour la voiture autonome à la reconnaissance de défaut, etc. Il y a donc un intérêt croissant à embarquer des applications de classification.

Il existe des structures de calcul performant et régulier pour la classification de données. En s'intéressant aux algorithmes aujourd'hui utilisés pour les tâches de classifications, on remarque qu'une classe de solution est devenue prédominante : les réseaux de neurones profonds (Deep Neural Network, DNN). Cette structure comme son nom l'indique s'inspire du fonctionnement du cerveau par l'utilisation de blocs de base, les neurones connectés entre eux pour former un réseau. L'idée du calcul à base de neurones artificiels n'est pas récente et a évolué des années 30 aux années 70 avec divers travaux de modélisation qui ont permis de formaliser plusieurs modèles théoriques du fonctionnement des neurones. Et les années 80-90 ont vu apparaître des structures de calculs capables de simuler un réseau de neurones, mais surtout les algorithmes d'apprentissage permettant d'entrainer un réseau en fonction d'un jeu de données.

Ces années ont vu l'apparition des réseaux de neurones convolutionnels (Convolutional neural network, CNN). Ces structures de réseaux de neurones et les algorithmes d'apprentissage associés ont modifié l'approche du traitement et de classification de données. Ainsi, la fonction de traitement permettant de déterminer des propriétés d'une donnée d'entrée par application de fonctions directement programmées s'est vue remplacée par une phase d'apprentissage dans laquelle le réseau est entrainé au traitement des données. La performance du réseau est à la fin liée à son taux de réussite à fournir en sortie les bonnes propriétés des jeux d'entrée. Cette approche permet donc de s'abstraire des propriétés des données d'entrée, car l'utilisateur n'a pas à fournir explicitement leur l'analyse. Ces structures sont rapidement devenues la référence pour la résolution des problématiques de classification, et sont aujourd'hui au cœur du fonctionnement de plusieurs services tels que Google images, la reconnaissance de personne sur Facebook, etc. Les CNN sont devenues des structures classiques de la classification, il est donc intéressant d'étudier leur embarquabilité.

Au vu de leur propriété et des résultats obtenus, ces structures CNN sont rapidement devenues des solutions populaires. Les réseaux utilisés ont naturellement et progressivement évolué en matière de nombre de neurones implémentés et de problématique adressée. Cette augmentation de taille à permis à ces types de structure de calcul d'adresser des problèmes de plus en plus complexes, voir le figure 1.1. Mais cette augmentation pose la question de l'instanciation efficace de ces structures au sein d'architectures embarquées.

Pour statuer sur les possibilités d'implémentation des CNN dans le domaine de l'embarquer, une étude est nécessaire. Il faut déterminer les ressources calculatoires nécessaires pour les différentes possibilités d'instanciation de ces réseaux. Dans ce manuscrit nous allons présenter notre méthode nos choix d'implémentation et nos

résultats sur le calcul des CNN.

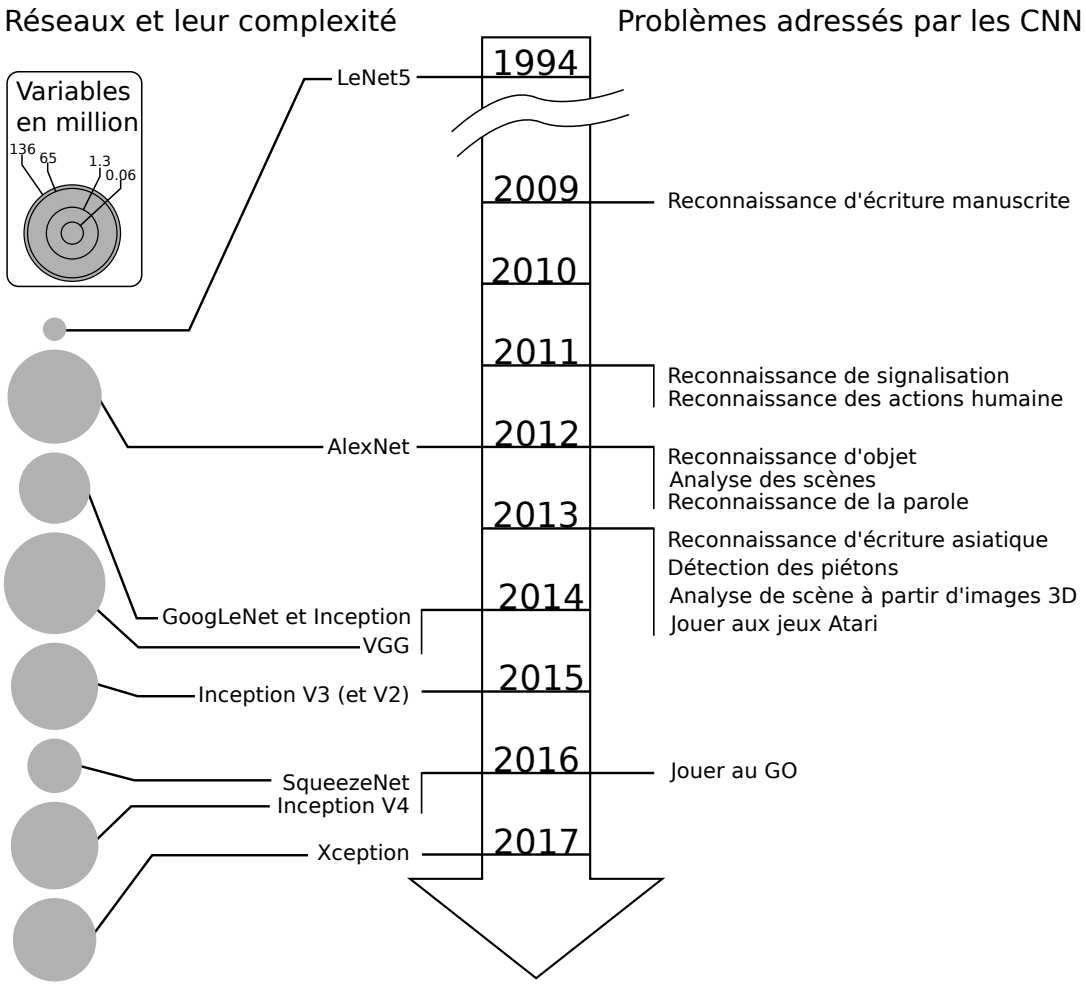

FIG. 1.1: Évolution des CNN et de leur complexité ainsi que l'évolution des problématiques adressées par des solutions contenant des CNN.

Organisation du mémoire Le mémoire est organisé de la manière suivante :

- Le premier chapitre débute avec une introduction aux réseaux de neurones convolutionels, leurs origines et leurs topologies. Puis nous expliquerons les différents modèles de neurones utilisables en explicitant leurs relations de transposition Le chapitre se conclura par une étude d'influence théorique des deux modèles sur la complexité calculatoire.
- Le deuxième chapitre est une étude des différentes solutions matérielles du calcul des CNN en fonction du modèle de neurone. Dans ce chapitre, nous verrons les différentes stratégies de calcul et déterminerons l'approche qui nous semble la plus prometteuse pour le calcul des CNN.
- Le troisième chapitre est la présentation de notre chaîne de mesure et la validation de notre architecture ainsi qu'une étude d'optimisation calculatoire

- Le quatrième chapitre est la présentation de notre architecture. Nous expliquerons les choix dans la distribution du calcul et les conséquences matérielles qui en découlent. Ce chapitre se conclura par une étude différentielle des performances de notre solution et de l'état de l'art.
- Le chapitre de conclusion nous permettra de discuter sur notre approche et de présenter un certain nombre de réflexions personnelles sur le calcul embarqué des CNN.

Chapitre 2

L'inférence

Contents

2.1 Introduction

Il existe deux types de modèles de neurone, les modèles formels, que nous aborderons dans la section 2.3.1 et les modèles impulsionnels, que nous aborderons dans la section 2.4.1. Notre objectif est d'étudier la potentielle réduction de la complexité calculatoire par changement du modèle de neurone formel pour un modèle de neurone impulsionnel pour les CNN (convolutional neural network) dans les systèmes embarqués. Nous ne considérons que le calcul de l'inférence du réseau, c'est-à-dire que l'utilisation de ce réseau pour classifier des données est que l'étape d'apprentissage a été effectuée, voir la figure 2.1.

Afin de comparer le plus objectivement possible ces deux modèles, il est nécessaire de mettre en place des métriques de comparaison. Dans l'état de l'art, on ne retrouve pas une grande variété de travaux de comparaisons entre les modèles formels et les modèles impulsionnels, et ceux existant [7, 8], ne sont pas assez ciblés pour notre problématique de l'inférence des CNN dans les systèmes embarqués. Le premier ne considère pas les opérations effectuées dans les différents modèles, le deuxième est plus une étude de l'équivalence entre les deux modèles sans prise en compte des architectures.

Dans ce chapitre, je propose donc mon expression des métriques de comparaison qui se base sur la complexité calculatoire des CNN selon les deux modèles de neurones. Nous utiliserons une définition de la complexité calculatoire en nombre d'opérations atomiques notées OP qui correspond à une opération d'addition (full-adder, FA) ou de comparaison numérique sur 1 bit.

Pour faire cette comparaison, je vais développer mon analyse progressivement :

Premièrement, je détaillerai la structure des CNN pour poser un certain nombre de notations. Deuxièmement, j'exprimerai la complexité des CNN formels en nombre d'opérations atomiques OP. Troisièmement, j'exprimerai la complexité des CNN impulsionnels en nombre d'opérations atomiques OP. Quatrièmement, j'effectuerai une comparaison de la complexité calculatoire théorique entre le modèle formel et le modèle impulsionnel. Finalement, j'utiliserai toutes ces notions pour définir notre logique de comparaison entre les architectures formelles et les architectures impulsionnelles.

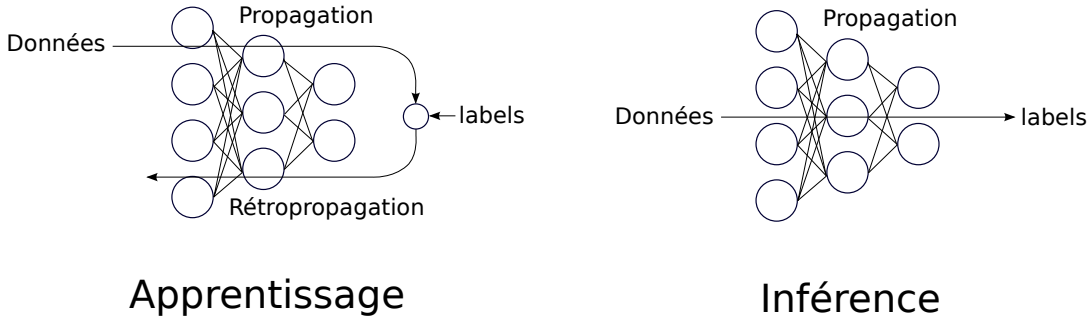

FIG. 2.1: Les deux modes d'utilisation d'un réseau en inférence ou en apprentissage. En apprentissage les données sont propagées à travers le réseau, le résultat est comparé au label et une erreur est rétropropagée dans les réseaux afin de modifier les paramètres. En inférence les paramètres sont fixés et le réseau ne fait que propager les données, le réseau est utilisé pour effectuer la tâche apprise.

2.2 Les réseaux de neurones convolutionnels

Nous allons présenter dans cette section les différents types de couches classiquement utilisés dans les CNN. Le but est d'exprimer leur connectivité indépendamment du modèle de neurone. Cette section nous permettra aussi de poser des notations que nous réutiliserons tout au long du mémoire. Nous verrons dans un premier temps les origines de cette topologie en parcourant les différents travaux sur le cortex visuel. Nous introduirons ainsi le principe de champ récepteur, de partage des pondérations, et la structure en couches.

Par la suite, nous introduirons la structure moderne des CNN ainsi que quelques-unes des couches les plus utilisées. Nous expliciterons chacune de ces couches pour en déduire leurs différents paramètres et fonctionnements. Nous serons à même d'exprimer la complexité calculatoire de chaque type de couche en fonction du modèle de neurone utilisé. Ce qui nous permettra finalement de dimensionner les besoins calculatoires théoriques de ce type de réseau en fonction du modèle, et la meilleure approche théorique du calcul.

2.2.1 Origine du principe des réseaux de neurones convolutionnel

L'origine des CNN est liée aux travaux sur les mécanismes biologiques de la vision. Ainsi on peut remonter dans les années 1950 - 1960 avec les travaux de caractérisation du cortex visuel de Hubel et Wiesel [9] qui ont identifié deux types de cellules dans le cortex visuel, les cellules simples et les cellules complexes. Les couches de cellules simples sont utilisées pour l'extraction de motifs et les couches de cellules complexes réunissent les motifs reconnus, rendant l'application moins sensible aux variations de position. Ils ont ainsi introduit les concepts de champs

réceptifs 2D des neurones et de pooling. Dans les années 1980, la notion de Neocognitron inspirée par les travaux de Hubel et Wiesel, fut introduite par Kunihiko Fukushima [10]. La structure Neocognitron est une structure en couches, où celles-ci sont interconnectées en cascade. Ces couches sont composées de cellules simples ou complexes et chacune possède son propre jeu de pondérations. Les couches de cellules simples sont utilisées pour l'extraction de motifs et les couches de cellules complexes réunissent les motifs reconnus, rendant l'application moins sensible aux variations de position. Finalement, cette structure de Neocognitron fut utilisée dans une application de reconnaissance de chiffres.

En 1986, la structure du Neocognitron a été réutilisée avec l'idée de partage des mêmes pondérations entre plusieurs neurones d'une même couche associée aux principes de base de l'apprentissage formel par optimisation du gradient, la rétropropagation du gradient (backpropagation, BP) [11]. Ainsi, les bases structurelles des réseaux convolutionels ont été posées avec une première formalisation d'un algorithme d'apprentissage. Par la suite, en 1998, Yann LeCun *et al* ont proposé une structure de réseaux de convolutions, LeNet-5 figure 2.2, associé à un algorithme d'apprentissage pour la classification de chiffres [12].

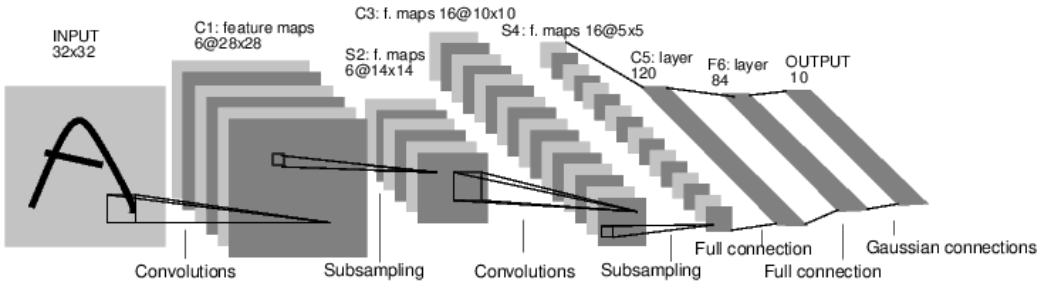

FIG. 2.2: Réseaux convolutionnels LeNet-5.

Au final, les CNN sont composés de couches non rebouclées, chaque couche est composée de neurones qui sont connectés à la couche précédente par leur champ récepteur. Il existe aussi plusieurs types de couches. Et dans ces couches un certain nombre de pondérations sont communes à plusieurs neurones.

Au vu de cette structure globale, nous allons par la suite définir la connectivité d'un CNN par la définition de la connectivité de chaque couche. Nous allons donc maintenant nous intéresser aux détails des connexions dans les couches.

2.2.2 Structure des réseaux de neurones convolutionnels

On peut modéliser une couche d'un CNN comme un tenseur 3D composé de plusieurs matrices 2D ou chaque élément de la matrice correspond à la sortie d'un neurone. Pour toutes les couches, il existe un tenseur d'entrée (qui correspond à la

couche précédente) et un tenseur de sortie. Les neurones d'une même matrice de sortie, partage les mêmes pondérations notées filtres sont connectés aux matrices du tenseur d'entrée.

Ainsi, on définit le tenseur de sortie comme O_{elmZ} matrice 2D de largeur O_{elmX} et de hauteur O_{elmY}. Par exemple pour le réseau LeNet-5 de la figure 2.2, la couche C1 est définie par $O_{elmX} = 28, O_{elmY} = 28, O_{elmZ} = 6$. Pour le tenseur 3D d'entrée, qui par exemple pour la couche C1 est le tenseur 3D d'entrée INPUT. Ce tenseur 3D d'entrée est défini par sa largeur I_{elmX}, sa hauteur I_{elmY} et sa profondeur I_{elmZ}.

Les connexions entre tenseurs 3D d'entrée et le tenseur de sortie sont effectuées par des filtres qui sont définis par leurs largeur C_{elmX} et hauteur C_{elmY}. Cette taille de filtre est identique pour toutes les connexions entre deux tenseurs. Le nombre de filtres dépend de la connectivité entre tenseurs 3D d'entrée et le tenseur 3D de sortie. On note ce nombre de filtres C_{elmZ}, si les deux couches sont entièrement connectées il y a $I_{elmZ} \times O_{elmZ}$ filtres. Dans le cas d'une connexion partielle, on définit une matrice d'interconnexion. Voir figure 2.3 pour une représentation de cette connectivité entre deux couches.

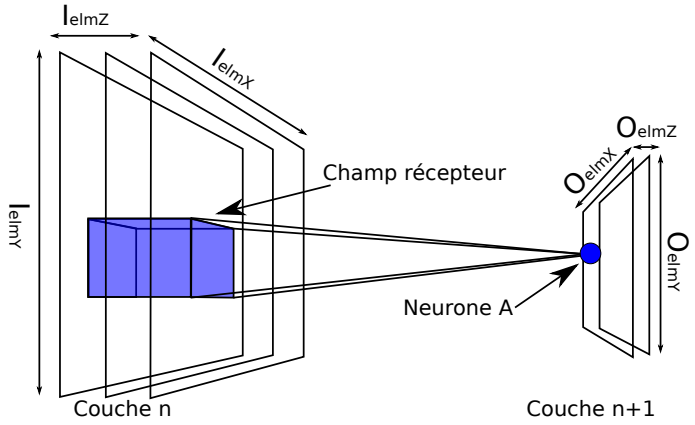

FIG. 2.3: Représentation d'un champ récepteur 2D du neurone A de la couche n+1 connecté aux 3 images de $I_{elmX} \times I_{elmY}$ pixels de la couche n. La sortie des neurones de la couche n+1 forme 2 images de $O_{elmX} \times O_{elmY}$ pixels.

Dans une matrice 2D de sortie, les pondérations entre deux neurones sont identiques, mais la position de leurs filtres sur la matrice d'entrée est différente. Le champ récepteur des neurones d'une couche est défini par : la distance minimale entre deux champs récepteurs sur la couche précédente exprimée en un nombre d'éléments qu'on note "pas" (anglais : stride) St_x, St_y, et l'ajout de zéros sur les bordures P_x, P_y (zéro padding) correspondant à un nombre de lignes et colonnes ajoutées en bordure des matrices originales. Pour résumer, la figure 2.4 représente la connectivité de deux neurones, la taille de la matrice de sortie est alors donnée par l'équation (2.1).

$$\begin{cases} O_{elmX} & = \lfloor \frac{I_{elmX} - C_{elmX} + 2 \cdot P_x + St_x}{St_x} \rfloor \\ O_{elmY} & = \lfloor \frac{I_{elmY} - C_{elmY} + 2 \cdot P_y + St_y}{St_y} \rfloor \end{cases} \tag{2.1}$$

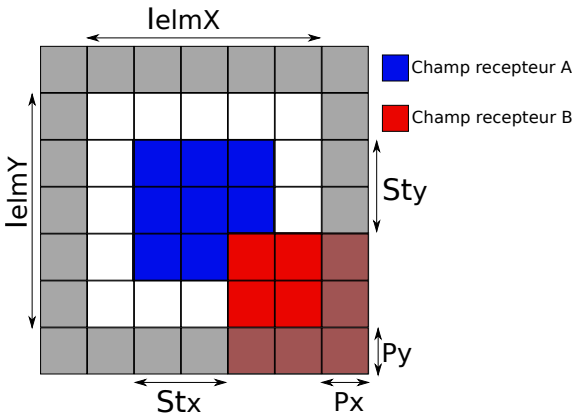

FIG. 2.4: Représentation de deux champs récepteurs de 3×3 éléments connectés à une même matrice d'entrée de 6×6 pixels avec $St_x = St_y = 2$ et un zéro sur les bordures de $P_x = P_y = 1$.

Enfin, pour définir la connexion entre deux couches nous utiliserons par la suite les notations introduites dans cette section. Nous avons donc tous les outils pour quantifier les connexions entre deux couches. Cependant, il existe plusieurs types de couches, toutes ces couches ne sont pas identiques en matière de traitement, mais aussi en matière de connectivité. Nous allons donc nous intéresser maintenant aux différents types de couches classiquement utilisés dans les réseaux CNN.

2.2.3 Les types de couches dans les CNN

Mon étude des types de couches ne se veut pas exhaustive et nous nous concentrerons sur les couches les plus utilisées dans les CNN. Nous allons nous focaliser sur les couches les plus classiques : la couche de convolution (Conv), la couche entièrement connectée (FC) et la couche d'union (pooling). On considère un Stride de 1 et Zeropading de 0, pour être dans le cas le plus connecté. On considère pour simplifier les expressions qu'il n'y a qu'une seule matrice dans le tenseur d'entrée et dans le tenseur de sortie pour les couches Conv et pooling. Pour la couche de FC on considère une seule matrice dans le tenseur d'entrée et que les neurones de sorties de cette couche correspondent à la profondeur du tenseur de sortie, il y en a donc O_{elmZ}.

Pour chacun de ces types de couches nous allons détailler sa fonction générale (indépendamment du modèle du neurone) et définir : Premièrement, le nombre total de connexions dans la couche $NbC_{total}(couche)$. Deuxièmement, le nombre de neurones connectés à la même entrée $NbC_{neuro}(couche)$. Ces deux valeurs seront

utilisées pour exprimer la complexité des couches en fonction du modèle de neurone utilisé.

La couche de convolution (Conv) est la brique de base des CNN. Cette couche est utilisée pour la détection de motif avec une invariance en translation. Pour cela l'opération de convolution utilise un jeu de filtres, composé de pondérations. Les pondérations composant les différents filtres sont déterminées lors de la phase d'apprentissage. On peut alors exprimer les connectivités d'une couche de convolution :

$$NbC_{total}(Conv) = O_{elmX} \times O_{elmY} \times C_{elmX} \times C_{elmY}$$

$$NbC_{neuro}(Conv) \approx C_{elmX} \times C_{elmY} \text{ au centre de la matrice d'entrée}$$

La couche entièrement connectée (FC) Cette couche est utilisée pour associer les différents motifs afin d'en déduire la classe. Comme son nom l'indique, la matrice de connexion est entièrement connectée. C'est-à-dire, qu'elle prend en entrée un tenseur 3D (potentiellement une couche de convolution) ou un vecteur et en ressort un vecteur. Le nombre de neurones de sortie de cette couche est alors égal à O_{elmZ}. On utilise classiquement ce type de couche plusieurs fois en fin de réseau pour résoudre le problème du XOR. La connectivité de ce type de couche peut être la même que celle de la couche de convolution, mais avec un champ récepteur égal aux dimensions de la couche précédente. Cette couche est elle aussi soumise à l'apprentissage. On peut exprimer le nombre de connexions totales et le nombre de neurones connectés à la même entrée avec :

$$NbC_{total}(FC) = I_{elmX} \times I_{elmY} \times O_{elmZ}$$

$$NbC_{neuro}(FC) = O_{elmZ}$$

La couche d'union (pooling) fait partie des couches standards des CNN. Le principe de ce type de couche est de réduire la détection de patterns ambigus. C'est pour cela que celles-ci sont classiquement utilisées à la suite d'une couche de convolution comme dans les réseaux VGG [13], AlexNet [14]. Ce type de couche ne dépend pas d'un jeu de pondération, elles ne sont pas modifiées par l'apprentissage. Il existe plusieurs types de couches d'union : par maximum $y = max(entrées)$, par moyenne, $y = moy(entrées)$ etc. Nous nous focaliserons sur l'étude de la couche d'union par maximum, car elle est dans de nombreux cas plus performante que la couche d'union par moyenne [15]. Le calcul du nombre de connexions et le nombre de neurones connectés à la même entrée de la couche d'union par maximum sont identiques à celui de la couche de convolution :

$$NbC_{total}(MaxPooling) = O_{elmX} \times O_{elmY} \times C_{elmX} \times C_{elmY}$$

$NbC_{neuro}(MaxPooling) \approx C_{elmX} \times C_{elmY}$au centre de la matrice d'entrée

Par la suite et pour des raisons de lisibilité nous utiliserons les notations I, C et O qui sont respectivement définie par $I = I_{elmX} \times I_{elmY}$, $C = C_{elmX} \times C_{elmY}$ et $O = O_{elmX} \times O_{elmY}$. Nous avons posé la définition de toutes les notations que nous allons utiliser pour décrire un réseau CNN ainsi que les principaux types de couches. Nous allons nous intéresser au fonctionnement des différents modèles pour exprimer la complexité des CNN en fonction du modèle utilisé.

2.3 Le CNN formel, le modèle et application

Dans cette partie, nous allons nous intéresser au modèle dit formel. Nous commencerons par le détail des opérations effectuées dans un neurone formel et l'explication des raisons de son utilisation courante. Puis nous étudions l'état de l'art des techniques d'optimisation et de discrétisation des réseaux formels pour quantifier les tailles des variables en inférence numérique. Pour finir, nous exprimerons la complexité calculatoire d'un neurone formel en nombre d'opérations atomiques pour chaque type de couche d'un CNN.

2.3.1 Le modèle formel

Le modèle de neurone formel a été proposé en 1943 par McCulloch *et al* [16]. Ce premier formalisme considère un neurone à sortie binaire déterminé par une fonction de seuil dépendante de la somme pondérée des entrées. Ce modèle formel se compose d'une partie pondération W, d'une somme et d'une fonction seuil $h()$ définie par l'équation 2.2. L'expression de la fonction du neurone formel de McCulloch *et al* est présentée dans l'équation 2.3, avec $x_0 = 1$ et le seuil donné par W_0. Une représentation graphique du neurone formel de McCulloch *et al* est présenté figure 2.5.

$$h(x) = \begin{cases} 1, & \text{si } x \geq 1 \\ 0, & \text{sinon} \end{cases} \tag{2.2}$$

$$y = h(\sum_{i=0}^{N}(w_i \times x_i)), x_0 = 1 \tag{2.3}$$

La popularisation du modèle formel est venue par la suite, dans les années 70-80 par l'introduction d'une méthode d'apprentissage, la rétropropagation du gradient (Backpropagation, BP). Cette méthode est une adaptation de la descente en gradient pour les réseaux de neurones multicouches qui permet d'entrainer un réseau pour une tache. Elle a été introduite par les travaux de Werbos en 1971 publiée en 1974

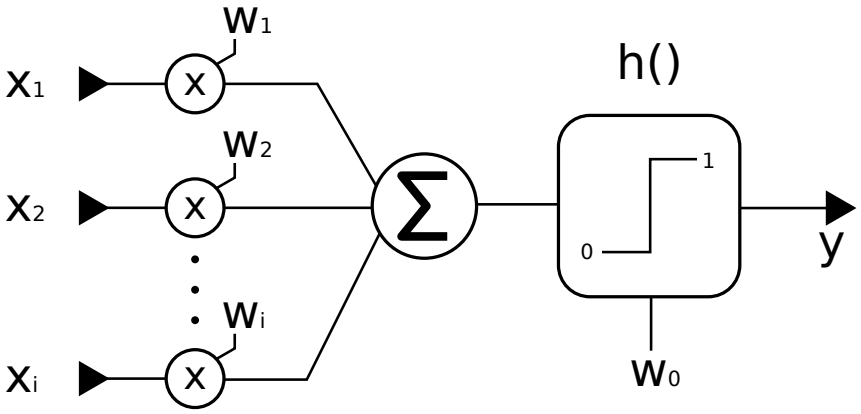

FIG. 2.5: Représentation générale d'un neurone formel proposé par McCulloch *et al* les entrées notées X_i sont binaires et pondérés par un facteur W_i, le tout est sommé et envoyé à une fonction d'activation binaire $h()$ dons le seuil est W_0, ce calcul nous donne une valeur binaire de sortie du neurone y.

[17] reprise et discutée par leCun [18]. Nous ne rentrerons pas dans le détail de cet algorithme et de ses diverses approches et optimisations.

Au niveau du neurone la seule modification à opérer sur le modèle de McCulloch *et al* [16] pour lui appliquer la rétropropagation du gradient est le changement de la fonction seuil $h()$ par une fonction d'activation dérivable $H()$. Les possibilités de fonction d'activation sont variées : fonction sigmoïde, fonction tangente hyperbolique, fonction identité, ReLU (Rectifed linear units) etc. C'est l'association du modèle formel à l'algorithme de rétropropagation du gradient qui a popularisé cette approche du calcul à base de réseaux de neurones formels.

Le problème de ce changement de fonction d'activation est de connaitre la précision des variables à utiliser pour son instrumentation numérique efficace. Nous allons donc maintenant nous intéresser au dimensionnement de la complexité des neurones formels en inférence. Le but est de déterminer quel codage et quel précision nous devons utiliser pour le calcul de l'inférence formelle numérique.

2.3.2 Discrétisation, optimisation calculatoire et mémoire

L'étude de la discrétisation des variables a été poussée par les différentes implémentations matérielles. Le but de cette discrétisation est de réduire les besoins matériels en simplifiant la représentation des variables et/ou en réduisant leur dynamique. Pour opérer cette réduction, plusieurs propositions ont été faites dans le sens de la réduction de la dynamique des entrées c.-à-d. la réduction des pondérations w et des valeurs d'entrée y. Ainsi on retrouve les travaux de J. L. Holt et T. E. Baker [19] qui montrent que le passage d'un codage à virgule flottante à un codage entier n'a pas dans la majorité des cas un fort impact sur l'inférence. Et les travaux de E.fiesler et R. Beale [20] qui font une étude des différentes techniques de réduction

des coûts d'implémentation matérielle. Ces travaux ont montré la possibilité du passage d'un codage virgule flottant à un codage entier en utilisant une discrétisation des pondérations post-apprentissage sur 8 bits. Ces études montrent que le codage entier et la réduction de la dynamique des pondérations ont un impact acceptable dans le cadre de l'inférence. Les travaux de Fiesler *et al* [21] ont montré quant à eux qu'il était possible de réduire encore la dynamique des pondérations. Pour ce faire, ils ont inclus la discrétisation dans le domaine de l'apprentissage en utilisant un réapprentissage post-discrétisation. Ces techniques de discrétisation et d'optimisation de la dynamique des pondérations sont aujourd'hui bien maîtrisées et permettent de simplifier l'implémentation sans grande perte par rapport à une implémentation en virgule flottante.

À la suite de ces études sur la discrétisation un deuxième axe de recherche sur le codage des données pour la réduction des besoins mémoires est apparu, l'idée générale étant de réduire les besoins mémoires d'une instanciation matérielle d'un réseau sans grande modification de ses performances en changeant le codage de l'information et/ou en réduisant son nombre de pondérations. Dans les travaux de Song Han *et al* [22] est proposée une chaîne de traitement avec un codage de Huffman permettant de réduire de 35× à 49× les besoins mémoire. Ou encore le passage à zéro des pondérations de faible valeur pour réduire le nombre d'opérations et les besoins mémoires [23].

Pour conclure, ces différentes études montrent que dans le cas de l'inférence il est possible d'envisager une instanciation des neurones numériques avec des calculateurs entiers sur 8 bits sans une grande perte de performance. Nous considérerons donc par la suite une instanciation numérique des neurones formels avec une dynamique maximale de 8 bits entiers.

2.3.3 Complexité des couches des CNN formels

Au vu de la section 2.3.2, nous considérerons une instanciation 8 bits. Le calcul dans un neurone formel est un produit scalaire entre le vecteur d'entrée et les vecteurs de pondération sur lequel on applique une fonction d'activation, pour obtenir la réponse du neurone. On peut considérer que le neurone effectue principalement des opérations de multiplications et d'accumulations (MAC) et le résultat de ces MAC est l'entrée de la fonction d'activation $H(x)$. La complexité calculatoire en nombre d'opérations atomiques OP du calcul de l'inférence d'un neurone formel peut alors être déduite du nombre de MAC à effectuer et de la complexité d'une MAC notée C_f.

Pour la complexité d'une MAC, on peut approximer la complexité de la multiplication des nombres entiers en un nombre d'additions équivalentes autrement dit d'opérations atomiques. Ainsi, si l'on considère une implémentation de la multiplication d'entiers combinatoire en grille (array multiplicator) avec une précision de

pondération en nombre de bits de Nbf_{bit}. Dans cette implémentation en grille pour un Nbf_{bit} donnée il y à $Nbf_{bit} \times (Nbf_{bit} - 2)$ full-adder (FA) et Nbf_{bit} half-adder (HA). Je considère que la complexité en opération atomique d'un half-adder est la moitié de celle d'un full-adder. Alors, le nombre d'additions 1 bit équivalent est approximativement égal à $Nbf_{bit} \times (Nbf_{bit} - 2) + \frac{Nbf_{bit}}{2}$. Si on lui ajoute la complexité d'une addition de $2 \times Nbf_{bit}$, avec $2 \times Nbf_{bit} - 1$ full-adder et 1 half-adder, le nombre d'opérations atomiques d'une MAC est égal à :

$$OP_{MAC}(Nbf_{bit}) = \frac{1}{2}(2 \times Nbf_{bit} - 1)(Nbf_{bit} + 1)$$

Si on considère, un réseau utilisant des pondérations sur 8 bits, il est possible de l'implémenter dans une architecture 32 bits. Cependant, cette architecture n'utilisera pas toute sa dynamique de calcul. Ainsi en évaluant le nombre d'opérations atomique qu'elle effectue il ne faut pas considérer toute la dynamique 32 bits Finalement, la complexité de l'opération MAC en termes d'opérations atomiques est donnée par l'équation 2.4.

$$\begin{cases} C_f = OP_{MAC}(Nbf_{bit}) \text{ Si } Nbf_{bit} \leq 8 \\ C_f = OP_{MAC}(8) \text{ Sinon} \end{cases} \tag{2.4}$$

Ainsi, la complexité selon notre expression en opération atomiques de l'opération MAC avec des entrées sur 16 bits est égale à la complexité de l'opération MAC avec des entrées sur 8 bits soit 67,5 OP.

Nous pouvons alors approximer la complexité d'une couche de neurones formels en fonction du nombre de connexions de cette couche $NbC_{total}(couche)$. Nous obtenons alors pour une couche formelle l'équation 2.5 qui exprime la complexité calculatoire de la couche en nombre d'opérations atomiques.

$$OP_{couche_{formel}} = C_f \times NbC_{total}(couche) \tag{2.5}$$

Même si en théorie le modèle formel ne nécessite pas de sauvegarde de la valeur d'un contexte lors de l'inférence, autrement dit son coût mémoire est nul. En pratique le calcul des MAC sur les entrées n'est pas totalement parallèle, il y a donc une sauvegarde des états intermédiaires du calcul. Cette quantité mémoire est fortement liée à l'architecture et à sa logique de calcul, ainsi il est complexe de l'évaluer. Nous considérerons donc pour nos calculs une instanciation matérielle sans coût mémoire dû à la sauvegarde des valeurs intermédiaire.

Au vu de la section 2.2.3, on peut exprimer $NbC_{total}(Conv)$ et $NbC_{total}(FC)$ ainsi la complexité en nombre d'opérations atomiques d'une couche de convolution formelle est donnée par l'équation 2.6. Et la complexité en nombre d'opérations atomiques d'une couche entièrement connectée (FC) formelle est donnée par l'équation 2.7.

$$OP_{Conv_{formel}} = O \times C \times C_f \tag{2.6}$$

$$OP_{FC_{formel}} = O_{elmZ} \times I \times C_f \tag{2.7}$$

La fonction MaxPooling formelle quant à elle ne peut pas être déduite des équations précédentes, car elle ne s'exprime pas en nombre de MAC. La sortie d'un neurone de MaxPooling formel est égale à la valeur maximale dans son champ réceptif. Il faut comparer donc toutes les valeurs d'entrées du neurone pour déterminer la valeur maximale qui sera propagée en sortie, voir figure 2.6.

Il est possible d'exprimer une complexité calculatoire en fonction du nombre de comparateurs nécessaires pour identifier la valeur maximale. En considérant qu'une opération de comparaison de deux éléments de 1 bit est une opération atomique voir 2.3.1, la complexité calculatoire pour un neurone MaxPooling formel est alors égale à $C_m = C \times log_2(C)$, avec C le nombre de comparaisons à effectuer (nombre d'entrées) par le neurone et $\lceil log2(C) \rceil$ la complexité de ces comparaisons. Ainsi la complexité en nombre d'opérations atomiques d'une couche MaxPooling formelle est donnée par l'équation 2.8.

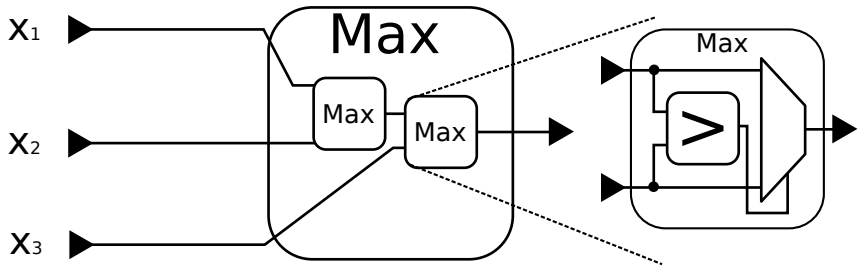

FIG. 2.6: Représentation générale d'un neurone MaxPooling formel. La sortie de ce neurone est égale à la plus grande valeur numérique à son entrée. Pour déterminer la valeur maximale de ses entrées, ce neurone est composé de comparateurs disposés en arbre.

$$OP_{MaxPooling_{formel}} = O \times C \times C_u = O \times C \times C \times \lceil log_2(C) \rceil \tag{2.8}$$

Nous pouvons à présent exprimer la complexité en opérations atomiques des couches de neurones formels. Nous allons maintenant nous intéresser au calcul impulsionnel, pour exprimer la complexité des couches impulsionnelles en matière d'opérations atomiques ce qui nous permettra de comparer les deux modèles : formels et impulsionnels.

Années	Modèle	Ref et calcul du potentiel
1907	Integrate and fire	[24] Intégration de courant
1952	Hodgkin-Huxley	[25] Intégration de multiples sources de courant
1961	Fitzhugh-Nagumo	[26] Intégration d'un courant avec un retour du potentiel de la membrane
1965	Leaky integrate-and-fire	[27] Intégration et fuite de courant

TAB. 2.1: Les modèles de neurones impulsionnels pré-années 80.

2.4 Le modèle impulsionnel, propriété et équivalence

Nous allons étudier la complexité du modèle impulsionnel en l'exprimant en opérations atomiques OP. Nous allons dans un premier temps faire un historique des différents modèles impulsionnels pour en déduire les principes généraux de fonctionnement et les raisons de l'utilisation d'une transposition du formel vers l'impulsionnel. Dans un second temps, nous étudierons les équivalences entre le modèle formel et le modèle impulsionnel pour poser les conditions de passage entre les deux modèles.

Pour finir, nous exprimerons la complexité calculatoire des modèles impulsionnels pour chaque type de couche d'un CNN. Nous aborderons aussi la question de l'instanciation matérielle de calculateur impulsionnel.

2.4.1 Le modèle impulsionnel

L'origine des modèles impulsionnels est liée aux études biologiques du comportement des neurones. Ces études ont permis de formaliser le comportement de la réponse d'un neurone à un jeu de stimuli d'entrées des modèles. Ces différents modèles modélisent plus ou moins finement les réactions biologiques des neurones. Une liste non exhaustive des premiers modèles ainsi que leurs publications associées sont présentées dans le tableau 2.1.

On peut diviser ces différents modèles en deux classes : les modèles de simulations biologiques et les modèles computationnels. Entre ces deux types, on note une différence en matière de complexité de la fonction de transfert. Les modèles de simulations biologiquement proches tentent de reproduire le plus fidèlement le comportement des neurones biologiques. Le plus souvent ce type de modèle fait appel à des opérateurs complexes permettant de simuler les différentes interactions

chimiques au sein d'un neurone. Les modèles computationnels sont des modèles approchés souvent calculatoirement moins complexes en matière d'opérateur. Ainsi on peut distinguer de l'état de l'art deux modèles computationnels classiquement utilisés : le "Leaky integrate and fire" (LIF) et une version simplifiée "l'integrate and fire" (IF), figure 2.8.

Un deuxième aspect des réseaux de neurones impulsionnels est le codage de l'information. Dans ce type de réseaux, les données sont transmises sous forme d'impulsion unitaire. Le codage de l'information peut alors être de plusieurs natures en phase et/ou en fréquence et/ou en délais etc...

Il est à noter que les opérations effectuées dans un neurone impulsionnel ne sont pas des opérations vectorielles. Ce qui implique que les architectures formelles voir la section 3.2, ne sont pas adaptées pour le calcul impulsionnel. Autrement dit, le calcul impulsionnel nécessite la mise en place d'une architecture matérielle dédiée.

Nous devons maintenant statuer sur le modèle impulsionnel et le codage à utiliser. Pour cela, nous nous appuyons sur les travaux de la transposition du formel vers l'impulsionnel. Nous avons choisi d'utiliser les techniques de transposition pour passer d'un réseau formel à un réseau impulsionnel pour profiter des performances de l'apprentissage formel. Nous allons voir ces méthodes de transposition et leur conséquence sur le choix du modèle et du codage de l'information.

2.4.2 Du formel à l'impulsionnel

Les techniques de transposition que nous utilisons sont basées sur la représentation fréquentielle des valeurs en entrée du neurone. La transposition fréquentielle des valeurs d'entrées x_i est en fonction de deux paramètres : la fréquence min f_{min} et de la fréquence max f_{max} des impulsions, voir l'équation 2.9 et la figure 2.7.

$$n_i = f_{min} + x_i \times (f_{max} - f_{min}), x_i \geq 0 \tag{2.9}$$

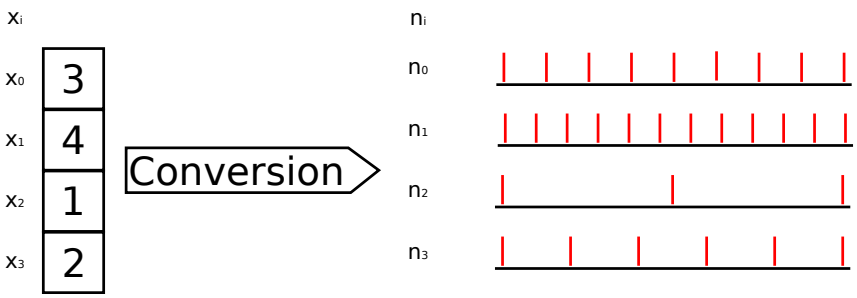

FIG. 2.7: Conversion d'une matrice 2×2 eléments en 4 train d'impulsions. Les valeurs numériques des pixels sont converties linéairement en une fréquence d'impulsions.

Ce principe de la transposition fréquentielle du formel vers l'impulsionnel a été dans un premier temps formalisé par J. A. Pérez-Carrasco *et al* pour des neurones du

type LIF utilisant une fonction d'activation en tangente hyperbolique [8]. Cependant, nous avons montré que ce principe peut être étendu pour le modèle impulsionnel IF avec une fonction d'activation du type ReLU.

Ainsi si on considère un neurone formel, on note qu'il effectue principalement deux opérations. Premièrement, une opération de MAC entre un vecteur d'entrée x_i et un jeu de pondérations $W_{i,j}$. Deuxièmement, l'application au résultat des MAC d'une fonction d'activation $H(x)$ ce qui nous donne l'équation 2.10.

$$y_j = H(\sum_i (w_{ij} \times x_i)) \tag{2.10}$$

Ainsi, si on considère un seuil noté x_{seuil}, le nombre d'impulsions n_i provenant de l'entrée x_i et n_j le nombre d'impulsions de sortie provenant de la sortie y_j durant une période T_{acc} on obtient l'équation 2.11.

$$\frac{nj}{T_{acc}} = \lfloor \frac{\sum_i (w_{ij} \times x_i)}{x_{seuil}} \rfloor \frac{1}{T_{acc}} \tag{2.11}$$

En considérèrant le cas d'impulsions n_j est n_i signées, le neurone j doit fournir un déclenchement positif et négatif. Il existe alors deux seuils, un positif $x_{seuil+} > 0$ et un négatif $x_{seuil-} < 0$. On considère que le franchisent de l'un des seuils implique remise à une valeur minimale $x_{seuil_{signe(n_j)}}$ de l'intégration et non à zéro comme dans [8], pour éviter la perte de précision après plusieurs impulsions. En considérant $n_j >> 1$, il est possible d'approximer l'équation 2.11 en l'équation 2.12.

$$\frac{nj}{T_{acc}} \approx \frac{\sum_i (w_{ij} \times x_i)}{\mid x_{seuil_{signe(n_j)}} \mid \times T_{acc}} \tag{2.12}$$

En ajoutant une période réfractaire T_R au modèle impulsionnel et en utilisant l'équation 2.12 nous obtenons l'équation 2.13

$$\frac{nj}{T_{acc}} \leq \frac{1}{T_R} \approx \frac{1}{T_R} \times h_{SAT}(\frac{T_R \times \sum_i (w_{ij} \times x_i)}{\mid x_{seuil_{signe(n_j)}} \mid \times T_{acc}}) \tag{2.13}$$

$$h_{SAT}(x) = \begin{cases} -1 \text{ si } x < -1 \\ x \text{ si } -1 \leq x < 1 \\ 1 \text{ si } x \geq 1 \end{cases}$$

Nous allons maintenant nous intéresser à la transposition des fonctions d'activations tanh et ReLU.

Approximation de la tanh pour l'inférence, la fonction de tangente hyperbolique peut être instanciée avec la fonction h_{SAT} tout en gardant de bonnes performances en pratique. Mais il existe aussi une approximation directe entre

le modèle formel et l'équation 2.13, avec $x_i \equiv \frac{n_i}{T_{acc}}$, $y_i \equiv \frac{n_j}{T_{acc}}$, $x_{seuil+} = 1$, $x_{seuil-} = -1$ et $\frac{1}{T_R}$ la valeur de saturation.

Équivalence ReLU est plus évident, car elle ne nécessite pas l'utilisation de la période réfractaire en plus du modèle IF. L'équation 2.12 peut donc être utilisée avec $x_i \equiv \frac{n_i}{T_{acc}}$, $y_i \equiv \frac{n_j}{T_{acc}}$, $x_{seuil+} = 1$ et $x_{seuil-} = -\infty$. Le neurone ne peut donc pas se déclencher si son intégration est négative.

Il est à noter que les pondérations sont identiques aux pondérations originales, ainsi les optimisations de la dynamique des pondérations sont applicables aux réseaux impulsionnels sans aucune modification. On peut donc considérer la même limite de 8 bits pour la précision des pondérations pour les réseaux impulsionnels que pour les réseaux formels. Au niveau du codage de l'information, nous devons utiliser un codage fréquentiel de l'information pour satisfaire les conditions de la transposition. Nous avons choisi d'utiliser le modèle ReLU au vu de son utilisation courante et de son efficacité [28, 29, 30], nous devons donc utiliser le neurone IF. Finalement, nous utiliserons dorénavant le neurone impulsionnel IF avec des pondérations sur 8 bits et un codage fréquentiel de l'information.

2.4.3 Le LIF et la complexité du IF

Nous allons présenter le fonctionnement des neurones suivant le modèle LIF. Nous avons choisi de partir du modèle LIF pour exprimer la complexité du IF pour deux raisons. Premièrement le LIF est un modèle répandu dans les architectures dédiées à l'impulsionnel [4, 31, 32]. Deuxièmement le modèle IF peut être facilement déduit du modèle LIF. Nous avons aussi choisi de présenter une généralisation du LIF avec deux seuils afin de présenter le fonctionnement le plus général possible.

Nous commencerons par expliciter les différents composants d'un LIF. Ceci nous permettra d'introduire les différentes possibilités d'implémentation matérielles. Finalement, nous pourrons ainsi exprimer la complexité des IF.

Les impulsions en entrées d'un neurone sont dites pré-synaptique. Ces impulsions correspondent à des Dirac, elles sont donc unitaires et peuvent être signées. Une fois pondérées ces impulsions sont dites post-synaptique et modifie le potentiel de la membrane du neurone (son intégration). Lors du déclenchement d'un neurone, celui-ci propage une impulsion en sortie (au niveau de son axone).

Dans une instanciation analogique du LIF, lors de la réception d'impulsions, celles-ci sont pondérées et vont modifier le potentiel de la membrane du neurone $V_m(t)$ (l'intégration). Les impulsions sont reçues par le cœur des neurones elles seront alors intégrées en un courant $I(t)$ en fonction de la capacité de la membrane Cm . Puis une fuite est appliquée à la valeur d'intégration en fonction de la résistance de la membrane Rm. Ces opérations sont définies par l'équation 2.14. Pour finir, si le potentiel de la membrane est supérieur ou inférieur aux seuils, le neurone

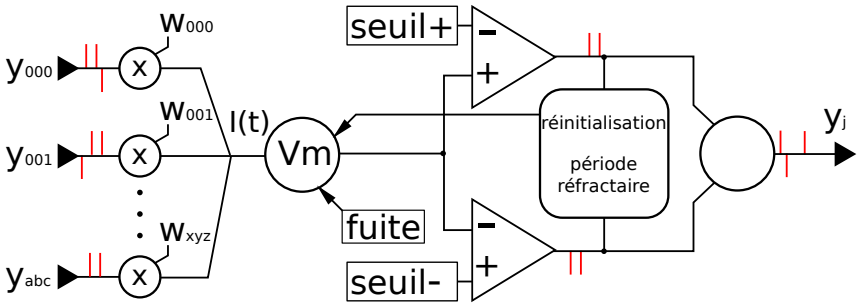

FIG. 2.8: Représentation d'un neurone impulsionnel du type LIF, les entrées sont des impulsions signées intégrées en un potentiel de membrane Vm soumis à une fuite. Le potentiel de membrane est comparée au seuil positif et négatif pour en déduire les impulsions de sortie. Le bloc de réinitialisation et de période réfractaire modifie le potentiel V_m en fonction des déclenchements du neurone.

se déclenche et envoie une impulsion signée en fonction du seuil traversé. Post-déclenchement, l'intégration est remise à une valeur de référence. Il est possible d'ajouter une période réfractaire pour fixer la fréquence maximum de déclenchement du neurone.

$$I(t) - \frac{V_m(t)}{R_m} = C_m \frac{dV_m(t)}{dt} \tag{2.14}$$

Dans le cas de la réalisation analogique, l'intégration des impulsions correspond à la charge ou la décharge d'une capacité soumise à des impulsions de courant. La valeur du courant codant la valeur de la pondération. La fuite est opérée par une résistance et la fonction de seuil par un comparateur de tension, voire figure 2.9. La remise à la référence de l'intégration est effectuée par un interrupteur contrôlé par les impulsions de sorties.

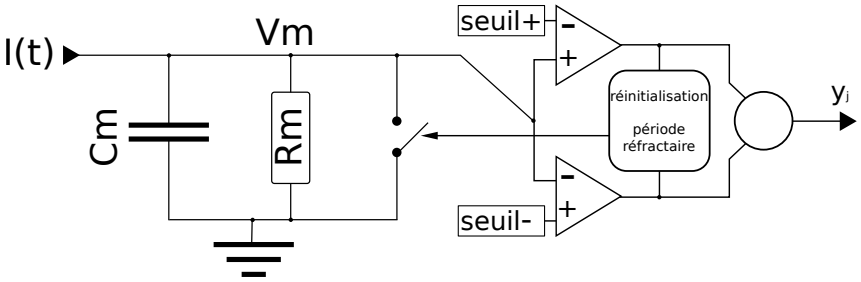

FIG. 2.9: Circuit analogique d'un neurone du type LIF, les entrées sont des impulsions signées intégrées en un potentiel de membrane V_m par une capacité C_m soumis à une fuite imposée par la résistance R_m. Le potentiel de membrane est comparé au seuil positif et négatif pour en déduire les impulsions de sortie. Le bloc de réinitialisation et de période réfractaire modifie le potentiel V_m en fonction des déclenchements du neurone.

Dans le cas de la réalisation numérique, la capacité est remplacée par un ac-

cumulateur, la résistance par un soustracteur et une horloge, et les seuils par des comparateurs numériques. Les neurones LIF numériques utilisent donc une horloge commune et un soustracteur exponentiel pour calculer la fuite. Le neurone reçoit en entrée des impulsions unitaires qui vont être pondérées et propagées sur un bus numérique. Celles-ci sont cumulées et on applique à cette valeur une fuite en fonction du temps d'arrivée de l'impulsion à l'aide du soustracteur et d'un compteur, le résultat est ensuite comparé aux seuils. Le tout est contrôlé par une machine à états gérant la remise à zéro de l'intégration et la période réfractaire, voir figure 2.10.

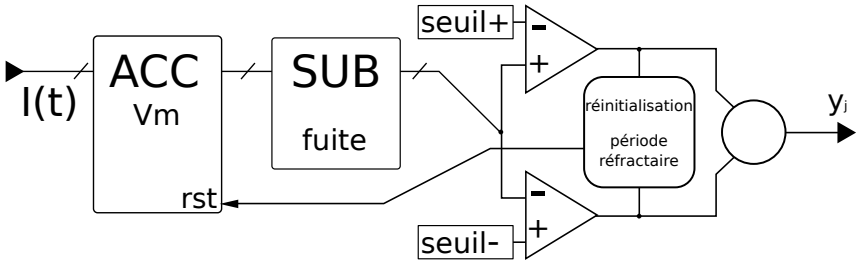

FIG. 2.10: Circuit numérique d'un neurone du type LIF, les entrées sont des impulsions signées intégrées en un potentiel de membrane V_m par un accumulateur soumis à une fuite. La potentielle de membrane est comparée au seuil positif et négatif pour en déduire les impulsions de sortie. Le bloc de réinitialisation et de période réfractaire modifie le potentiel V_m en fonction des déclenchements du neurone.

Connaissant le fonctionnement du neurone LIF et son instanciation il est possible d'exprimer la complexité du neurone IF. Pour cela il faut considérer le modèle LIF sans l'opération de fuite. Ainsi le neurone IF ne nécessite pas d'horloge commune ni de soustracteur exponentiel.

Pour le calcul de la complexité, je considère une implémentation numérique d'un neurone IF. La complexité calculatoire du neurone impulsionnel dépend alors du nombre d'impulsions pré-synaptiques reçues par ce neurone $NbImp_{pré}$ et du nombre de neurones connectés à la même entrée $NbC_{neuro}(couche)$ et de la complexité des opérations faites dans le neurone IF notées C_i. La complexité des opérations C_i dépend quant à elle de la précision des pondérations en bit Nbi_{bit}. On peut aussi exprimer le nombre d'impulsions post-synaptiques avec l'équation 2.15, qui est égale au nombre d'opérations faites lors de la réception d'une impulsion.

$$NbImp_{post} = NbImp_{pré} \times NbC_{neuro}(couche) \tag{2.15}$$

En considérant que l'impulsion reçue par le neurone est équivalente à une opération d'accumulation suivie de deux comparaisons du résultat aux deux seuils. En considérant une accumulation Nbi_{bit} correspond à $Nbi_{bit}-1$ full-adder et 1 half-adder et une comparaison Nbi_{bit} est identique en termes d'opérations à un additionneur. Et en considérant la même limite de précision de 8 bits en phase d'inférence que

Operation MAC		Operation ACC	
MULT 16 bits	1,17 pJ	ADD 8 bits	0,03 pJ
ADD 32 bits	0,1 pJ*		
Total	1,27 pJ	Total	0,03 pJ

TAB. 2.2: Comparatif de la consommation d'énergie d'une opération ACC et de MAC estimée pour du 45nm 0,9V [5]. * valeur interpolée.

pour le modèle formel la complexité du neurone IF en opération atomique C_i est donnée par l'équation 2.16.

En utilisant le tableau 2.2 il nous est possible de confirmer notre calcul de C_f et C_i. Ainsi, on peut déduire la complexité en opération atomique tC_f formelle sans seuil à 8 bits au vu de la consommation des opérations

$$1,27 \times \frac{C_i(8)}{3 \times 0,03} = tC_f(16) = 317,5OP$$

Or, selon le calcul théorique de $C_f(16)$ sans le seuil à 8 bits nous avons $263,5OP$, voir l'équation section 2.3.3. Il y a donc une différence de $54OP$ ce qui montre que notre calcul des complexités C_f et C_i est proche de la réalité physique.

$$\begin{cases} C_i = 3 \times (Nbi_{bit} - 1 + \frac{1}{2}) \text{ Si } Nbi_{bit} \leq 8 \\ C_i = 3,5 \text{ Sinon} \end{cases} \tag{2.16}$$

Ainsi, la complexité calculatoire d'un neurone impulsionnel est donc donnée par l'équation 2.17.

$$NbImp_{pré} \times NbC_{neuro}(couche) \times C_i = NbImp_{post} \times C_i \tag{2.17}$$

À la différence du modèle formel, le modèle impulsionnel sauvegarde pour chaque neurone sa valeur interne, l'intégration. Ainsi pour le modèle impulsionnel il faut considérer en plus de la complexité calculatoire une empreinte mémoire due à la sauvegarde de l'intégration. On peut estimer cette empreinte mémoire en fonction de la précision des pondérations en nombre de bits Nbi_{bit}.

En considérant que le neurone doit intégrer des impulsions pondérées sur Nbi_{bit}, que l'intégration est signée en complément à deux. Nous avons choisi de faire une transposition d'un ReLU, ce qui implique que le seuil négatif est égal à $-\infty$, voir section 2.4.2. Il nous faut donc une précision suffisante pour approximer la limite en $-\infty$.

Nous avons constaté sur un certain nombre de simulation de réseaux classiques qu'une précision d'intégration de $2 \times Nbi_{bit}$ est suffisante pour modéliser la limite en $-\infty$ du modèle IF. Dans ces conditions, il faut que $C \leq 2^{Nbi_{bit}+1}$ pour éviter une saturation négative de l'intégration à la réception de C impulsions ayant toutes

des pondérations égale à $-2^{Nbi_{bit}-1}$, ce qui reste un cas peu probable. Ainsi, pour $Nbi_{bit} = 8bits$ alors $C \leq 512$ ce qui représente un filtre carré de 22×22 éléments. Après avoir exprimé la complexité d'un neurone IF nous allons nous intéresser à l'expression de la complexité des couches de CNN impulsionnels.

2.4.4 Calcul et complexité des couches de CNN impulsionnels

Au niveau de la couche de convolution, on peut estimer sa complexité calculatoire pour calculer la réponse à une impulsion $OP_{Conv_{impul}}$. Cette complexité est égale au nombre de neurones connectés à la même entrée et donc qui réceptionnent la même impulsion simultanément, multiplié par le nombre d'impulsions en entrées $NbImp_{pré}$. Je pose comme approximation que le nombre de neurones connectés à la même entrée dans une couche de convolution, soit égale à C. Cette approximation est d'autand plus vraie que la taille du filtre est négligeable par rapport à l'entrée. Ainsi la complexité calculatoire d'une couche de convolution soumise à $NbImp_{pré}$ impulsions nous est donnée par l'équation 2.18. L'empreinte mémoire de cette couche est quant à elle liée au nombre de neurones total de cette couche ce qui nous donne l'équation 2.19.

$$OP_{Conv_{impul}} = NbImp_{pré} \times C \times C_i \tag{2.18}$$

$$Mem_{Conv_{impul}} = 2 \times Nbi_{bit} \times O \tag{2.19}$$

Dans une couche entièrement connectée, la logique est légèrement différente, car tous les neurones de cette couche sont connectés aux mêmes entrées. De plus, la connexion d'un neurone sur la couche précédente est complète, il n'y a pas de notion de Stride. Ce qui nous donne l'équation 2.20 pour la complexité calculatoire et l'équation 2.21 pour l'empreinte mémoire.

$$OP_{FC_{impul}} = NbImp_{pré} \times O_{elmZ} \times C_i \tag{2.20}$$

$$Mem_{FC_{impul}} = 2 \times Nbi_{bit} \times O_{elmZ} \tag{2.21}$$

Dans le cas de la couche de MaxPooling, le codage fréquentiel de l'information à un impact sur son fonctionnement et son empreinte mémoire. Ainsi la sortie d'un neurone de ce type de couche est égale à la plus grande valeur de son champ récepteur. L'équivalent impulsionnel avec un codage fréquentiel de ce comportement est que la sortie du neurone correspond à sa plus haute fréquence d'entrée. Ainsi pour chacune de ses entrées le neurone doit mémoriser leurs activités pour les comparer. La complexité de cette comparaison d'activité est la même qu'en formel car

il y a autant de compteurs d'activité que d'entrées, voir équation2.22. L'équation 2.23 nous donnent l'empreinte mémoire de la sauvegarde des activités d'entrées du MaxPooling impulsionnel.

$$OP_{MaxPooling_{impul}} = NbImp_{pré} \times C \times C \times \lceil log2(C) \rceil \tag{2.22}$$

$$Mem_{MaxPooling_{impul}} = \lceil log2(C) \rceil \times O \times C \tag{2.23}$$

Nous avons donc vu l'expression de la complexité calculatoire et de l'empreinte des différentes couches d'un CNN impulsionnel. On remarque que la complexité calculatoire dépend de la topologie de la couche, mais aussi du nombre d'impulsions $NbImp_{pré}$ reçues par la couche. Or ce nombre d'impulsions $NbImp_{pré}$ peut varier en fonction de la topologie du réseau et des données en entrées ce qui implique que la complexité calculatoire des réseaux impulsionnels ne peut pas être déterminée a priori. Dans la prochaine section, nous allons étudier les différences entre ces deux modèles et conclure sur l'apport théorique du modèle impulsionnel sur la complexité calculatoire.

2.5 Évaluation des modèles

Dans cette section, je vais dans un premier temps évaluer l'apport théorique de l'utilisation du modèle impulsionnel pour la réduction du nombre d'opérations atomiques. Puis en utilisant les définitions d'opérations atomiques, je poserais les équations qui nous permettrons de comparer les différentes architectures.

2.5.1 Pertinence de l'utilisation du modèle impulsionnel

Nous avons déterminé la complexité calculatoire de toutes ces différentes couches d'un CNN formel et impulsionnel. Nous avons montré que la complexité impulsionnelle est fonction du nombre d'impulsions reçues en entrée de la couche $NbImp_{pré}$. Cependant, le nombre d'impulsions reçues en entrée de la couche ne peut pas être déterminé a priori.

J'ai donc décidé d'exprimer la comparaison en matière de nombre d'impulsions pré-synaptiques pour chaque type de couche par entrée $NbImp_{Pré_{Max}}(couche)$ pour que la couche impulsionnelle effectue le même nombre d'opérations atomique que une couche formelle identique. Ainsi, si $NbImp_{Pré_{Max}}(couche)$ est supérieur à 1 la couche impulsionnelle peut traitée plus d'une impulsion par entrée et effectue moins d'opérations que la même couche en formelle. Cette métrique nous permet d'évaluer la faisabilité de la réduction de complexité calculatoire car avec le codage fréquentiel des impulsions, il risque d'être complexe de garantir moins d'une impulsion

pré-synaptique par entrée. Donc si $NbImp_{Pré_{Max}}(couche) \leq 1$ l'utilisation du codage impulsionnel ne va certainement pas permettre la réduction de la complexité calculatoire.

Je considère par la suite que la matrice 2D d'entrées est suffisamment grande par rapport au filtre pour faire l'approximation $\frac{O}{I} \approx 1$. Pour la couche de convolution et en utilisant les équations 2.18 et 2.6, nous obtenons la relation 2.24 qui fixe la limite en nombre d'impulsions en entrées d'une couche de convolution impulsionnelle pour obtenir un gain en matière de complexité calculatoire par rapport à la même convolution en formel.

$$NbImp_{Pré_{Max}}(Conv) = \frac{O}{I} \times \frac{C_f}{C_i} \approx \frac{C_f}{C_i} \tag{2.24}$$

Pour la couche entièrement connectée FC, en utilisant les équations 2.20 et 2.7, nous obtenons la relation 2.25 qui fixe la limite en nombre d'impulsions en entrées d'une couche FC impulsionnels pour obtenir un gain en matière de complexité calculatoire par rapport à la même couche entièrement connectée formel.

$$NbImp_{Pré_{Max}}(FC) = \frac{I}{I} \times \frac{C_f}{C_i} = \frac{C_f}{C_i} \tag{2.25}$$

Pour la couche MaxPooling et en utilisant les équations 2.22 et 2.8, nous obtenons la relation 2.26 qui fixe la limite en nombre d'impulsions en entrées d'une couche de MaxPooling impulsionnelle pour obtenir un gain en matière de complexité calculatoire par rapport à la même MaxPooling en formel. La connectivité de la couche MaxPooling étant identique à celle de la convolution nous considérerons la même approximation de la taille de l'image d'entrée.

$$NbImp_{Pré_{Max}}(MaxPooling) = \frac{O}{I} \approx 1 \tag{2.26}$$

On constate que $NbImp_{Pré_{Max}}(Conv)$ de la convolution est approximativement égal à $NbImp_{Pré_{Max}}(FC)$ du FC et si l'on considère les mêmes précisions pour le calcul de C_i et de C_f nous obtenons pour ces deux types de couches au maximum 3 impulsions pré-synaptiques par entrée. Ce résultat nous montre qu'il est potentiellement possible de réduire le nombre d'opérations à effectuer pour le calcul d'une couche de convolution ou entièrement connectée par utilisation du modèle impulsionnel.

Dans le cas du MaxPooling on obtient 1 impulsion pré-synaptique par entrée, or en considèrant le codage fréquentiel il semble difficile d'obtenir une réduction du nombre d'opérations atomiques par utilisation du MaxPooling impulsionnel à la place d'un MaxPooling formel. Ce résultat nous montre donc que l'opération de MaxPooling impulsionnelle nécessite une optimisation, que nous étudions dans la section 4.4.

Finalement, le changement de modèles nous permet potentiellement de diminuer la complexité calculatoire d'un CNN. Cependant, notre étude ne prend pas en compte les caractéristiques des architectures. Je vais donc étendre cette analyse en incluant les caractéristiques des architectures pour mettre en place notre modèle de comparaisons entre les différentes architectures formelles et impulsionnelles.

2.5.2 Comparaison des architectures formelles et impulsionnelles

Dans le cas de la comparaison entre deux architectures impulsionnelles, comme nous avons vu dans la section 2.4.4, la complexité de calcul dépend du nombre d'impulsions pré-synaptique $NbImp_{pré}$ reçues. Or $NbImp_{pré}$ ne peut pas être déterminé a priori. Cependant, considérant le même nombre d'impulsions pré-synaptique en entrée des deux architectures il est possible de comparer leur performance exprimée en opération atomique par seconde OP/s et par Joule OP/J directement.

Pour calculer les OP/s pour une architecture impulsionnelle, il nous faut le nombre d'accumulations traitées par seconde par l'architecture soit

$$ACC/s = \frac{NbPara_{neuro} \times NbPara_{impul}}{t_{traitement}}$$

avec $NbPara_{neuro}$ le nombre de neurones traités parallèlement et $t_{traitement}$ le temps de traitement . Nous convertissons ACC en utilisant C_i. Et pour calculer le nombre OP/J nous utiliserons la consommation de l'architecture impulsionnelle en Watt W_{impul}.

Dans le cas de la comparaison entre deux architectures formelles, comme nous l'avons vu dans la section 2.3.3, il est possible d'exprimer en fonction d'une couche le nombre d'opérations à effectuer. Pour les mêmes couches, on peut exprimer leurs performances en opérations atomiques par seconde OP/s et par Joule OP/J directement. Pour calculer les OP/s nous utilisons le nombre de MAC/s effectué par une architecture puis convertissons les MAC en OP en utilisant C_f. Et pour calculer le nombre OP/J nous utiliserons la consommation de l'architecture impulsionnelle en Watt W_{formel}.

La comparaison des architectures formelles et des architectures impulsionnelles n'est pas évidente, car il n'y a pas de lien entre le nombre d'impulsions pré-synaptiques reçues par une architecture impulsionnelle $NbImp_{pré}$ et le nombre de MAC faites par une architecture formelle. De plus, le codage de l'information et les opérateurs utilisés sont très différents entre ces deux modèles. Cependant, pour un nombre d'opérations atomiques $OP_{couche_{formel}}$, on peut en déduire le temps de calcul t_f et l'énergie consommée J_f par une architecture formelle. Ces relations ne sont vraies que pour les couches de convolutions et de FC, car elles sont basées sur le calcul de la réponse d'un neurone. Pour la couche de MaxPooling il est difficile de déterminer

le nombre de comparaisons faites par seconde pour une architecture donnée, nous ne considérerons donc pas ce type de couche pour nos comparaisons. Le temps de traitement et l'énergie consommée pour les deux types d'architectures sont alors :

$$t_f = \frac{OP_{couche_{formel}}}{C_f \times MAC/s}$$

$$J_f = W_f \times tF_{apli} = \frac{W_f \times OP_{couche_{formel}}}{C_f \times MAC/s}$$

On peut faire de même pour les architectures impulsionnelles en fonction du nombre d'impulsions pré-synaptique $NbImp_{pré}$ reçues .

$$t_i = \frac{OP_{couche_{impul}} \times NbImp_{Pré}}{C_i \times ACC/s}$$

$$J_i = W_i \times \frac{OP_{couche_{impul}} \times NbImp_{Pré}}{C_i \times ACC/s}$$

Finalement, on exprime le nombre d'impulsions pré-synaptiques par entrée t : $NbImp_{pré}/entrées$ et J : $NbImp_{pré}/entrées$ que peut traiter une architecture impulsionnelle pour être aussi efficace que l'architecture formelle de référence en temps de calcul et/ou en énergie consommée. Avec les équations du nombre d'opérations en fonction du type de couche, voir les sections 2.4.4 et 2.3.3. En considérant que la matrice d'entrées est suffisamment grande par rapport au filtre pour faire l'approximation $\frac{O}{I} \approx 1$, l'expression du nombre d'impulsions pré-synaptiques par entrée que peut traiter une architecture impulsionnelle pour être aussi efficace que l'architecture formelle de référence en temps de calcul et en énergie consommée est identique pour les couches de convolution et de FC et est donnée par les équations 2.27 et 2.28.

$$t : NbImp_{pré}/entrées = \frac{ACC/s}{MAC/s} \tag{2.27}$$

$$J : NbImp_{pré}/entrées = \frac{W_f}{W_i} \times \frac{ACC/s}{MAC/s} \tag{2.28}$$

Pour une architecture impulsionnelle avec $10GACC/s$ et qui consomme $0,5W$ et une architecture formelle avec $10GMAC/s$ et qui consomme $1W$, nous allons avoir $t : NbImp_{pré}/entrées = 1$ et $J : NbImp_{pré}/entrées = 2$.

On note que les paramètres C_i et C_i impactent seulement la comparaison entre deux architectures du même type. La comparaison des architectures formelles et impulsionnelles ne dépend que des caractéristiques de leurs architectures. Nous avons à présent tous les outils pour comparer les architectures entre elles.

2.6 Conclusion

Les solutions de comparaison entre les architectures formelles et les architectures impulsionnelles de l'état de l'art ne sont pas adaptées à notre problématique d'inférence dans des architectures embarquées. J'ai donc proposé ma méthode d'estimation de la complexité formelle et impulsionnelle dans le cas de l'inférence. Nous avons montré que pour les couches de convolution et FC le modèle impulsionnel permet en théorie une réduction du nombre d'opérations atomiques par rapport au modèle formel. Cette possible réduction n'est pas constatée pour la couche de MaxPooling, cette couche nécessite donc une optimisation.

Puis j'ai mis en place une méthode pour comparer les différents types d'architectures entre elles en utilisant la notion d'opérations atomiques OP. Nous pouvons à présent exprimer le nombre d'impulsions que peut traiter une architecture impulsionnelle par entrée pour être plus efficace en temps de traitement et/ou en énergie consommée par rapport à une architecture formelle. Et nous pouvons comparer les architectures du même type pour le traitement de l'inférence des CNN.

Nous allons nous intéresser à l'état de l'art des architectures impulsionnelles et formelles dans le but de mettre en place une comparaison de leur efficacité en temps de traitement et en puissance consommée pour identifier les meilleures approches du calcul des CNN impulsionnels.

Chapitre 3

Architectures et analyse de performance

Contents

3.1 Introduction

Nous avons étudié les deux modèles de neurone, le modèle formel et le modèle impulsionnel et nous avons montré l'apport théorique du modèle impulsionnel par rapport au modèle formel sur la complexité calculatoire. Nous avons mis en place notre méthode de comparaison entre les architectures formelles et des architectures impulsionnelles dans le cas du calcul de l'inférence des CNN.

Dans ce chapitre, nous allons étudier l'état de l'art des architectures formelles et des architectures impulsionnelles. Je vais proposer dans un premier temps une analyse de performance entre les architectures du même type. Cette analyse nous permettra d'identifier les architectures formelles et impulsionnelles les plus performantes en matière OP/s et Op/J pour l'inférence d'un CNN.

Par la suite, je comparerai les architectures formelles et les architectures impulsionnelles pour identifier quelles méthodes d'implémentation des CNN impulsionnelle les plus efficaces et les éventuelles optimisations possibles.

3.2 Architectures de calcul formel

Comme nous l'avons vu dans la section 2.3.1, le neurone formel effectue des opérations de MAC qui peuvent être vue comme des opérateurs vectoriels. Plusieurs architectures sont envisageables pour le calcul efficace de l'inférence formelle : GPU, CPU et architecture dédiée ASIC ou FPGA. On peut alors identifier deux catégories d'architectures : Premièrement les architectures génériques du type CPU, GPU. Deuxièmement les architectures dédiées avec une instanciation ASIC ou FPGA. Pour chacune de ces architectures, nous analyserons son fonctionnement pour exprimer leur performance en matière d'opérations atomiques.

3.2.1 Les architectures formelles génériques

Les architectures du type GPU ont été conçues originellement pour le traitement d'images. Elle s'adapte donc parfaitement au traitement des vecteurs dans les réseaux convolutionnels à base de neurones formels. Aujourd'hui les GPU représentent la solution classiquement utilisée pour le calcul de l'inférence, mais aussi pour l'accélération de l'apprentissage [33, 34].

L'efficacité de traitement avec un GPU est influencée par la taille du paquet d'images qu'il peut traiter en parallèle (Batch Size). Or nous considérons une application embarquée, où les images reçues proviennent d'un capteur. Dans ce type d'application, il est difficile d'accumuler plusieurs images en un paquet, car cette opération engendre une latence égale au nombre d'images dans le paquet. Nous considérerons donc pour notre analyse des performances des GPU un Batch Size égal à 1.

Architecture	MAC/s	Consommation	Nbf_{bit}	MAC/J
Titan X (FP32)	283G	164 W	32	1,7G
Tegra X1 (FP32)	32,9G	5,5 W	32	5,9G
Tegra X1 (FP16)	46,9G	5,1 W	16	9,1G
Core i7 6700K (FP32)	43,4G	49,7 W	32	0,8G
Xeon E5-2698 v3 (FP32)	53,2G	111,7 W	32	0,4G

TAB. 3.1: Estimation des performances calculatoires en $GMAC/s$ et de la consommation en Watt des architectures GPU NVIDIA et CPU pour le calcul du réseau AlexNet [6].

En utilisant les estimations de NVIDIA[6] sur le réseau AlexNet [33] d'une complexité d'environ $0,725 GMAC/image$, nous sommes en mesure d'estimer les valeurs de Nbf_{bit}, MAC/s et W_f pour plusieurs architectures GPU et CPU. Le tableau 3.1 résume les différentes caractéristiques des architectures génériques GPU et CPU.

On remarque qu'à consommation équivalente les architectures GPU sont plus performantes que les architectures CPU. Cependant, les architectures types GPU peuvent être sous-optimales pour deux raisons. Premièrement, la répartition des calculs peut être sous-optimale, mais cela est difficilement évaluable. Mais, surtout les opérateurs implémentés comme les opérateurs flottants sont peu utiles en inférence et matériellement coûteux, voir section 2.3.2. Il est à noter que ce problème de précision est en cours de correction avec les dernières versions de GPU NVIDIA pouvant faire de l'inférence sur 8 bits, mais actuellement il y a peu d'information sur leur performance.

3.2.2 Les architectures formelles dédiées

La deuxième catégorie regroupe les architectures dédiées aux réseaux de neurones formels. Ce type d'architecture a été créé pour atteindre de plus grandes efficacités énergétiques et surfaciques. On peut classer ces architectures en deux classes : les architectures SIMD (Single Instruction ont Multiple Data) et les Dataflow.

Les architectures Dataflow sont représentées entre autres par les architectures Neuflow [35] et DianNao [36]. Elles sont composées de blocs , utilisés pour effectuer une opération atomique d'un réseau telle qu'une multiplication, addition ou fonction non linéaire, voir la figure 3.1. Les fonctions à effectuer sont décomposées en opération atomique, chaque opération est placée dans un bloc, et pour finir les blocs sont reliés par un système de multiplexeur. Le traitement en flux dans ce type d'architecture permet théoriquement de minimiser la mémoire, car il n'y a pas de sauvegarde d'état intermédiaire et de minimiser le temps de traitement. Cependant, l'implémentation d'un réseau est fortement liée à l'architecture matérielle, rendant cette approche peu flexible.

Architecture	MAC/s	Consommation	Nbf_{bit}	MAC/J
XETAL-II – ASIC 90nm	107G	600 mW	16	178G
NeuFlow – FPGA	147G	10 W	16	14,7G
NeuFlow – ASIC 45nm	160G	5W	16	32G
DianNao – ASIC 65nm	≈ 226G	485 mW	16	465,9G
PNeuro – FPGA	12,8G	2,5 W	8	5,1G
PNeuro – ASIC 28nm	64G	150 mW	8	426,6G
nn-X	≈ 113G	4 W	32	28,2G

TAB. 3.2: Comparatif des efficacités énergétiques en fonction des plates-formes pour un réseau contenant 60 neurones dans la couche cachée cumulant 450 Kops.

Les architectures SIMD, ou l'on trouve les architectures EV processor, Lneuro [37], XETAL-II [38] et Pneuro, permettent d'exécuter une instruction sur plusieurs données simultanément, voir figure 3.1 Ces structures sont adaptées au calcul de réseaux de neurones, car les neurones effectuent les mêmes opérations, mais sur des données d'entrées différentes. Ce sont des architectures flexibles, car la topologie du réseau n'est pas liée à l'architecture matérielle, elle est programmée. Cependant, cette flexibilité a un coût mémoire, car le système doit sauvegarder des résultats de calculs intermédiaires, de plus, la quantification de ce coût est très complexe.

Le tableau 3.2 contient les informations de consommation énergétique et de temps de traitement d'une MAC pour les différentes plates-formes dédiées. Les métriques utilisées sont basées sur la rapidité de calcul d'une opération atomique et l'efficacité énergétique de ce calcul.

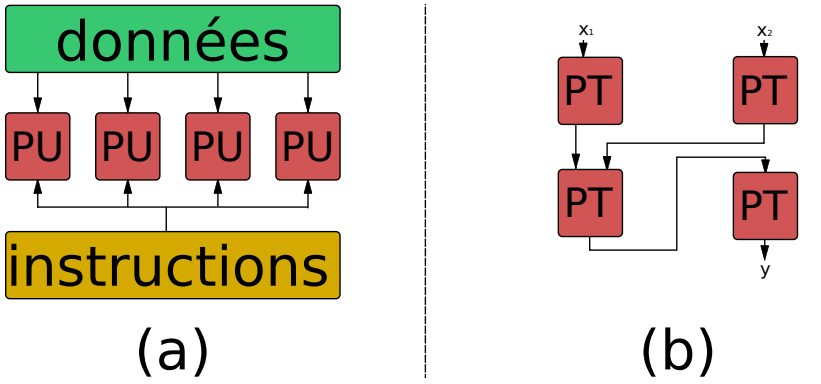

FIG. 3.1: Représentation des différentes approches architecturales du calcul des réseaux de neurones formel. (a) structure SIMD, (b) structure DataFlow.

3.2.3 Référence de la comparaison impulsionnelle

Par la suite, pour comparer les architectures formelles et les architectures impulsionnelles pour une application d'inférence CNN embarquée, il nous faut sélectionner les architectures formelles avec les meilleures performances pour ce type d'application. Nous allons donc utiliser la complexité calculatoire en opérations atomiques, que nous avons définie dans la section 2.3.3 pour l'étude du calcul de l'inférence CNN formelle embarquée. Nous avons utilisé ces définitions pour exprimer les performances des architectures formelles en matière OP/s et Op/J, voir la figure 3.2.

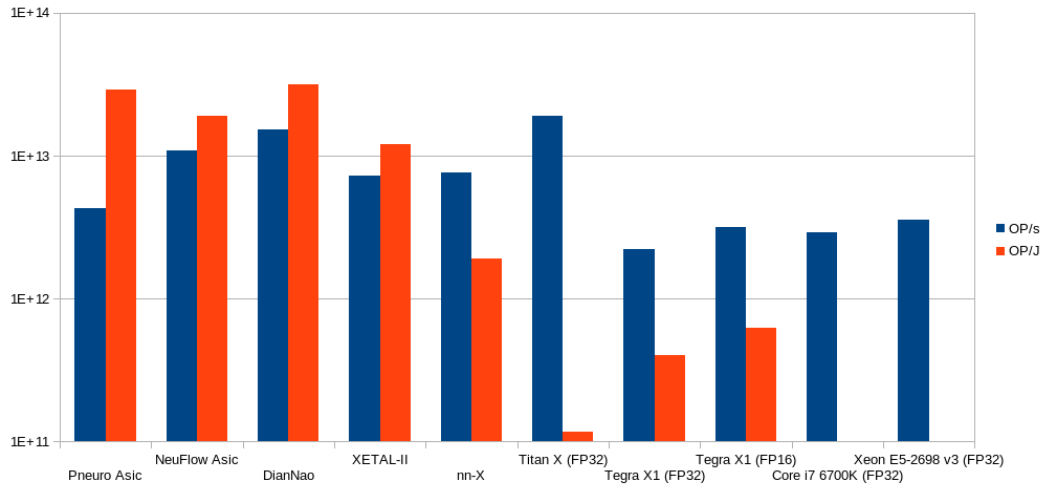

FIG. 3.2: Performance en nombre d'opérations atomiques par seconde OP/s et en nombre d'opérations atomiques par seconde et par watt Op/J des différentes architectures de calcul de réseau de neurones formel.

Au vu de la figure 3.2, on constate que l'efficacité énergétique en Op/J des architectures dédiées est au-dessus des $1TOp/J$ et que celle des architectures génériques est en dessous des $1TOp/J$. Les architectures dédiées présentent donc de meilleures performances énergétiques selon notre définition d'opérateurs atomiques que les architectures génériques.

Au niveau de la rapidité de calcul en OP/s la seule architecture générique véritablement comparable aux architectures dédiées est l'architecture Titan X. Cependant, l'efficacité énergétique de la Titan X est environ $10\times$ moins élevé que pour les architectures dédiées, ce qui finalement montre quelle n'est pas pertinente pour notre comparaison.

Les architectures dédiées sont donc plus propices à un environnement embarqué de calcul de l'inférence d'un CNN. Nous les utiliserons donc comme base de comparaison avec les architectures impulsionnelles et ainsi cette comparaison se fera en fonction des architectures formelles que nous avons identifiées comme les plus perfor-

mantes. Nous allons maintenant nous intéresser aux architectures impulsionnelles, pour exprimer leurs différentes caractéristiques et évaluer leurs performances.

3.3 Architectures impulsionnelles et adéquations avec les CNN

L'origine historique des architectures impulsionnelles est liée aux travaux de caractérisation des neurones biologiques. Les différents modèles et le fonctionnement général des neurones impulsionnels présentés dans la section 2.4.1. Il existe une grande variété d'approches du calcul impulsionnelle que nous allons présenter pour sélectionner l'approche qui nous semble la plus pertinente dans le cas du calcul d'inférence CNN embarqué.

La notion de Perceptron [39] a été introduite en 1958 à la suite de ces différents travaux de caractérisations. Cette notion a permis de proposer une étude de l'implémentation des neurones dans un réseau. Cependant, dans les années 70 plusieurs difficultés techniques, dont les ressources calculatoires disponibles, ont freiné les recherches dans le domaine. Les années 80 quant à elles ont vu un essor des structures de calculs parallèles, rendant le calcul neuromorphique possible. Le concept de calculateur neuromorphique fut introduit dans les années 1980 par Carver Mead [40, 41, 42, 43], avec une architecture neuroinspirée implémentée à très grande échelle (VLSI). Ainsi, plusieurs calculateurs neuromorphique ont vu le jour dans les années 80 à 90, et ont posé les bases architecturales des architectures neuroinspirées. On retrouve dans ces calculateurs le "Zero Instruction Set Computer" (ZISC036) d'IBM Micro electronics ou encore le Electrically Trainable Analog Neural Network (ETANN) d'Intel, la publication d'Heemskerk *et al* [1] propose un tour d'horizon de ces calculateurs.

Au vu de ces travaux, il est possible de partitionner les différentes architectures, en différentes approches possibles, voir figure 3.3. La branche dite des architectures standards est définie par l'utilisation de CPU en parallèle programmés pour le calcul neuromorphique. Dans les architectures standards, on retrouve les approches CPU, GPU et aussi SpiNNaker [44]. Ces types d'architectures sont caractérisées par l'utilisation de calculateurs à instructions disposant d'une mémoire et d'une partie calculatoire séparée (structure Von Neumann). Ces architectures sont en générales limitées pour le calcul neuromorphique dû au phénomène d'étranglement des accès mémoire, le goulot d'étranglement de Von Neumann [45]. Un des avantages des architectures standards est leur flexibilité. Cette flexibilité est due à la possibilité de les programmer. Il est ainsi possible d'expérimenter une grande variété de modèles et de topologies de réseaux sans changement du matériel. Comme nous l'avons dit, ces structures sont limitées par le goulot d'étranglement de Von Neumann entrainant un impact sur leur efficacité calculatoire. Or, notre approche est une approche

de maximisation de l'efficacité calculatoire et non une maximisation de la flexibilité d'implémentation. C'est pour cela que nous avons décidé d'écarter de notre étude les architectures standards impulsionnelles, pour nous focaliser sur les architectures neuromorphiques.

Cette problématique du goulot d'étranglement de Von Neumann a justifié la recherche sur les architectures neuromorphique. Le but de ce type d'architecture est de rapprocher voire même de combiner la mémoire et le calculateur pour supprimer cette problématique d'étranglement. Dans une architecture neuromorphiques, on retrouve des architectures spécialisées telles que Neurogrid [46], TrueNorth [47] ... Il existe une grande variété d'architectures neuromorphique, et chacune a été conçue pour répondre à une problématique calculatoire précise. Cette problématique peut être séparée en deux tendances distinctes. La tendance du biomimétisme et la tendance computationnelle. Notre approche fait partie des approches computationnelles, ces dernières utilisent des modèles neuromorphiques qui ne sont pas forcément biologiquement plausibles pour résoudre une problématique calculatoire. Ils forment donc une bonne base de comparaison. Dans le cas du biomimétisme, l'architecture est utilisée pour simuler le plus fidèlement possible des phénomènes biologiques par l'utilisation de modèles neromorphiques plus complexes.La structure calculatoire de ce type d'architecture reste valide pour notre étude, mais la complexité des modèles utilisés risque de minimiser leur performance calculatoire par rapport aux approches computationnelles.

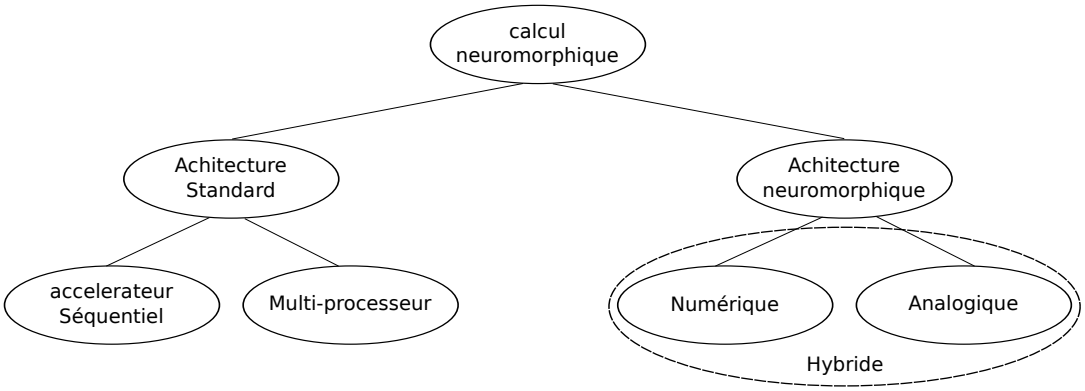

FIG. 3.3: Les différentes approches du calcul neuromorphique [1].

Pour notre étude nous avons sélectionné 6 architectures impulsionnelles capables de simuler au moins une opération de convolution. Nous avons partitionné notre étude en deux catégories d'architectures : les architectures à topologie générique, capable d'instancier un réseau quelconque et les architectures à topologie de réseaux spécialisés dans l'opération de convolution. Nous allons étudier leurs différentes approches et caractéristiques pour en déduire leur performance en OP/s et Op/J et leur capacité d'instanciation d'un CNN. Nous conclurons par une comparaison de

ces différentes architectures pour pouvoir déterminer la meilleure approche du calcul de l'inférence d'un CNN impulsionnelle en embarqué.

3.3.1 Étude des architectures à topologie générique

Une architecture à topologie de réseaux génériques peut simuler une ou plusieurs couches FC. Un neurone d'une couche de FC est connecté à toutes les entrées de la couche. Pour changer de topologie, il suffit de couper certaines connexions des neurones d'une couche FC.

Ainsi, ces architectures sont capables de simuler n'importe quel type de connexion entre deux couches. La limitation en matière de topologie vient du nombre maximal de synapses par neurone et du nombre total de neurones. Dans le cas d'une couche de convolution, l'architecture doit pouvoir fournir autant de neurones que d'éléments sur le tenseur de sortie.Chaque neurone doit disposer d'autant de synapses que de pondérations dans les filtres. Nous avons sélectionné 2 architectures à topologie de réseaux génériques. L'architecture IBM TrueNorth [47, 48] du projet américain Synapse et l'architecture BrainScaleS [31, 32, 49] du projet européen "Human Brain Project".

Le but de l'architecture TrueNorth d'IBM est de fournir une plate-forme générique pour des applications computationnelles. Cette architecture regroupe 4096 cœurs sur une seule puce CMOS. Chaque cœur contient 256 "leaky integrate and fire" (LIF) neurones numériques et chaque neurone est connecté à 256 synapses. Tous les cœurs sont interconnectés par un réseau sur puce (NOC). Il est possible d'interconnecter plusieurs cœurs entre eux pour former un grand réseau. Dans cette architecture le temps est discrétisé et les cœurs sont synchronisés par un signal global de $1kHz$. La consommation moyenne est estimée à environs $68mW$.

L'architecture BrainScaleS quant à elle, est consacrée à l'accélération de simulation neuromorphique. Cette architecture propose un circuit mixte à neurones analogiques. Elle se compose de 352 cœurs tous connectés par une grille 2D. Elle est conçue pour se déployer sur un wafer complet de 20 cm de diamètre. Chaque cœur peut au maximum instancier 512 neurones et contenir en plus un circuit d'apprentissage non supervisé (STDP). À la différence de True North, il est possible de modifier le nombre de synapses connecté à un neurone. La consommation estimée de l'architecture sur un wafer est de $1kw$.

On peut identifier une approche générale pour ces architectures à topologie de réseaux génériques. Ces architectures peuvent être vues comme des calculateurs multi-cœurs spécialisés dans le calcul d'une couche FC. Ces cœurs sont reliés par un système d'interconnexion configurable. Pour instancier un réseau, celui-ci doit être découpé en sous-réseaux instanciables dans un cœur. Au final le système d'interconnexion est utilisé pour relier les cœurs/sous-réseaux pour former le réseau final. Ainsi ces architectures sont composées de deux structures distinctes : les cœurs et

le système d'interconnexion.

Nous allons analyser la mise en place d'un CNN dans ces architectures. L'étude se focalisera sur les cœurs et leur capacité et performance dans le calcul d'un ou plusieurs neurones d'une couche de convolution. Nous n'étudierons pas les systèmes d'interconnexions, nous considérerons qu'ils permettent l'association d'un nombre suffisant de cœurs pour former un CNN.

Le neuro-cœur de IBM TrueNorth

La structure générale du neuro-cœur d'IBM TrueNorth consiste en K entrées connectées à N neurones par une grille de $N \times K$ synapses. Les neuro-cœurs contiennent 256 neurones numériques avec 64k synapses soit 256 entrées par neurones. Pour un neurone i connecté à une entrée j la valeur de la pondération est donnée par $W_{ij} \times S_i^{G_j}$. Avec $W_{ij} \in \{0, 1\}$ pour le codage de la connexion et $S_i^{G_j}$ pour la valeur de la pondération. Sachant que $G_j \in \{0, 1, 2\}$, les pondérations d'un même neurone ne peuvent prendre que 4 valeurs différentes. Au niveau des neurones, les pondérations peuvent être utilisées comme une valeur déterministe ou stochastique.

Dans le cas du calcul d'une couche de convolution, un cœur peut implémenter un filtre maximal de 256 pondérations et au maximum une matrice de sortie de 256 éléments. Tant que les maximaux ne sont pas atteints, la taille et les dimensions du filtre peuvent être quelconques. Il en est de même pour le nombre d'éléments de l'image de sortie. Cependant, ces cœurs ne sont pas optimaux pour le calcul de convolution. Ainsi, pour deux neurones de la même couche, les pondérations sont identiques. Elles doivent cependant être répétées dans le cœur pour chaque neurone. De plus, si un neurone implémente un filtre avec moins de 256 pondérations, les pondérations non utilisées par ce neurone ne sont pas réutilisables. Pour finir, pour un même neurone seul 4 valeurs différentes de pondération sont possibles dans un filtre, ce qui pour certaines applications CNN est insuffisant.

Nous allons maintenant nous intéresser aux propriétés dynamiques des neuro-cœurs. Le traitement des impulsions par un cœur se déroule en 6 étapes :

1. Le cœur reçoit des impulsions du réseau et les sauvegardes dans un buffer.
2. Le cœur reçoit un signal de synchronisation à 1 kHz, et distribue les impulsions sur les lignes horizontales des axones correspondant sur la grille synaptique.
3. S'il existe une connexion entre l'entrée et le neurone, l'impulsion est transmise au neurone
4. Les neurones qui ont reçu une impulsion mettent à jour leur potentiel de membrane.
5. Quand toutes les impulsions sont traitées, le cœur applique la fuite sur le potentiel de membrane de tous les neurones.

6. Si le potentiel de membrane d'un neurone est supérieur au seuil, une impulsion est envoyée dans le réseau.

Tous les cœurs sont synchronisés par le signal de 1 kHz. Cette synchronisation impose donc un temps de traitement de 1 ms, quelle que soit la complexité de la couche implémentée par un cœur. Malgré cette fréquence de fonctionnement basse, la parallélisation de la lecture des pondérations et du calcul permet à un cœur de calculer 256k impulsions/s.

Pour conclure, cette architecture est compatible avec le calcul de l'inférence d'un CNN impulsionnel. Le fonctionnement parallèle de ses cœurs lui permet d'avoir de bonnes performances en temps de traitement malgré une fréquence basse de fonctionnement. Cependant, la faible variété de valeur de pondération possible dans un filtre, la répétition des pondérations et le nombre de synapses sont fixes par neurone, rendent cette architecture incompatible avec les CNN.

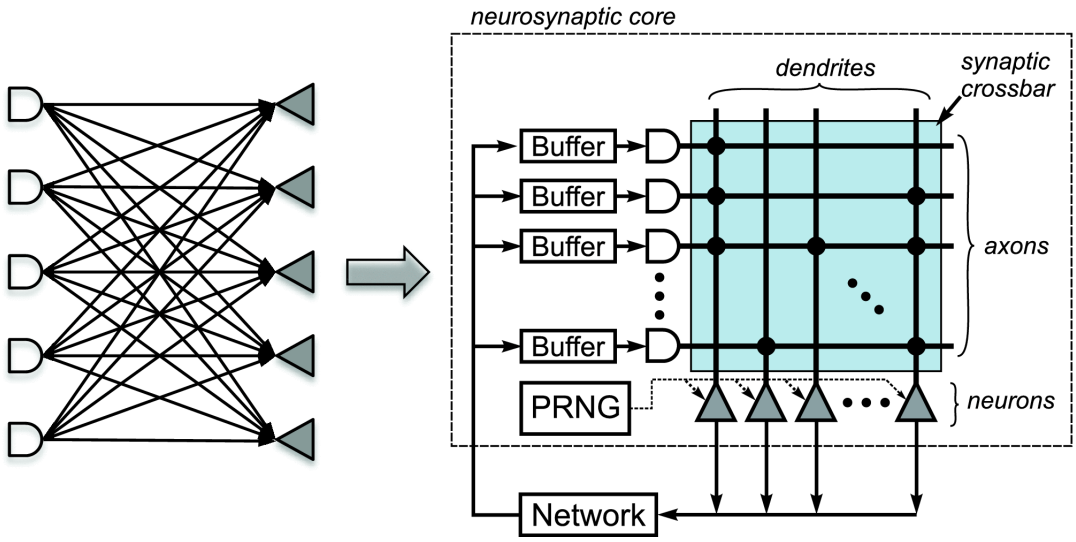

FIG. 3.4: Représentation du placement d'un réseau FC sur un cœur TrueNorth. On identifie les entrées du système et la sortie connectée au réseau de communication. Dans la puce les entrées et sorties sont interconnectées par une grille synaptique contenant les pondérations.

Les cœurs de BrainScaleS, les HICANN

Dans l'architecture BrainScaleS les cœurs sont notés HICANN pour High Input Count Analog Neural Network. Comme pour TrueNorth, on retrouve des structures en grilles de $N \times K$ synapses. Dans un cœur, il y a deux grilles synaptiques de 224×256, interconnectées par un circuit de simulation de neurone. Cependant, à la différence de TrueNorth les neurones peuvent avoir un nombre de synapses variable. Chaque synapse est connectée à plusieurs entrées. Pour permettre une variabilité du nombre de synapses par neurone, les neurones sont découpés en sous-circuit les

DenMem (dendrite membrane). Chacun des DenMem est connecté aux 224 synapses. Les neurones sont formés d'un nombre arbitraire de DenMem.

Chaque synapse de la grille est adressable sur 6 bits. Chaque synapse est alors connectée à l'équivalant de $2^6 = 64$ entrées. Dans l'architecture, deux colonnes de synapses adjacentes partagent les mêmes 64 entrées pré-synaptiques, ce qui implique que dans une ligne de synapse il y a 112 synapses connectées aux mêmes entrées. Ainsi un neurone qui utilise les deux grilles synaptiques peut être connecté au maximum à $\frac{224}{2} \times 64 \times 2 = 14336$ entrées, voir la figure 3.6 .

Au niveau des pondérations, chaque synapse contient une SRAM de 4 bits, cette valeur est convertie en un courant par un convertisseur numérique/analogique à gain variable. Il est possible de fixer un gain pour chaque colonne de synapse, ainsi on peut utiliser deux colonnes adjacentes avec des gains multiples pour obtenir des pondérations avec une précision de 6 à 8 bits. Les DenMem ont deux entrées A ou B. Les synapses d'une ligne sont connectées au DenMem par l'une de ces deux entrées. Les pondérations d'une même colonne sont connectées à un DenMem par l'une ou l'autre de ces entrées, ce système permettant d'avoir des pondérations positives ou négatives.

Dans le cas du calcul d'une couche de convolution. Si on considère que deux colonnes synaptiques adjacentes partagent le même adressage pré-synaptique il y a $\frac{224}{2} = 112$ synapses adressables par DenMem. Un DenMem peut alors implémenter un filtre de 112 pondérations, soit en double précision, soit signées. Sachant que les synapses sont connectées à 64 entrées, on obtient 64 DenMem ne partageant pas les mêmes connexions par grilles synaptiques. Pour une grille synaptique, il y a 256 DenMem. Il y a alors $\frac{256}{64} = 4$ groupes de 64 DenMem connectés aux mêmes entrées. Ainsi en utilisant une grille synaptique et un DenMem par neurone, il est possible de calculer 4 matrices de sortie de 64 éléments connectés à la même matrice d'entrée par un filtre maximal de 112 pondérations. Le cœur est connecté à deux grilles synaptiques, il est alors possible de faire 8 matrices de sortie ou d'avoir deux matrices d'entrées connectées à 4 matrices de sortie sans recouvrement. Pour finir, plusieurs DenMem peuvent être utilisés, soit pour l'implémentation de plus grands filtres et/ou pour avoir des pondérations en doubles précisions et signées.

Conclusion des architectures à topologie de réseaux générique

Nous avons vu que les différentes architectures présentées se basent sur des structures entièrement connectées pour garantir leur généricité. Dans chaque cœur, l'accès parallèle aux pondérations permet une utilisation efficace du matériel lors de la réception d'impulsion(s). Les cœurs de ces architectures sont compatibles avec le calcul de couche de convolution impulsionnelle, mais elles ne sont pas compatibles avec les couches de MaxPooling. Par extension, ces architectures peuvent implémenter un réseau CNN sans MaxPooling par interconnexion de plusieurs cœurs à travers

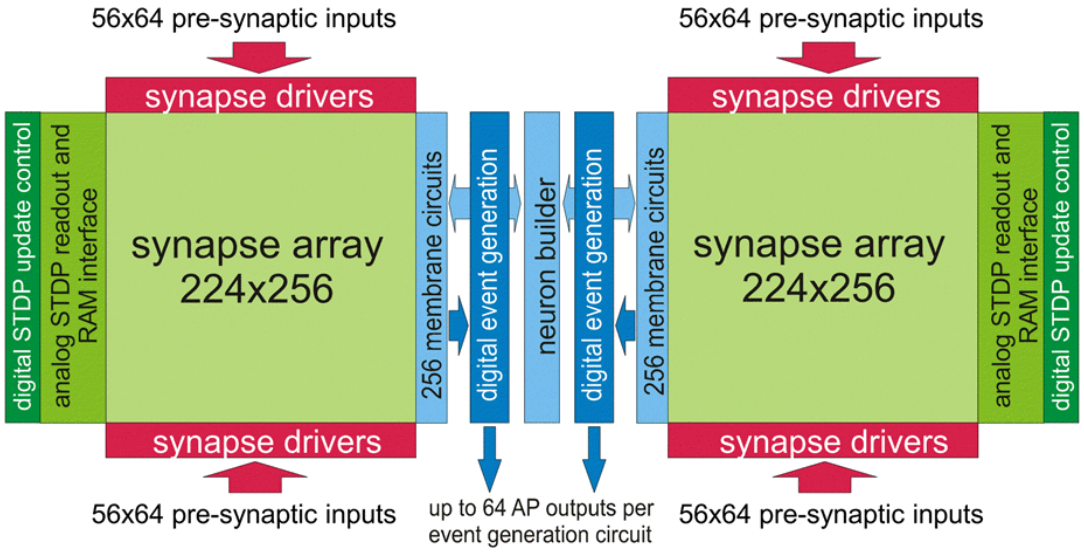

FIG. 3.5: Les cœurs de BrainScaleS, les HICANN qui sont composés de deux grilles synaptiques (en vert) et d'un circuit de simulation de membrane et de communication (en bleu). Chaque cœur est connecté à $4 \times 56 \times 64$ entrées pré-synaptic (en rouge).

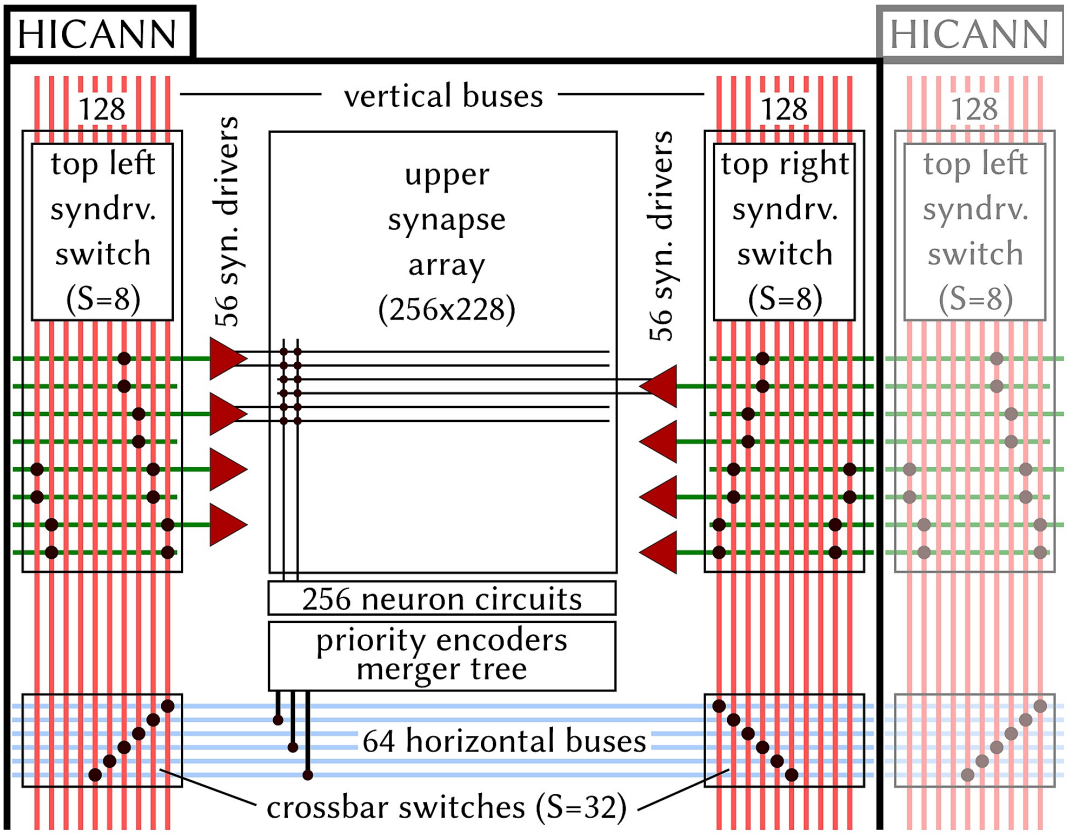

FIG. 3.6: Zoom sur les connexions entre les entrées pré-synaptique (ligne rouge verticale) et les synapses et entre les sorties des neurones le bus horizontal (bleu) d'une grille synaptique d'un HICANN.

	IBM TrueNorth [47] [48]	BrainScaleS [31, 32, 49]
Technologie	45 nm SOI	180 nm CMOS
Surface	6 $3 \times 2mm^2$	$5 \times 10mm^2$
Nombres de neurones	256	8×64
Tailles du filtre max	256	112
Nombre de filtres max	256	8
Type de calcul	numérique	analogique
Impulsions en parallèle max	256	8
Temps de traitement	1 ms	100 ms (10Hz)
Précision	1 bit	4 bits à 8 bits
Consommation	$\approx 10^{-2}mW$	$\approx 1.7W$

TAB. 3.3: Comparatif des cœurs des architectures à topologie de réseaux génériques pour l'implémentation de couche de convolution.

leur réseau d'interconnexion. Le résumé des caractéristiques de ces architectures est présenté dans le tableau 3.3.

Il est donc possible de calculer une couche de convolution avec ces architectures, cependant elles n'ont pas de système de partage de pondérations entre plusieurs neurones. Ainsi pour implémenter une couche de convolution, il faut répéter pour chacun des neurones les pondérations du filtre. Dans le cas d'un CNN le coût de la répétition devient un facteur limitant, engendrant un surcoût matériel non négligeable. Par exemple pour une couche de sortie de 10×10 neurones et un filtre de 3×3 pondérations la répétition du filtre représente un surcoût de 899 pondérations à stocker en plus. Il existe de nombreuses architectures [50, 51] dont le cœur de calcul est comparable aux deux architectures traitées et donc souffrant des mêmes lacunes pour l'opération de convolution.

Nous allons maintenant nous intéresser aux architectures optimisées pour l'opération de convolution impulsionnelle et à leur système de partage des pondérations entre neurones.

3.3.2 Les architectures à topologie de réseaux spécialisés dans l'opération de convolution

Dans cette partie, nous allons nous intéresser aux architectures dédiées aux couches convolutionnelles impulsionnelles. Nous allons présenter l'évolution de ce type d'architectures pour analyser les différentes approches et sélectionner les architectures qui nous semblent les plus pertinentes dans notre cas.

Les travaux de Venier *et al* [52] sont inspirés du principe de cellules simples [53].

Une cellule simple est connectée à plusieurs cellules de la rétine, pour former un groupe sensible à la direction de la lumière. Cette architecture analogique permet d'avoir un filtre maximal de 10×10 pour 27×27 neurones de sorties. Cette approche n'est pas pertinente dans notre cas, car les pondérations des filtres ne peuvent pas être fixées arbitrairement. Il faut noter cependant que cette approche permet d'obtenir une consommation $< 9mW$.

Dans travaux de Choi *et al* [2] est présentée une architecture analogique faible consommation, environ $6mW_{max}$. Cette architecture est inspirée du cortex visuel primaire des mammifères [54, 55] le "ice-cube" modèle. L'ice-cube" modèle représente le cortex visuel (V1) comme un empilement de couche de neurones avec des connections 2D. Le nombre de couches empilées est quelconque et les neurones d'un même plan sont sensibles à la même direction. Cette architecture est divisée en plusieurs puces, chaque puce applique sur les impulsions d'entrées un filtre de type gabor. Les filtres/puces sont chaînés pour reproduire le "ice-cube" modèle, voir figure 3.7.

Ces architectures sont des approches intéressantes de la problématique de traitement de matrices par des réseaux de neurones impulsionnels. Cependant, elles ne sont pas compatibles avec le calcul à base de filtre arbitraire. Ce partie prise leur permet d'instancier les pondérations avec un circuit analogique réglable. Cette approche à un effet notable sur la complexité de l'architecture, car il n'y a pas de mémoire de pondérations. Et cette réduction de la complexité et l'utilisation d'un circuit analogique faible consommation leur permettent d'atteindre des consommations $< 10mW$. Cependant, les CNN nécessitent l'implémentation de filtres arbitraires issues d'un apprentissage par optimisation du gradient.

Nous concentrerons donc notre étude sur les architectures à filtre(s) arbitraire(s). Nous avons sélectionné les différentes approches architecturales portées par les équipes de Linares-barranco. Dans ces architectures, les pondérations sont stockées en mémoire et partagées entre tous les neurones. On retrouve dans ces architectures une transmission séquentielle des impulsions du type "Address-Event Representation" (AER) [56]. Le traitement séquentiel des impulsions n'est pas anodin, car il permet d'éviter la problématique de l'accès concurrent aux pondérations par plusieurs neurones.

J'ai identifié deux approches dans l'instanciation des neurones dans les architectures à topologie de réseaux spécialisés : l'approche à neurones unitaire et l'approche à neurones multiplexés. Dans l'approche à neurones unitaires, tous les neurones sont physiquement implémentés et indépendants. Tandis que dans l'approche à neurones multiplexés, les neurones sont découpés en plusieurs fonctions dont certaines sont communes à plusieurs neurones. Nous allons nous intéresser à ces deux approches pour identifier les possibles gains dans le calcul de l'inférence dans les CNN impulsionnels.

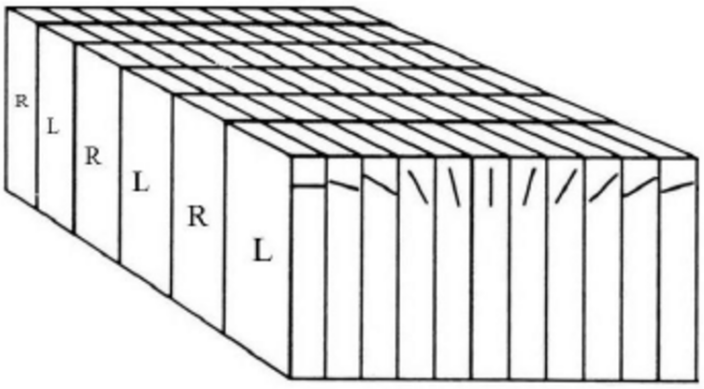

FIG. 3.7: le "ice-cube" modèle, logique d'assemblage en couches [2].

Architecture de convolution à filtre(s) arbitraire à neurones unitaire

On peut distinguer dans l'état de l'art 3 architectures proposant une méthode de calcul d'une opération de convolution impulsionnelle à neurones unitaires. Ces 3 architectures sont très similaires et peuvent être considérées temporellement comme des évolutions d'une même architecture. Avec une première architecture à neurones analogique [57] puis une version numérique [3] et enfin une architecture numérique multi-filtres [58].

Dans ce type d'architecture, les neurones sont implémentés séparément. Chaque neurone à sa propre intégration et son propre circuit de membrane. Tous les neurones sont répartis sur une grille qui représente la matrice de sortie de la couche de convolution, chaque élément de cette matrice est un axone d'un neurone. Les pondérations sont stockées dans une RAM statique et sont distribuées aux neurones en fonction de l'impulsion reçue. Tous les adressages sont en 2D, ainsi, les impulsions sont identifiées par leurs coordonnées (x_{in}, y_{in}) et les neurones par leurs coordonnés (x, y).

Nous allons baser notre étude sur l'architecture Camunas 2011, cette étude est valable globalement pour les deux autres architectures en raison de leur fonctionnement similaire. L'architecture est décomposable en 4 blocs : une entrée communication/adressage, une communication de sortie, une mémoire des pondérations avec un contrôleur et une grille de neurones, voir la figure 3.8. La mémoire des pondérations est commune à tous les neurones, évitant ainsi la redondance des pondérations. La grille de 32×32 neurones est composée de neurones numériques et adressables en fonction de leurs coordonnées (x, y). Le bloc d'entrée communication/adressage détermine en fonction de l'impulsion d'entrée, quelle pondération est envoyée à quels neurones. Le bloc de communication de sortie reçoit les activations des neurones, les sérialisent et les envoie en sortie. Les communications sont assurées par un bus AER et les impulsions sont identifiées par deux coordonnées (x_{in}, y_{in}) et un bit de signe

s_{in}. Du point de vue de la couche de convolution, l'architecture peut implémenter différentes tailles de filtre pour différentes tailles de matrice de sortie.

Lors de l'arrivée d'une impulsion, le bloc d'entrée communication/adressage va vérifier que cette impulsion est connectée à la matrice de sortie et donc aux neurones. Si c'est le cas, les pondérations du filtre sont lues ligne par ligne et signées en fonction du s_{in} puis envoyées sur la colonne de la grille de neurones concernés par l'impulsion. Parallèlement, la ligne de la grille de neurones concernés par l'impulsion est activée par le bloc d'entrée communication/adressage, activent ainsi l'intégration de la pondération par le neurone concerné. Ces opérations sont effectuées jusqu'à la lecture complète du filtre. Le bloc de communication de sortie peut à un instant t recevoir plusieurs activations de neurones d'une même ligne. Ce bloc va alors sérialiser les impulsions de sortie et les envoyer en sortie de l'architecture.

On remarque que le nombre de neurones est fixé matériellement et que pour traiter des matrices plus grandes ou un tenseur 3D, il faut utiliser plusieurs instances de cette architecture. De plus, lors de la réception d'une impulsion, seul un nombre limité de neurones est sollicité, mais tous les neurones sont physiquement implémentés, il y a donc une redondance matérielle. Ceci implique qu'une grande partie du matériel peut être regroupée et partagée entre plusieurs neurones sans perte de performance. Cette logique est celle de l'approche par neurones multiplexés.

Architecture de convolution à filtre(s) arbitraire à neurones multiplexé

On retrouve cette approche de neurones multiplexés dans l'architecture FPGA S-ConvNet présentée par A.Yousefzadeh *et al* [4]. Dans cette architecture, les neurones se partagent certaines fonctions. Les neurones sont ainsi séparés en une partie mémoire pour l'intégration et une partie calcul pour le modèle de membrane et d'axone. Ainsi, il est possible de réduire la redondance matérielle en partageant des fonctions de calculs entre plusieurs neurones. Ce partage n'implique pas forcément une perte de performance, car dans une couche de convolution soumise à une impulsion, seul un nombre limité de neurones sont sollicités simultanément.

L'architecture présentée peut implémenter l'équivalent d'une grille de 128×128 neurones. Le traitement des impulsions est pipeline et l'architecture calcule parallèlement la réponse à une impulsion d'une ligne de 128 neurones. Dans cette architecture, les neurones partagent la partie calculatoire de la membrane et de l'axone. Elle impose ainsi la séparation entre la valeur interne des neurones (l'intégration) et la partie calculatoire. Ainsi l'architecture peut être décomposée en 4 blocs : les pondérations, les intégrations, l'adressage et le calcul, voir la figure 3.9.

Lors de l'arrivée d'une impulsion, le bloc d'adressage sélectionne une ligne de 128 intégrations et une ligne du filtre. La ligne du filtre est placée dans un vecteur de 128 éléments en fonction de l'adresse d'entrée de l'impulsion par le "kernel size matching". Les autres éléments du vecteur de 128 éléments contiennent des pondérations mises

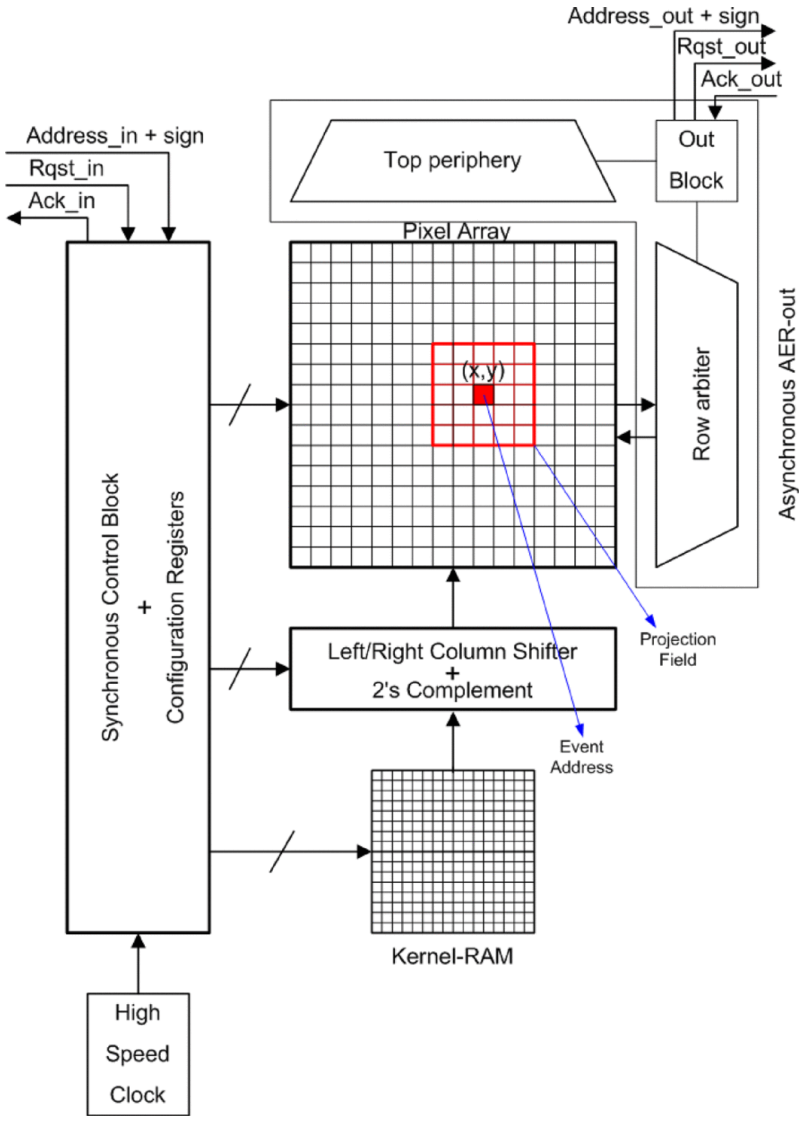

FIG. 3.8: Architecture spécialisée dans l'opération de convolution impulsionnelle [3]. Les pondérations stockées dans une RAM sont distribuées séquentiellement aux neurones contenus dans le "Pixel Array" à chaque réception d'une impulsion.

à zéro. Les deux vecteurs pondération et intégration de 128 éléments sont alors transmis au bloc de calcul. Le bloc de calcul va parallèlement déterminer la nouvelle valeur d'intégration et la sortie des 128 neurones de la ligne. Les 128 nouvelles valeurs d'intégration sont sauvegardées par le bloc d'intégration dont le vecteur de sortie est traité par un bloc de communication (non représenté). Ces opérations sont répétées et pipelines, jusqu'au traitement de toutes les lignes. Le temps de traitement dépend du nombre de lignes du filtre L et correspond à $L+3$ clk. On note une variation du temps de traitement en fonction de la plate-forme, ainsi pour un filtre de 24×24 il faut environ 312 ns sur un Spartan-6 et 156 ns sur Virtex-7 de temps de traitement.

Dans cette architecture, on retrouve la logique de calcul ligne par ligne des architectures présentées dans la section 3.3.2. Cependant, la partie calculatoire n'est

instanciée que pour une seule ligne de 128 neurones. Ce qui permet de passer de $N \times N$ calculateurs à N calculateurs sans perte de performance. L'approche en neurones multiplexés par rapport à l'approche unitaire est donc matériellement moins onéreuse à performance maintenue. Cependant, le bloc de calcul traite un vecteur de pondération majoritairement nul pour de petits filtres, ce qui risque de nuire à l'efficacité globale.

Conclusion : Les architectures à topologie de réseaux spécialisés dans l'opération de convolution impulsionnelle

Nous avons vu que les architectures convolutionnelles permettent de résoudre la problématique de la redondance des pondérations du filtre par leur partage entre tous les neurones. Cette approche permet de réduire le matériel nécessaire au stockage des pondérations en introduisant une problématique d'accès concurrent à la mémoire. Cette problématique d'accès concurrent est résolue par le traitement unitaire et séquentiel des impulsions en entrée.

Nous avons étudié deux approches d'instanciation de neurones : en neurones unitaires 3.3.2 et en neurones multiplexés 3.3.2. Dans le cas de l'approche à neurones unitaire, nous avons vu que chaque neurone était instancié physiquement, mais peu sollicité. Cette approche par neurones unitaires est donc sous-optimale dans son utilisation matérielle. Dans le cas de l'approche à neurone multiplexé, un certain nombre de neurones partagent le même calculateur. Ce partage permet de réduire la redondance matérielle, mais le calcul ligne par ligne nécessite d'implémenter de grands blocs de calculs majoritairement non utilisés. Les différentes performances calculatoires des architectures sont résumées dans le tableau .

3.3.3 Conclusion des architectures impulsionnelles

En regroupant les informations des différentes architectures impulsionnelles génériques et spécialisées, nous sommes en mesure d'établir une comparaison. Nous utilisons dans cette comparaison la vitesse de calcul d'une opération OP/s et l'efficacité énergétique du calcul Op/J pour le calcul d'une convolution. La figure 3.10 représente les différentes caractéristiques pour chaque architecture.

On remarque que les architectures impulsionnelles spécialisées présentent de meilleures performances $> 10GOP/s$ et $> 100GOP/J$ que les architectures génériques $< 0.1GOP/s$ et $< 10GOP/J$, selon nos métriques. Ceci s'explique par le fait que les architectures génériques sont désavantagées par la complexité d'un adressage générique et par une incapacité à partager les pondérations entre plusieurs neurones.

Les architectures spécialisées représentent donc une bonne base de travail pour l'implémentation CNN impulsionnelle. Cependant, elles ne peuvent pas dans l'état implémenter un CNN complet car elles ne sont pas capables de calculer des couches

	Serrano[57]	Camunas 2011[3]
Technologie	0,35µm	0,35µm
Surface	$4.0 \times 5.0mm^2$	$4.3 \times 5.4mm^2$
Nombres de neurones	32×32	32×32
Tailles du filtre max	32×32	32×32
Nombre de filtres max	1	1
Type de calcul	analogique	numérique
Impulsions en parallèle	1	1
Temps de traitement	30-340 ns	50-565 ns
Précision	3 bits	18 bits
Consommation	$150mW_{max}$	$200mW_{max}$
	Camunas 2012[58]	Multiplexé [4]
Technologie	0,35µm	Virtex-7
Surface	$5.5 \times 5.8mm^2$	X
Nombres de neurones	64×64	128×128
Tailles du filtre max	32×32	128×128
Nombre de filtres max	32	1
Type de calcul	numérique	numérique
Impulsions en parallèle	1	1
Temps de traitement	60-680ns	24-774ns
Précision	6 bits	6 bits
Consommation	$200mW_{max}$	X

TAB. 3.4: Comparatif des architectures de convolution à filtre(s) arbitraire.

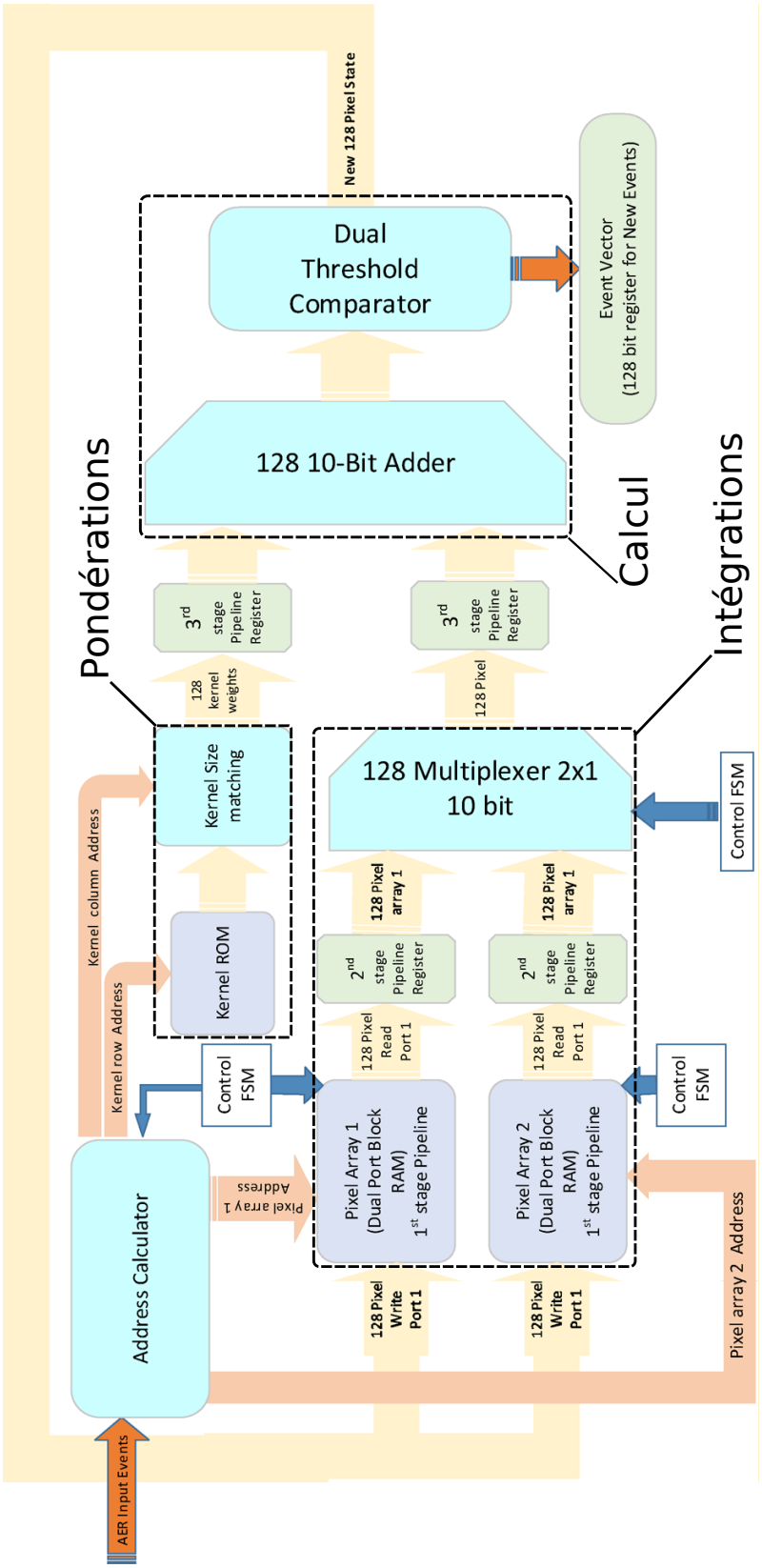

FIG. 3.9: Diagramme de traitement de l'architecture à neurones multiplexés [4]. Dans cette architecture, les calculs sont effectués ligne par ligne, les neurones de toutes les lignes partagent donc les mêmes blocs calculatoires.

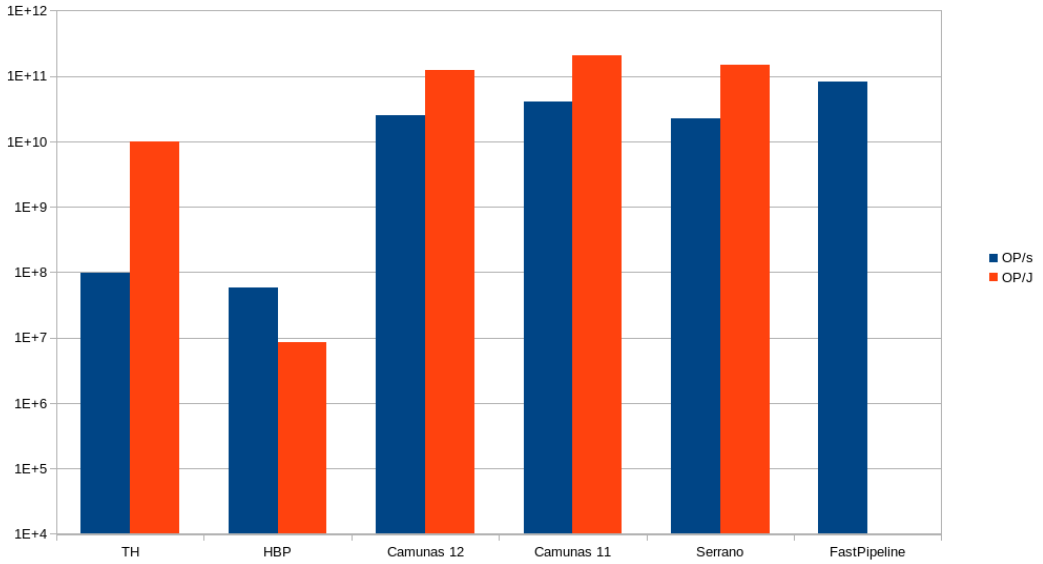

FIG. 3.10: Performance en nombre d'opérations atomiques par seconde OP/s et en nombre d'opérations atomiques par Joule Op/J des différentes architectures de calcul de réseau de neurones impulsionnels pour une application de convolution.

de MaxPooling ou d'implémenter un tenseur 3D par couche ou plusieurs couches. Nous allons maintenant comparer les architectures impulsionnelles aux architectures formelles de l'état de l'art pour voir si les architectures impulsionnelles rivalisent avec les architectures formelles.

3.4 Comparatif des architectures de l'état de l'art

L'analyse des différentes architectures développée section 3.2 et section 3.3 vont nous permettre de mettre en place notre comparatif entre les architectures formelles et les architectures impulsionnelles. Pour cela, nous utilisons notre méthode de comparaison du temps de traitement et l'énergie consommée définie dans la section 2.5.2. La figure 3.11 représente le nombre d'impulsions pré-synaptiques par entrées pour qu'une architecture impulsionnelle donnée soit aussi rapide d'une architecture formelle donnée. La figure 3.12 représente le nombre d'impulsions pré-synaptiques par entrées pour qu'une architecture impulsionnelle donnée consomme la même énergie d'une architecture formelle donnée.

On constate que l'on retrouve les deux résultats sur les performances des architectures formelles et impulsionnelles précédentes. Ainsi, si l'on compare entre les architectures impulsionnelles on retrouve les mêmes écarts que dans la figure 3.10. Si l'on regarde les architectures formelles, on constate que plus elles sont performantes, moins il y a d'impulsions pré-synaptiques par entrées, voir la figure 3.2, ce qui reste

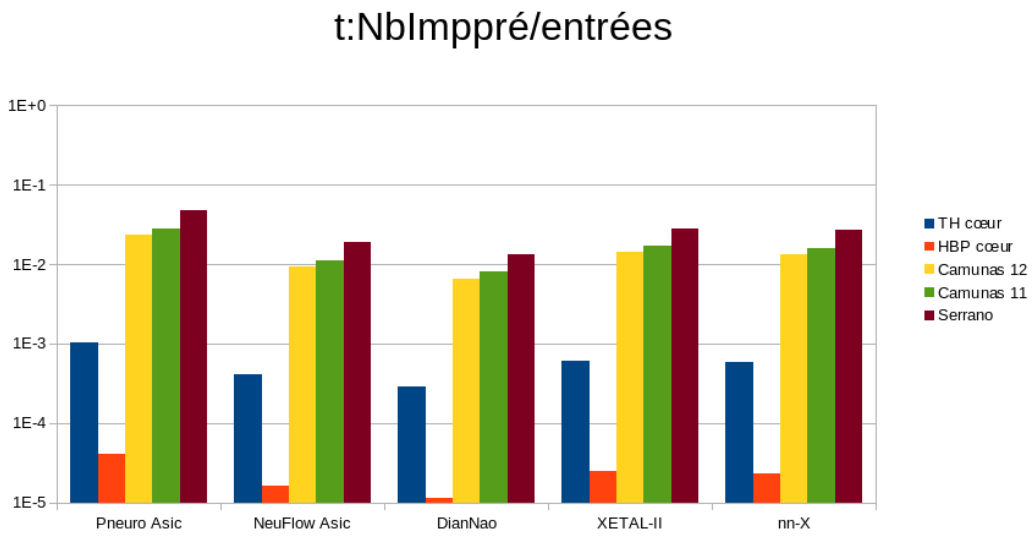

FIG. 3.11: Le nombre d'impulsions pré-synaptiques par entrées pour une équivalence en matière de temps de traitement. Le nombre d'impulsions pré-synaptiques par entrées est exprimé pour chaque architecture formelle et pour chaque architecture impulsionnelle.

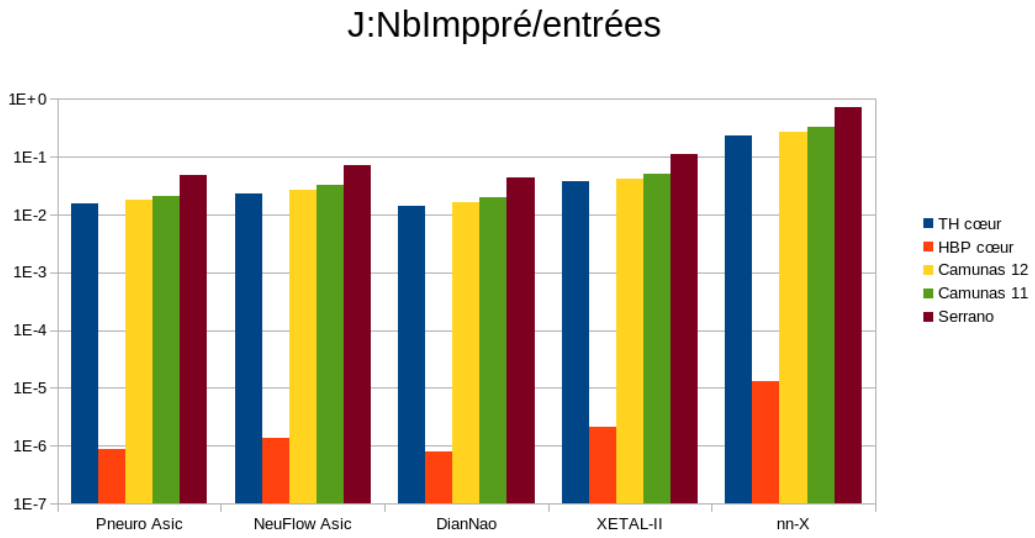

FIG. 3.12: Le nombre d'impulsions pré-synaptiques par entrées pour une équivalence en matière d'énergie. Le nombre d'impulsions pré-synaptique par entrées est exprimé pour chaque architecture formelle.

cohérent les définitions de nos métriques.

On constate qu'en matière de vitesse de traitement les architectures impulsionnelles les plus performantes sont limitées à environ 0,01 impulsion pré-synaptique par entrées. En matière d'énergie les architectures impulsionnelles les plus performantes sont limitées à environ 0,1 impulsion pré-synaptique par entrées. Or, la transposition

du formel vers l'impulsionnel nous impose un codage fréquentiel de l'information, ce qui implique en général plusieurs impulsions par entrées.

Pour conclure, notre comparatif montre que les architectures impulsionnelles de l'état de l'art ne présentent pas des performances suffisantes pour être assurées de rivaliser avec les architectures formelles de l'état de l'art. À mon sens, ce résultat est dû à plusieurs facteurs : Premièrement, le codage impulsionnel n'a pas profité du portage sur GPU, il reste donc moins répandu pour les problématiques de calcul que le codage formel et cela impacte les architectures impulsionnelles qui sont potentiellement moins abouties que leur équivalent formel. Deuxièmement, le surcoût mémoire du modèle impulsionnel engendre potentiellement une surconsommation énergétique qui annule les potentiels gains obtenus par la simplification des opérateurs. Troisièmement, le manque d'optimisation des architectures impulsionnelles spécialisées, qui pourraient cumuler plus de matérielles entre les neurones.

3.5 Analyse, Conclusion et positionnement

En s'intéressant aux fonctions implémentables dans les architectures impulsionnelles, on constate qu'il n'est pas possible de calculer une couche de MaxPooling, plusieurs couches ou un tenseur 3D de sortie. Ainsi, il n'existe pas d'architecture impulsionnelle réellement compatible avec le calcul d'un CNN complet.

Du point de vue de l'efficacité les architectures impulsionnelles de l'état de l'art sont très potentiellement moins efficaces que leur équivalent formel. Cependant, le traitement unitaires des impulsions simplifie les problématiques de lecture concurrentielle des pondérations et le multiplexage des neurones est une logique d'optimisation intéressante. À mon sens des optimisations sont encore possibles pour le traitement des CNN impulsionnels tel que : le multiplexage complet de tous les neurones, la lecture parallèle des pondérations du filtre, le regroupement/minimisation des fonctions des adressages des intégrations et des pondérations.

Au vu de ces résultats, j'ai donc choisi de m'orienter vers une architecture suffisamment flexible pour l'implémentation des CNN impulsionnels avec un traitement unitaire des impulsions et à neurone fortement multiplexé. Cette stratégie nous permettra à mon sens de proposer une architecture impulsionnelle plus performante et paramétrable pour les CNN que celles de l'état de l'art.

Dans le prochain chapitre, nous présenterons notre chaîne générique de validation des approches impulsionnelles à traitement unitaire. Ce qui nous permettra par la suite de présenter notre architecture, notre approche du calcul de l'inférence des CNN impulsionnels et les résultats obtenus.

Chapitre 4

Modélisation

Contents

4.1 Introduction

Dans ce chapitre, nous allons présenter notre environnement de tests et de validations logicielles et matérielles utilisées pour valider et tester notre approche. Cet environnement est générique pour les architectures qui reçoivent et émettent des impulsions une par une, nous dirons de ces architectures qu'elles sont à traitement unitaire des impulsions.

Notre environnement de tests se base principalement sur un outil de simulation de Deep Neural Network open source développé au CEA, N2D2 [BCB+17]. À la différence des outils actuels TensorFlow [59], Torch [60], Caffe [61], N2D2 propose une chaîne de simulation bi-modèles formelle/impulsionnelle. Il est donc possible d'effectuer des simulations formelles et impulsionnelles dans le même environnement.

De plus, N2D2 implémente des modules de transposition du modèle formel vers le modèle impulsionnel. Ainsi cet outil nous permet de créer un réseau formel, de lui appliquer l'apprentissage formel ainsi que les techniques de réduction de la dynamique et au final effectuer la transposition de ce réseau formel en impulsionnel.

N2D2 facilite l'exportation des réseaux vers les plates-formes génériques du type CPU, GPU, mais aussi vers des plates-formes spécifiques telles que PNeuro ou encore un modèle SystemC UTF (UnTimed Functional model) d'une architecture impulsionnelle pour les CNN avec un traitement entièrement parallèle des impulsions.J'ai utilisé et modifié ce modèle SystemC dans le but de simuler différentes approches d'architectures à traitement unitaire des impulsions, en incluent à ce modèle les aspects de répartition et d'ordonnancement des calculs de ces architectures. J'ai par la suite créé un export NS, utilisé pour paramétrer des simulations matérielles (RTL, PrimeTime et synthèse DC) de ces architectures simulées en SystemC.Ainsi, j'ai créé une équivalence entre : la simulation dans N2D2, le modèle SystemC et pour finir la simulation RTL, le tout pour la validation de notre approche.

En utilisant cet outil, j'ai défini deux chaînes, une d'exploration et une de validation. La chaîne d'exploration permet d'exporter une solution formelle pour un réseau de neurones donné. Puis de le transposer vers son équivalent impulsionnel. Elle permet donc de concevoir le réseau CNN que nous allons utiliser comme référence pour la validation de la simulation matérielle.

La chaîne de validation est utilisée pour valider et explorer les solutions architecturales. Elle se compose d'un modèle SystemC de l'architecture et de l'exportation des paramètres réseau vers la simulation RTL de l'architecture, le tout généré par N2D2. Le modèle SystemC nous permet de simuler le fonctionnement d'une architecture pour un réseau donné et l'exportation permet de configurer la simulation matérielle de l'architecture. Cette chaîne peut-être facilement paramétrée pour simuler différentes répartitions et ordonnancement de calculs de différentes architectures impulsionnelles à traitement unitaire des impulsions. Cette chaîne est suffisamment générique pour être décrite indépendamment d'une architecture. La figure 4.1 pré-

sente une vue d'ensemble de la chaîne d'exploration et de validation.

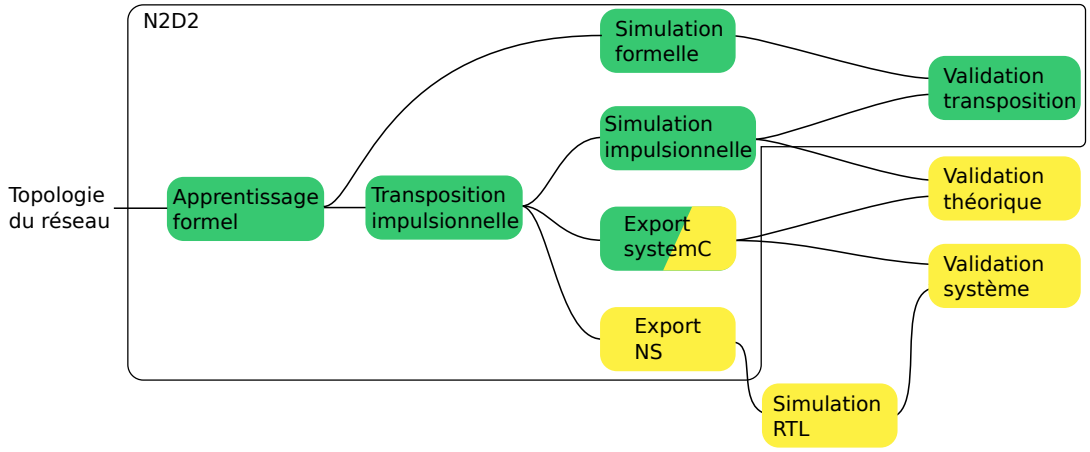

FIG. 4.1: La chaîne de validation N2D2. Cette chaîne nous permet d'évaluer un réseau formel et/ou impulsionnel, de gérer les transpositions du modèle formel vers le modèle impulsionnel, de valider notre approche matérielle en SystemC et d'exporter les paramètres du réseau pour la simulation RTL. En vert les blocs N2D2 préexistants, en jaune les blocs développés/modifiés dans le cadre de ces travaux.

Nous allons présenter le fonctionnement de tous les blocs de cette chaîne. Puis, nous présenterons le détail de notre logique d'exploration et la métrique de validation. Nous illustrerons le fonctionnement de cette chaîne d'exploration par un exemple de transposition d'un réseau formel en son équivalent impulsionnel. Puis nous détaillerons le fonctionnement de la chaîne de validation. Nous finirons par la présentation d'un cas de validation et d'implémentation d'une optimisation du MaxPooling.

4.2 La chaîne d'exploration

La chaîne d'exploration est une chaîne logicielle utilisée pour faire de l'exploration de réseau CNN formel et sa transposition en impulsionnelle. L'exploration de réseau CNN formel consiste à déterminer la topologie du réseau (type, nombre et taille des filtres des différentes couches) en fonction d'un problème donné (*ex.* classification). Pour ce faire, il est possible dans un premier temps de baser notre topologie de réseau sur une préexistante. Puis on effectue un apprentissage en utilisant une base de donnée d'apprentissage qui correspond au problème à traiter. Cela va permettre de déterminer la valeur des variables de ce réseau (typiquement la valeur des pondérations). Puis on teste ce réseau sur une base de test qui contient des données non utilisées lors de l'apprentissage, on va ainsi évaluer la performance de ce réseau. Ces étapes sont répétées jusqu'à l'obtention d'un réseau suffisamment performant, nous utilisons donc une méthode empirique.

À la fin, nous obtenons un réseau CNN formel et les valeurs de ses pondérations. Il nous est alors possible de faire une transposition de ce réseau formel en un réseau impulsionnel selon le principe exposé dans la section 2.4.2. Nous sommes alors en mesure de simuler ce réseau impulsionnel pour le valider par rapport au réseau formel d'origine.

Nous allons présenter dans cette section le détail des étapes de la chaîne d'exploration et les métriques utilisées pour les valider.

4.2.1 Métrique de la chaîne d'exploration

Durant ces différentes étapes, j'ai utilisé deux types de métriques : Premièrement, les métriques de performance qui sont utilisées pour analyser le bon déroulement d'une étape. Dans le cas de l'apprentissage formel et de la simulation formelle, les métriques de performance sont :

- Le taux de succès en apprentissage T_{app}.
- Le taux de reconnaissance T_{test}.
- La matrice de confusion.

Le taux de succès en apprentissage correspond au taux de reconnaissance du réseau sur la base d'apprentissage. Il mesure l'efficacité du réseau à traiter la base d'apprentissage et donc la convergence de l'algorithme d'apprentissage. Cependant, il ne mesure pas l'efficacité du réseau en inférence, car le réseau peut sur-apprendre la base d'apprentissage.

La matrice de confusion permet quant à elle de vérifier le taux de classification classe par classe. Et ainsi de vérifier qu'il n'y a pas de confusion entre les classes. C'est donc un outil d'analyse du réseau.

Le taux de reconnaissance est la capacité du réseau à classifier des données non utilisées lors de l'apprentissage. Ce test permet de certifier le fonctionnement du réseau et de vérifier les problématiques de sur-apprentissage. C'est donc cette valeur qui nous donne l'efficacité réelle d'un réseau.

Les métriques de performance utilisées pour les étapes de transposition impulsionnelle et de simulation impulsionnelle sont identiques à celle des étapes d'apprentissage et de simulation formelle on utilise seulement en plus une métrique spécifique au codage impulsionnel. Ainsi, nous utilisons en plus le nombre d'impulsions présynaptiques moyennes par entrées $MoyI/entrées$ qui permet d'évaluer si le réseau impulsionnel effectue en moyenne moins d'opérations atomiques que son équivalent formel pour la même précision synaptique, voir section 2.5.2.

Cependant, les performances du réseau impulsionnel sont fortement liées aux paramètres de transposition. Ainsi, les résultats sur le nombre d'impulsions et taux de reconnaissance du réseau dépendent des variables de transposition : Δmax, O_{max}, $freq_{max}$ et $freq_{min}$. Le Δmax est la différence en nombre d'impulsions que doit obtenir la sortie d'une classe par rapport à toutes les autres pour être considérée comme

la classe de reconnue. Le O_{max} est le nombre d'impulsions maximales en sortie. S'il est atteint et qu'aucune classe ne s'est démarquée, alors on considère que la classe de sortie est la classe de plus haute fréquence. La $freq_{max}$ et $freq_{min}$ définissent la relation linéaire entre la valeur L d'une l'entrée et la fréquence d'impulsion associée à cette entrée $freq_{out}$, voir équation 4.1.

$$freq_{out} = freq_{max} + (freq_{min} - freq_{max}) \times L \tag{4.1}$$

Les variables Δmax et O_{max} définissent l'interprétation des impulsions de sortie et peuvent influencer le taux de succès et il est fixé empiriquement, voir figure 4.2. Ainsi en général plus le Δmax est grand plus les performances du réseau impulsionnel vont converger vers les performances du réseau formel, car plus la précision de la sortie sera grande. Cependant, cela augmentera le nombre d'impulsions propagées dans le réseau et donc le $MoyI/entrées$, réduisant voir annulant le gain en complexité calculatoire par l'utilisation du codage impulsionnelle.

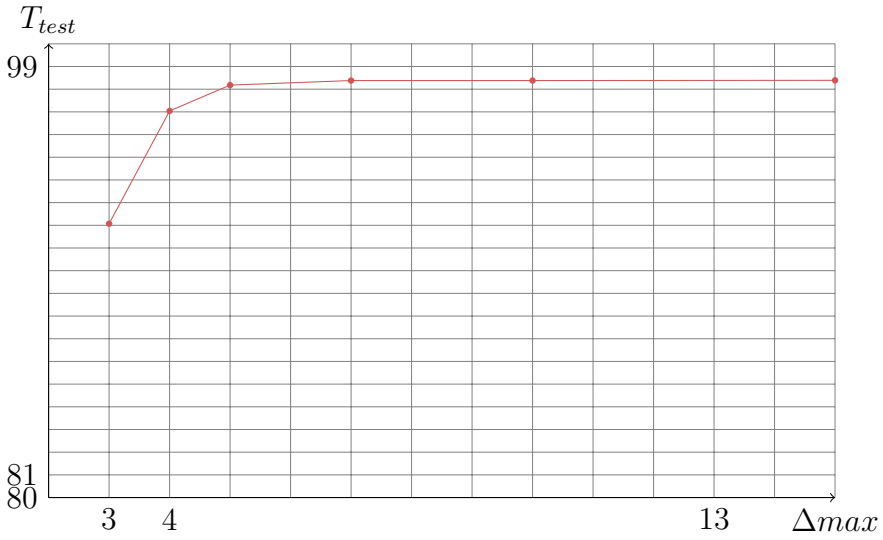

FIG. 4.2: Influence de Δmax sur les performances de classification en test d'un réseau conçu pour la base GTSRB.

Deuxièmement, la métrique de validation. La métrique de validation ne concerne que les étapes de simulation impulsionnelle et de simulation formelle. Cette métrique n'évalue pas l'efficacité des étapes, mais leur équivalence dans le sens applicatif. La métrique de validation utilisée est la différence entre le taux de succès des réseaux impulsionnels et le taux de succès des réseaux formels, le $\Delta T_{test} = T_{test}impultionnel(\Delta max, O_{max}) - T_{test}frame$. Ainsi, si le résultat de la différence est nul, le réseau impulsionnel est équivalent du point de vue applicatif au réseau formel, sinon il y a une erreur dans la transposition.

Toute cette chaîne d'exploration nous permet de concevoir un réseau formel, de faire l'apprentissage et de valider les performances de ce réseau pour une application

donnée. Nous pouvons par la suite transposer ce réseau formel en un réseau impulsionnel et travailler sur l'efficacité du codage impulsionnel pour optimiser le réseau. Ces étapes sont itératives tant que les performances du réseau formel et de son équivalent impulsionnel ne sont pas satisfaisantes, voir figure 4.3. À la fin, de cette chaîne nous obtenons toutes les caractéristiques du réseau impulsionnel à porter sur notre architecture *c.-à-d.* la topologie du réseau, la valeur des pondérations, le Δmax, le O_{max}, la $freq_{max}$, la $freq_{min}$ et les impulsions de sorties du réseau impulsionnel.

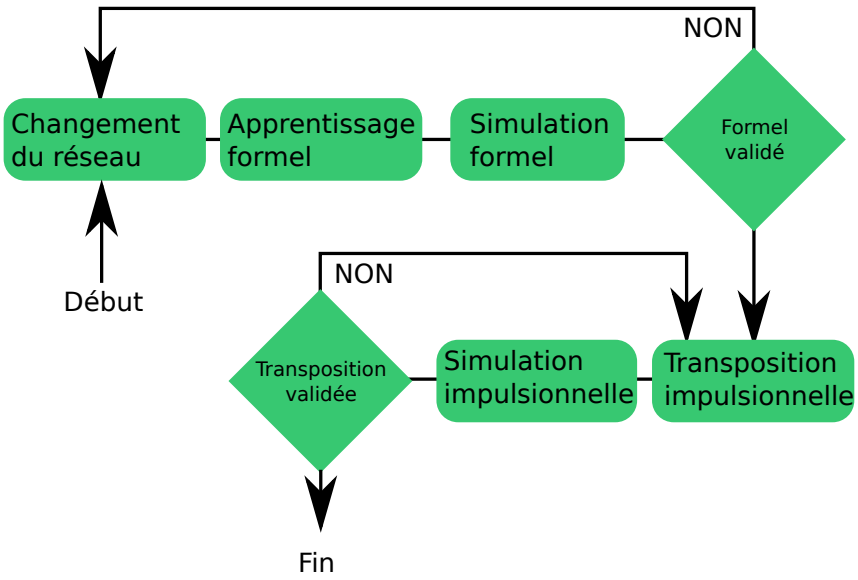

FIG. 4.3: Détail de la chaîne d'exploration présentant les conditions de validation des étapes et les différentes possibilités d'itérations.

4.2.2 Exemple de déroulement de la chaîne d'exploration

Pour illustrer notre chaîne d'exploration, nous allons présenter un cas d'utilisation. Cet exemple est basé sur une application de classification de chiffres manuscrits sur la base Mixed National Institute of Standards and Technology database (MNIST). Cette base est un classique des réseaux de neurones convolutionnels sur laquelle de nombreuses applications existent [62]. Cette base contient des images de 28×28 pixels en niveaux de gris $0 - 255$, des chiffres de 0 à 9. Elle est constituée de 60 000 images pour la phase d'apprentissage et 10 000 images pour la phase de test.

Pour cette application, nous avons créé une topologie de réseau, voir figure 4.4. Ce réseau n'est pas optimal, mais nous permet d'illustrer notre exemple. Le réseau a 10 sorties correspondant aux 10 classes de chiffres (0 à 9). On note que l'image d'entrée est passée de 28×28 pixels à 24×24 pixels en utilisant une fonction de "crop" de Opencv [63], le but étant de réduire la complexité, car les chiffres sont centrés avec une marge.

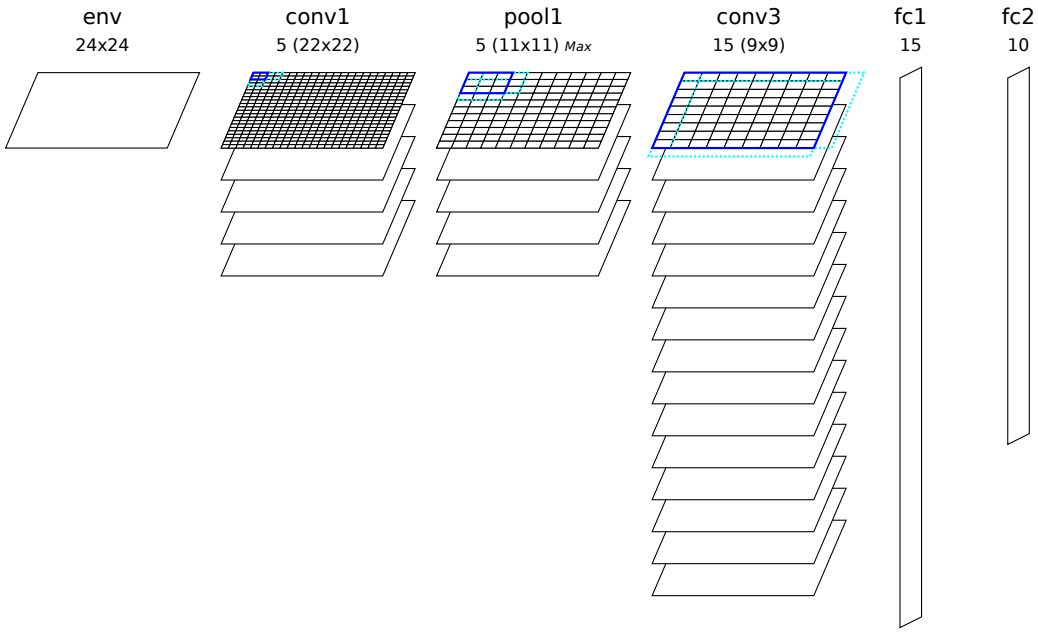

FIG. 4.4: Réseau convolutionel de classification de la base MNIST.

Les réglages des paramètres de la transposition utilisés sont présentés dans le tableau 4.7. Ces paramètres de la transposition ont été déterminés empiriquement selon la méthodologie exposée précédemment en testant différentes valeurs puis en analysent leur effet récursivement. Les résultats des étapes apprentissages et des simulations formelles sont présentés figure et tableau 4.6.

Notre réseau obtient un score de $98,1\%$ taux de reconnaissance ce qui pas n'est très performant par rapport au $99,79\%$ de taux de reconnaissance de l'état de l'art [64], mais suffisant pour illustrer notre logique de transposition. La matrice de confusion est proche de l'identité ce qui montre qu'il n'y a pas de confusion entre les classes. Ce réseau formel est suffisamment performant pour être utilisé dans les étapes de transpositions et simulation impulsionnelle. Au niveau de la transposition impulsionnelle nous avons un $\Delta T_{test} = 0$ ce qui montre que la transposition est correcte et que le réseau impulsionnel est aussi efficace que le réseau formel d'un point de vue applicatif. Le nombre d'impulsions pré-synaptiques moyennes par entrées $MoyI/entrées$ est de $1,27$, il en dessous des 9 impulsions pré-synaptique par entrées qui est la valeur maximale théorique pour laquelle il y a moins d'opérations atomiques en impulsionnel que d'opérations atomiques en formel pour le calcul des couches de convolution et FC, cette valeur a été déterminée dans la section 2.5. Ainsi, les couches de convolution et de FC impulsionnelles dans ce réseau sont en moyenne plus efficaces en matière d'opérations atomiques que leur équivalent formel.

Cependant, le nombre d'impulsions pré-synaptique moyennes par entrée est au-dessus de 1 impulsion pré-synaptique par entrées qui est la valeur maximale théorique pour laquelle il y a moins d'opérations atomiques en impulsionnel qu'en formel

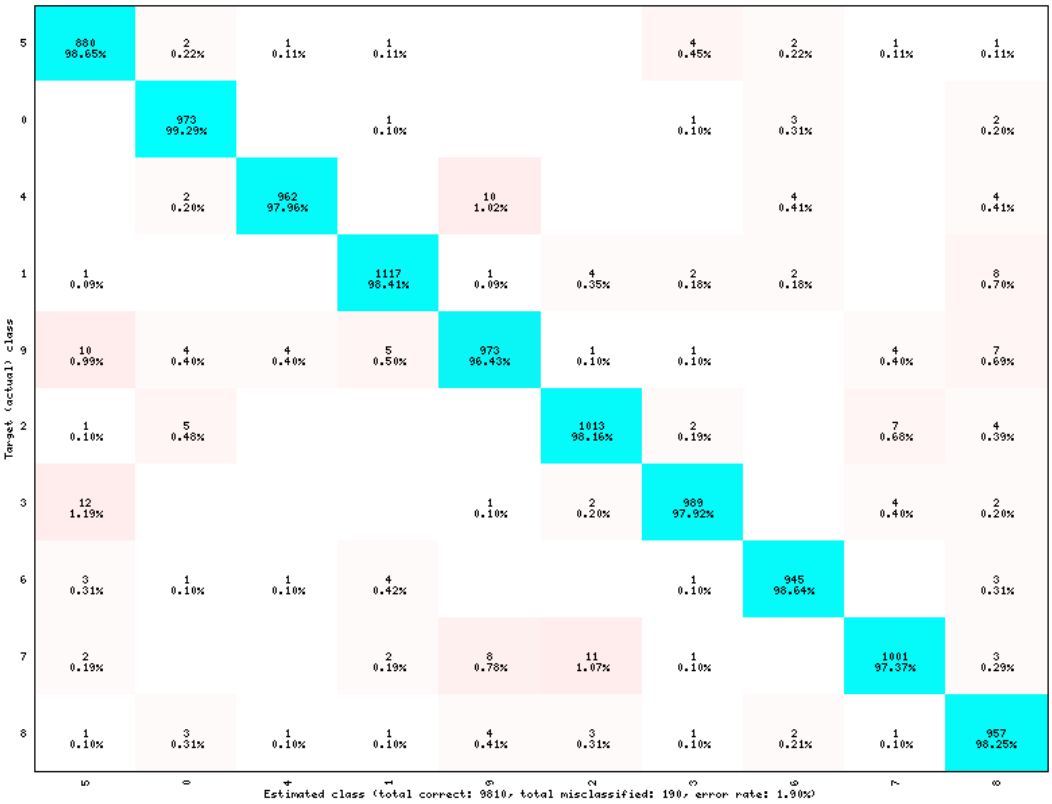

FIG. 4.5: Matrice de confusion en test.

variable	formel	impulsionnel
T_{app}	94,36%	
T_{test}	98,1 %	98,1 %
$MoyI/entrées$		1,27

FIG. 4.6: Résultat de transposition du formel vers impulsionnel pour une topologie de réseau fixé et une application de classification de la base MNIST.

variable	valeur
Δmax	10
O_{max}	1 000
$freq_{min}$	10 Hz
$freq_{max}$	1 MHz

FIG. 4.7: Configuration de la transposition du formel vers impulsionnel pour une topologie de réseau fixé et une application de classification de la base MNIST.

pour le calcul des couches de MaxPooling, voir la section 2.5. Ainsi, la couche de MaxPooling impulsionnelle dans ce réseau est donc en moyenne moins efficace en termes d'opérations atomiques que la même couche de MaxPooling formelle, voir

la section 2.5.2. Au vu de ces résultats, le réseau impulsionnel est équivalent au réseau formel, les résultats en $MoyI/entrées$ nous permettant d'envisager un gain en termes de complexité calculatoire, nous pouvons donc l'utiliser pour la chaîne de validation.

4.3 La chaîne de validation

La chaîne de validation est utilisée pour passer du réseau impulsionnel à son implémentation et validation pour une architecture. Une grande partie de cette chaîne n'a pas été conçue pour une architecture bien définie, mais pour les architectures à traitement unitaire des impulsions en général.Nous allons donc dans cette section présenter son fonctionnement général sans définir une architecture spécifique. Le but est de présenter les différents outils que j'ai mis en place et leur utilisation pour la validation de notre approche.

La chaîne de validation est décomposable en deux étapes : simulation SystemC et l'exportation des configurations de l'architecture. L'étape de simulation SystemC permet de vérifier le bon déroulement du calcul de l'inférence d'un réseau CNN donné par la chaîne d'exploration pour une architecture donnée. L'exportation des configurations est utilisée pour configurer les simulations matérielles de l'architecture cible. Nous allons détailler le fonctionnement de toutes les étapes de la chaîne de validation.

4.3.1 SystemC : simulation équivalente au niveau impulsion

À la suite de la chaîne d'exploration, j'ai mis en place une abstraction équivalente au niveau impulsion des architectures à traitement unitaire des impulsions (architectures ne traitant en entrées et ne fournissant en sortie qu'une impulsion à la fois) en SystemC. Cette abstraction n'est pas "cycle accurate", mais l'ordonnancement des impulsions peut être paramétré en fonction de l'architecture cible. Elle est utilisée pour l'étude du bon déroulement du calcul et comme référence pour la validation finale de l'instanciation matérielle. Nous allons dans un premier temps expliquer son fonctionnement, nous verrons son utilisation et nous finirons par l'explication de son utilité en matière de validation matérielle.

L'architecture SytemC est décomposable en blocs. Chaque bloc SytemC correspond à une couche du CNN fourni par la chaîne d'exploration. Il existe trois types de blocs pour les trois types de couches classiquement utilisés dans les CNN : la convolution, le MaxPooling et la couche FC. Chaque couche/blocs est interconnectée par un bus asynchrone, utilisé pour transmettre les impulsions une par une entre les différents blocs. Les parties communication entrées/sorties ainsi que l'ordonnancement des impulsions sont identiques pour tous les types blocs, seul le traitement

des impulsions diffère, voir figure 4.8.

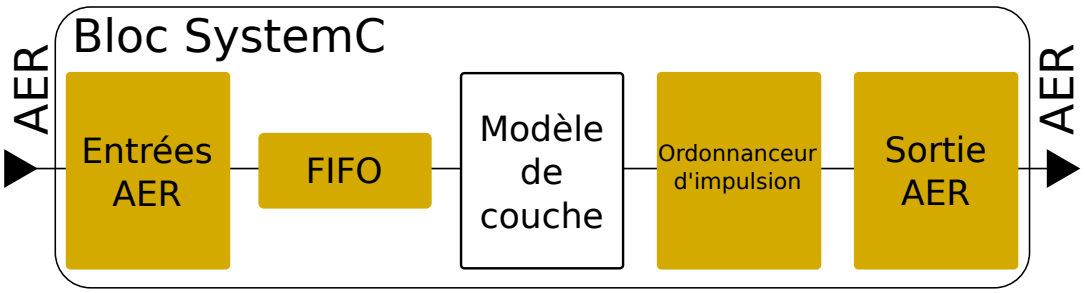

FIG. 4.8: Représentation générique d'un bloc SytemC d'une couche du réseau. Les sous-blocs en jaune sont communs à tous les types de couches SystemC, seul le calcul du modèle de la couche diffère. Dans les blocs communs, on retrouve les sous-blocs pour la communication AER et la FIFO d'entrée et un bloc d'ordonnancement des impulsions de sortie permettant de reproduire l'ordonnancement de l'architecture cible.

Au niveau des communications, les impulsions sont transmises entre les blocs via un bus AER (Address Event Representation) figure 4.9, nous avons choisi ce type de bus pour plusieurs raisons : Premièrement, le bus AER nous permet de transmettre les impulsions une par une, ce qui est adapté à notre logique de simulation d'architecture à traitement des impulsions unitaires. Deuxièmement, ce type de bus est utilisé dans les capteurs bio-inspirés, par exemple pour les rétines artificielles [65, 66], c'est donc un protocole classique de communication impulsionnelle.

Les communications AER utilisent un bus parallèle pour transmettre les coordonnées des impulsions. Lorsqu'une donnée est disponible, l'envoyeur émet un signal de requête (R) au récepteur. Lorsque le récepteur a lu la donnée, il retourne à l'envoyeur un signal d'acquittement (A). Le signal de requête retourne à 0 puis le signal d'acquittement retourne à 0. Tant que le récepteur n'a pas retourné un d'acquittement, la donnée et la requête sont maintenues.

Les impulsions sont stockées dans une FIFO d'entrée qui est lue par le bloc de calcul. Si la FIFO d'entrée est pleine le bloc ne permet plus l'entrée de nouvelle impulsion. Ce qui implique que le bloc précédent doit attendre pour transmettre ses impulsions de sortie. La communication est donc un aspect critique de la simulation, car un défaut engendre le dysfonctionnement de tous les blocs.

Le traitement d'une impulsion d'entrées est effectué par le "modèle de couche", qui fournit en sortie les impulsions de sortie de la couche. Le temps de traitement est fixé à un cycle ce qui n'est pas réaliste, mais cela n'impacte pas notre logique de simulation qui n'est pas "cycle accurate". Le "modèle de couche" ne dépend pas de l'architecture, mais uniquement du type de couche simulée, car, quelleque soit l'architecture utilisée le traitement des impulsions doit être identique, il s'agit d'un modèle purement fonctionnel. Ceci augmentant la flexibilité de notre modèle SystemC en réduisant le nombre de modifications à effectuer pour changer d'architecture. Le

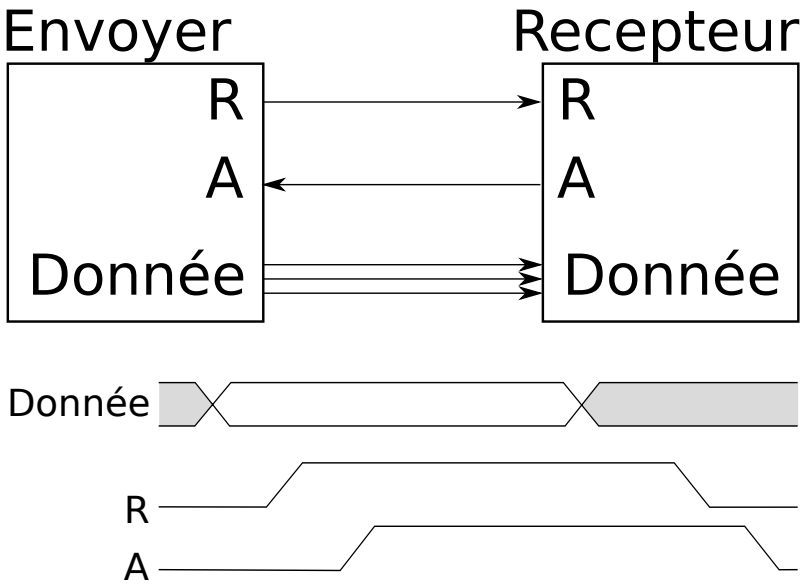

FIG. 4.9: Câblage et chronogramme du protocole AER. Le signal de requête R annonce la présence d'une donnée en entrée, le signal acquittement A signale la lecture de la donnée.

"modèle de couche" peut aussi être utilisé pour tester des modèles différents et leurs influences sur les performances de classification, voir un exemple dans la section 4.4.

L'"ordonnanceur" agit sur l'ordre des impulsions de sortie du bloc, c'est lui qui modélise l'ordre des calculs d'une architecture. Ceci est possible, car les communications ne permettent qu'une impulsion à la fois, donc peu importe la logique du calculateur, les impulsions de sortie vont devoir être sérialisées. Ainsi, l'"ordonnanceur" prend en entrée les impulsions de sortie de la couche, fournie par le "modèle de couche". Ces impulsions sont identifiées par leur numéro dans la liste de sortie N_{liste} et leur adresse de sortie (o_x, o_y, o_z). L'"ordonnanceur" modifie l'ordre de sortie des impulsions de la liste en fonction de leur (o_x, o_y, o_z) et en fonction de l'ordre de sérialisation et de calcul dans l'architecture cible, voir un exemple figure 4.10. Cela nous garantit que pour une même impulsion d'entrée, ce sont les mêmes impulsions de sortie et dans le même ordre entre une couche SystemC et une couche de l'architecture cible. Cette logique permet une grande flexibilité d'exploration de différentes architectures, car le changement de l'architecture cible ne nécessite que la modification de l'"ordonnanceur".

Pour former un réseau, les blocs SystemC correspondant aux différentes couches du réseau sont chaînés. Cette chaîne est connectée en entrée à un bloc de génération d'impulsions et en sortie à un bloc d'analyse, voir figure 4.11. Le bloc de génération d'impulsion convertit une donnée d'entrée en impulsions en fonction de $freq_{max}$ et $freq_{min}$, cette conversion est identique à la simulation impulsionnelle. Ce qui nous permet de garantir que les impulsions en entrées de la simulation SystemC sont les mêmes que dans la simulation impulsionnelle. Le bloc d'analyse détermine la classe en fonction des impulsions de sortie du réseau SystemC, son fonctionnement est

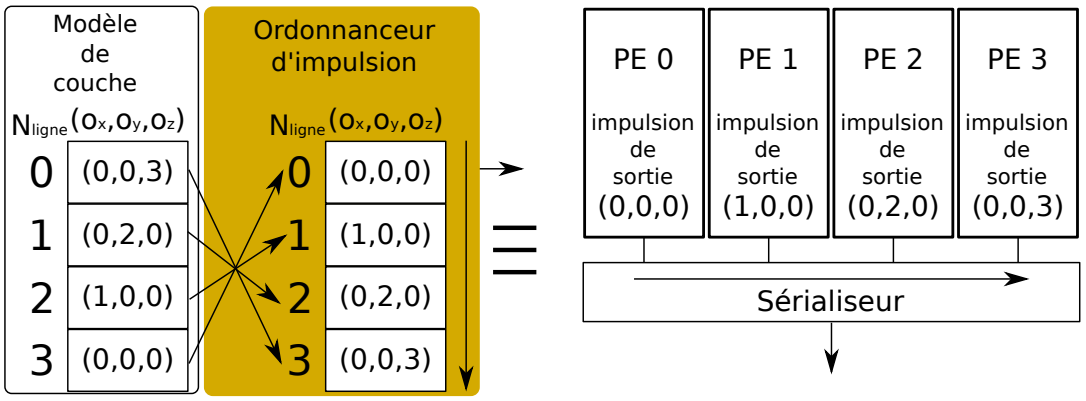

FIG. 4.10: Représentation d'un réordonnancement de 3 impulsions de sortie pour correspondre au placement matériel des calculateurs pour garantir l'équivalence en ordre des impulsions du modèle SytemC et du matériel.

identique à la simulation impulsionnelle. Le bloc d'analyse utilise les mêmes valeurs de Δmax et de O_{max} que la simulation impulsionnelle. On peut donc comparer la sortie du bloc d'analyse à la sortie de simulation impulsionnelle pour valider le fonctionnement applicatif de la simulation SystemC.

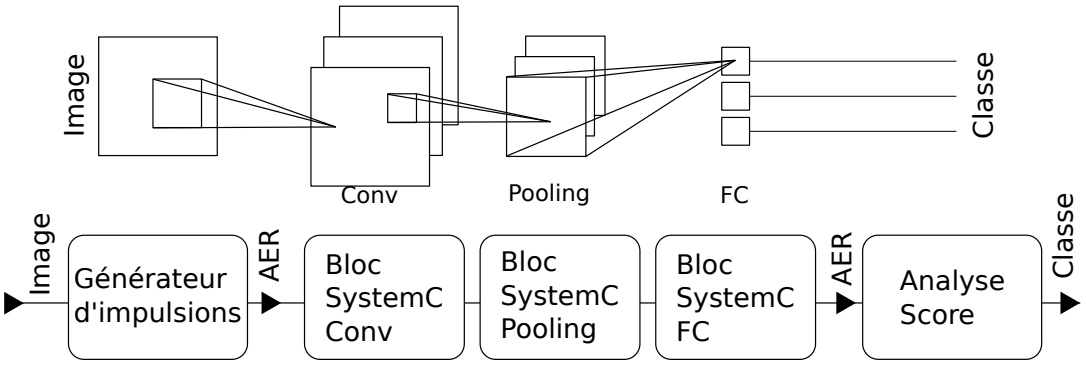

FIG. 4.11: Représentation générique d'un réseau en SystemC. Chaque bloc correspond à une des couches du réseau et tous les blocs sont chaînés dans le même ordre que le réseau. La chaîne de blocs est encadrée par deux blocs de gestion d'impulsion, le bloc d'entrée qui convertit une image en impulsions et le bloc de sortie qui convertit les impulsions en classe.

Nous avons défini un modèle SystemC générique pour la simulation d'architecture à traitement unitaire d'impulsion, permettant une grande flexibilité d'instanciation d'architecture. Le fonctionnement de ce modèle SystemC peut être vérifié grâce à la simulation impulsionnelle. Je vais maintenant expliquer la méthode que j'ai mise en place pour valider le fonctionnement d'une architecture en utilisant ce modèle SystemC. Pour cela, je présenterai la mise en place des simulations matérielles puis l'utilisation des résultats de ces simulations pour la validation du fonctionnement de l'architecture et de la caractérisation de ses performances en énergie et temps de

traitement.

4.3.2 Configuration et simulations matérielles

L'étape d'exportation NS, est la génération des configurations pour la simulation RTL (Register Transfer Level) de l'architecture. Cette étape nécessite d'avoir spécifié l'architecture, mais il est possible d'identifier les données à générer et différentes simulations à effectuer sur le RTL pour caractériser l'architecture.

L'étape d'exportation NS va fournir 3 types de données au simulateur : les données du banc de test, les données de configuration de l'architecture et les données de validation. Toutes ces données sont strictement les mêmes qu'utilisées dans le SystemC. Les données du banc de test sont les adresses des impulsions d'entrée à fournir à l'architecture, les adresses des impulsions de sortie du réseau et les valeurs de Δmax et de O_{max}. Ceci nous permet d'injecter exactement les mêmes données d'entrée au simulateur RTL que celles utilisées dans le modèle SystemC. Les données pour la configuration de l'architecture sont de différents types. Premièrement, la configuration mémoire des pondérations. Deuxièmement, les différentes variables de réglage de l'architecture par exemple : nombre de couches, taille des couches, types de couches, etc. Cette exportation est dépendante de l'architecture, le détail de la répartition mémoire pour notre architecture finale est présenté dans la section 5.2. On obtient donc à la fin de cette étape une simulation RTL avec des paramètres identiques au modèle SystemC.

La simulation RTL est alors utilisée pour valider le bon déroulement du calcul dans l'architecture. Sachant que le modèle SystemC reproduit la réponse d'une couche à une impulsion d'entrée de l'architecture cible. La réponse d'une couche à une impulsion d'entrée dans le modèle SystemC et dans la simulation de l'architecture est identique en matière de nombre d'impulsions de sortie et d'ordre et ceci étant vrai pour toutes les couches du réseau. La validation fonctionnelle de l'architecture consiste donc à comparer de toutes les impulsions transmises entre les couches SystemC et entre les couches de l'architecture. L'égalité stricte de toutes les impulsions générées par le SystemC et la simulation RTL implique l'équivalence entre le SystemC et la simulation RTL, voir la représentation schématique de cette validation dans la figure 4.13. L'équivalence entre la simulation SystemC et la simulation RTL implique l'équivalence entre la simulation RTL et la simulation impulsionnelle et pour finir implique l'équivalence applicative de l'architecture et de la simulation formelle.

Pour mesurer les performances en temps de traitement et en énergie consommée de l'architecture, j'utilise plusieurs outils, une synthèse DC et PrimeTime , voir figure 4.12.

La mesure de performance consiste en une synthèse DC et une simulation PrimeTime. La synthèse DC nous permet d'obtenir la surface S, la consommation en

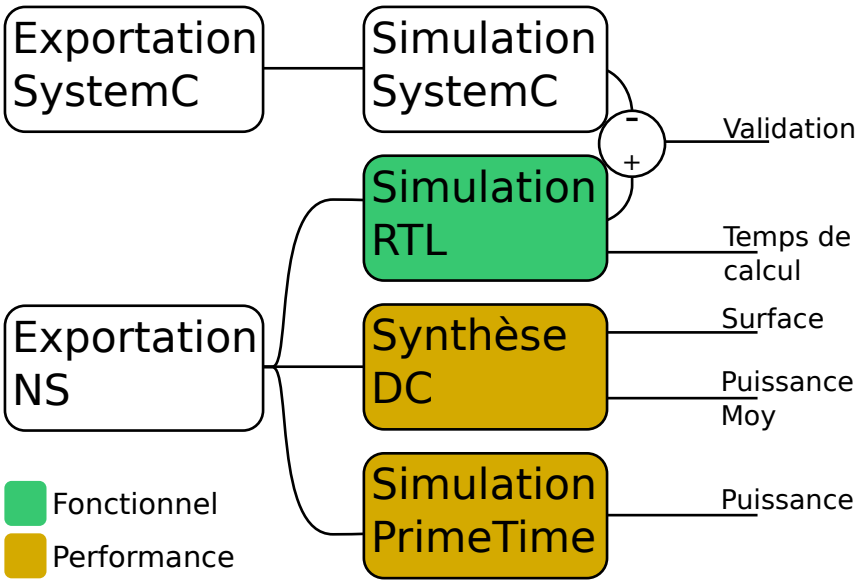

FIG. 4.12: Détail de la simulation RTL, la simulation fonctionnelle en vert est utilisée pour valider le fonctionnement de l'application et nous donne le temps de calcul. Les blocs de performance sont utilisés pour estimer la caractéristique physique de l'architecture à savoir sa surface et sa puissance moyenne, mesurée pour une application donnée.

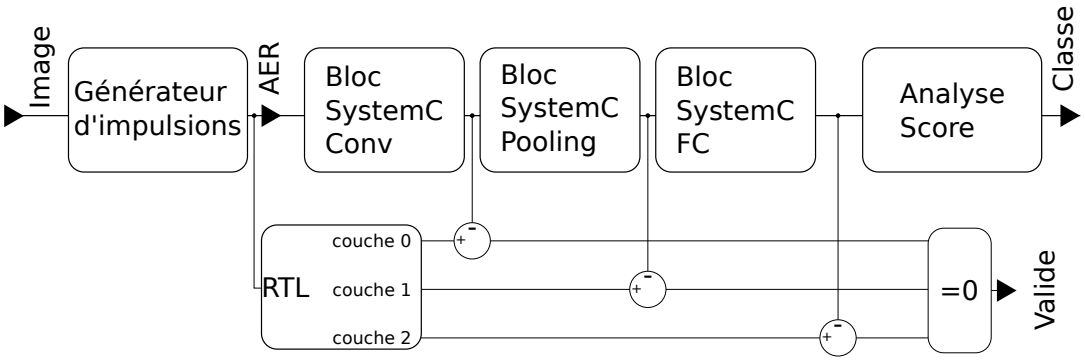

FIG. 4.13: Chaîne de validation du RTL. Comparaison des sorties RTL et SystemC pour toutes les couches, la validation est l'égalité des sorties de toutes les couches en matière de nombre d'impulsions, d'adresse des impulsions et d'ordonnancement des impulsions.

Watt pour une activité moyenne de 50 % P_{moy} et la fréquence max de fonctionnement f_{max} de l'architecture sans les mémoires. En connaissant le nombre d'impulsions pré-synaptiques traitées en parallèle $NbPara_{impul}$, la précision des pondérations Nbi_{bit} avec laquelle on peut déduire la complexité des opérations C_i, on peut estimer les performances en opération par seconde $OP/s = C_i \times NbPara_{impul} \times f_{max}$ et les performances en opération par Joule $Op/J = \frac{OP/s}{P_{moy}}$ de notre architecture sans mémoire pour une activité moyenne de 50%.

La simulation PrimeTime quant à elle nous permet d'évaluer la consommation de l'architecture sans les mémoires, à une fréquence de f_{max} et pour un réseau donné. Cette simulation nous permet une plus grande précision dans l'estimation

des performances de l'architecture. La consommation des mémoires est quant à elle estimée en fonction de leur fréquence de fonctionnement f_{max} de leurs taux de lecture/écriture déduits lors de la simulation RTL. Cette chaîne nous offre donc tous les outils nécessaires à la validation et à l'estimation des architectures à traitement unitaire des impulsions.

4.4 Validation et apport d'un modèle de Maxpooling approximé

Comme décrit dans la section 2.5, la couche de MaxPooling impulsionnelle ne permet pas de gain en matière de complexité calculatoire, mais plus problématique elle nécessite une grande quantité de mémoire pour stocker le taux d'activation de toutes ses entrées. Pour remédier à cette problématique, j'ai proposé une simplification du MaxPooling impulsionnel, le MaxCompt. Pour cela j'ai utilisé toute la chaîne de validation N2D2 présentée auparavant à l'envers pour certifier que l'approximation matérielle du MaxPooling a peu d'impact sur les performances applicatives des réseaux implémentés.

Nous allons dans un premier temps présenter le calcul de MaxPooling impulsionnel. Ce qui nous permettra d'identifier les ressources nécessaires à son calcul et les optimisations possibles. Puis, nous introduirons le modèle approximé, le MaxCompt son fonctionnement, ses conditions de fonctionnement et la conséquence sur les performances de classification. La validation du comportement du MaxCompt effectué, nous étudierons son impact en matière de complexité calculatoire et d'implémentation matérielle.

4.4.1 Le MaxCompt

La sortie d'un neurone d'une couche de MaxPooling impulsionnel est égale à la plus haute fréquence qu'il reçoit en entrée, voir figure 4.14. Si l'on considère un neurone d'une couche de MaxPooling avec $O \times C$ synapses, il faut une mémoire de $O \times C$ bloc de N bits pour stocker l'intégralité des activités synaptiques du neurone. Au niveau calculatoire la fonction nécessite un incrémenteur pour modifier l'activité d'une synapse lors de la réception d'une impulsion et une fonction de tri pour déterminer la synapse la plus active, voir figure 4.15 cette complexité est égale à la complexité de la fonction de MaxPooling formelle.

De plus, si on considère le circuit numérique d'un neurone impulsionnel, celui-ci ne dispose que d'une seule mémoire pour stocker son intégration. Les besoins mémoires du MaxPooling impulsionnel sont incompatibles avec les ressources d'un neurone impulsionnel. Pour implémenter le MaxPooling tel quel, il nous faut un matériel dédié et le coût énergétique risque d'impacter les performances de notre

architecture.

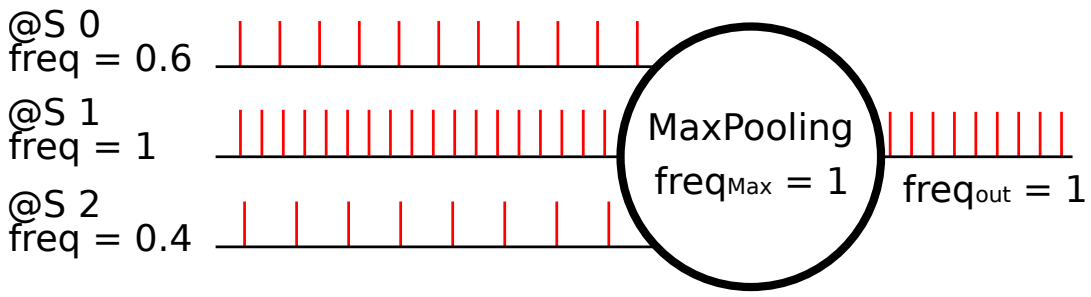

FIG. 4.14: Comportement de la fonction de MaxPooling impulsionnelle soumise à trois entrées de fréquence différentes. On remarque que les impulsions de sortie du neurone correspondent aux impulsions d'entrée de la synapse la plus active.

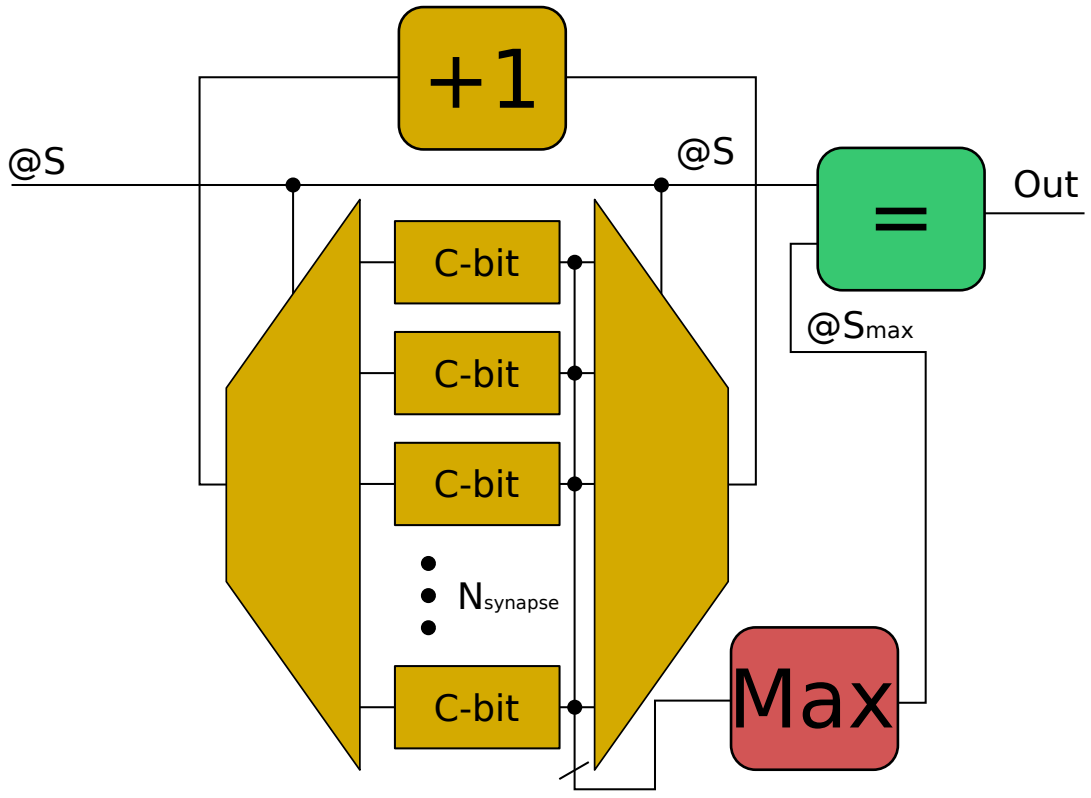

FIG. 4.15: Implémentation numérique possible de la fonction de MaxPooling impulsionnelle. Cette implémentation se décompose en trois parties : en jaune, le compteur d'activité des synapses, en rouge la fonction de tri Max pour déterminer l'adresse de la synapse la plus active $@S_{max}$ et pour finir, en vert la fonction de déclenchement du neurone qui compare l'adresse de la synapse reçue $@S$ à l'adresse de la synapse la plus active.

Pour remédier à cette problématique, j'ai introduit une optimisation de la couche de MaxPooling, le MaxCompt. Cette approximation se base sur le fait que la différence d'activité entre les synapses doit être assez importante pour que l'opération

de MaxPooling induise une altération significative de l'activité en sortie par rapport à l'entrée. Ainsi, on peut déterminer la synapse la plus active par rapport à l'activité générale des entées, cela nous évite de stocker toutes les activités de toutes les synapses. Le modèle MaxCompt utilise en matière de mémoire un compteur d'activité différentielle A sur N bits et une mémoire pour stocker l'adresse de la synapse la plus active $@S_{max}$ sur $\lceil log_2(C) \rceil$. Au niveau de la partie calculatoire, le MaxCompt utilise un incrémenteur/décrémenteur pour modifier un compteur d'activité différentielle. Lors de la réception d'une impulsion sur la synapse $@S$, celle-ci est comparée à $@S_{max}$, voir la figure 4.16 Si $@S = @S_{max}$ alors $A = A + 1$ et le neurone se déclenche. Si $@S \neq @S_{max}$ alors $A = A - 1$. Le changement de $@S_{max}$ est effectué lorsque $A = 0$ alors $@S_{max} = @S$, $A = Ref$ et le neurone ce déclenche, voir figure 4.17. La valeur de Ref étant la valeur d'initialisation du compteur d'activité différentielle. Les besoins mémoires de cette optimisation sont réduits par rapport au MaxPooling classique, on passe de $O \times C \times N$ bits à $\lceil log_2(C) \rceil + N$ bits. Les besoins calculatoires ont aussi été réduits avec le remplacement de la fonction de tri par deux comparateurs et un incrémenteur/décrementeur.

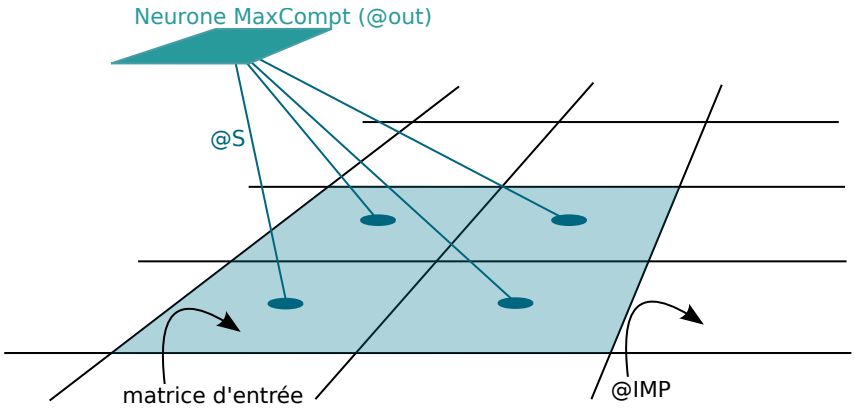

FIG. 4.16: Représentation de la connectivité pour un neurone avec un champ réceptif 2D. $@out$ est l'adresse de sortie du neurone, $@S$ l'adresse de la synapse et $@IMP$ l'adresse de l'impulsion reçue à un instant t.

Pour valider cette optimisation en matière d'impact sur la performance de classification totale, nous avons utilisé une comparaison entre la simulation impulsionnelle et la simulation du modèle SystemC. La vérification consiste à comparer le résultat de la simulation utilisant un modèle MaxPooling au résultat du modèle SystemC qui utilise le MaxCompt pour un même réseau et une même application. Pour cela, j'ai instancié en SystemC une couche de MaxCompt, cette instanciation a été validée par la comparaison avec la simulation RTL. La validation effectuée, j'ai ensuite comparé le taux de reconnaissance obtenu en SystemC et en simulation impulsionnelle utilisant le modèle MaxPooling classique pour quantifier l'impact de cette optimisation sur le taux de reconnaissance, voir la figure 4.18 ,et sur le nombre d'impulsions

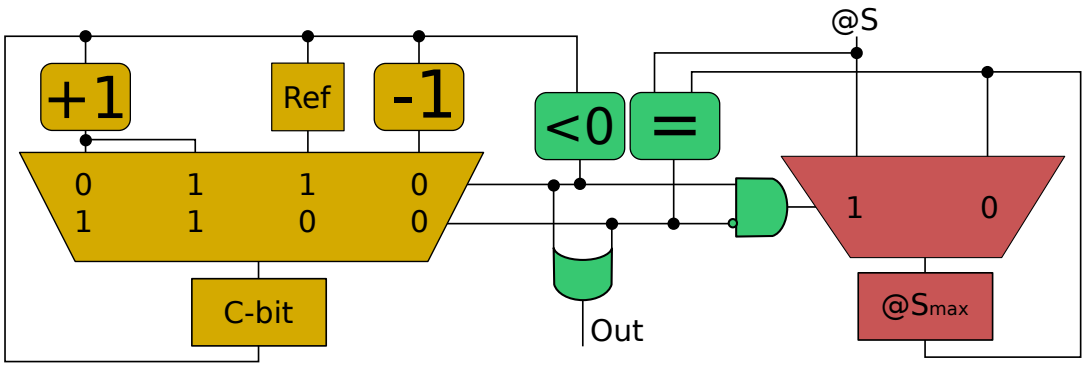

FIG. 4.17: Implémentation numérique possible de la fonction de MaxCompt impulsionnelle. Cette implémentation se décompose en trois parties : en jaune le compteur d'activité différentielle, en rouge, l'unité de sauvegarde de l'adresse de la synapse la plus active $@S_{max}$ et en vert, la logique de déclenchement, de contrôle du compteur et de la sauvegarde de $@S_{max}$.

pré-synaptique par entrées voir figure 4.19 en fonction de Ref exprimé en nombres d'éléments de l'entrée des neurones C voir section 2.2.2.

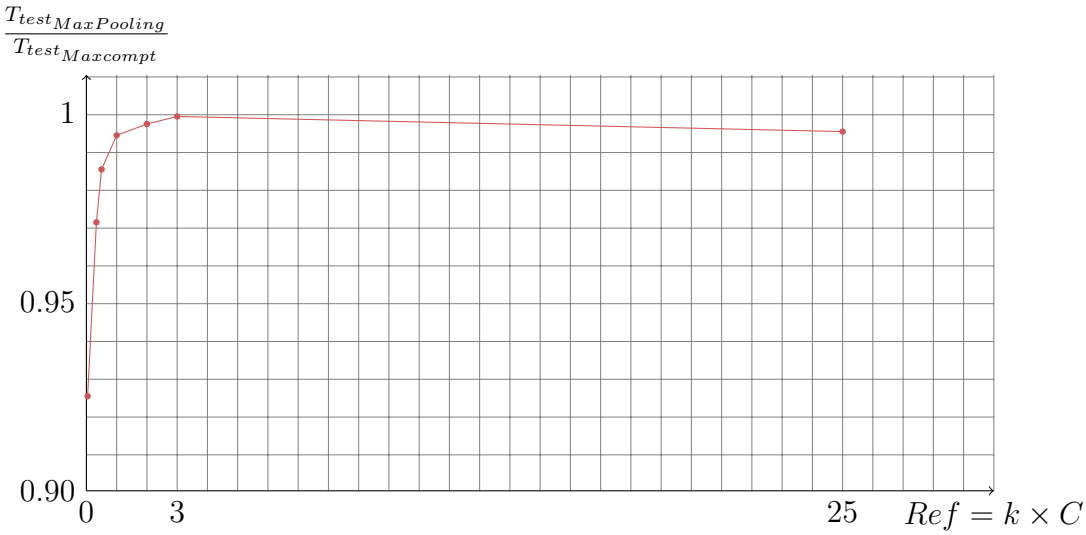

FIG. 4.18: Rapport entre le taux de test T_{test} d'un réseau utilisant le modèle MaxPooling et un réseau utilisant le modèle Maxcompt en fonction de la référence Ref. La valeur de la référence correspond à k fois le nombre d'éléments du champ récepteur des neurones de la couche.

L'utilisation du MaxCompt à la place du MaxPooling engendre $< 0,1\%$ de perte sur le taux de reconnaissance du réseau si la référence Ref est égale à $3 \times C$ et une augmentation de 8% du nombre d'impulsions moyennes pré-synaptique par entrées, ce qui au vu du gain mémoire et calculatoire me semble acceptable.Par la suite, nous avons effectué un grand nombre d'évaluations de l'effet du MaxCompt sur les

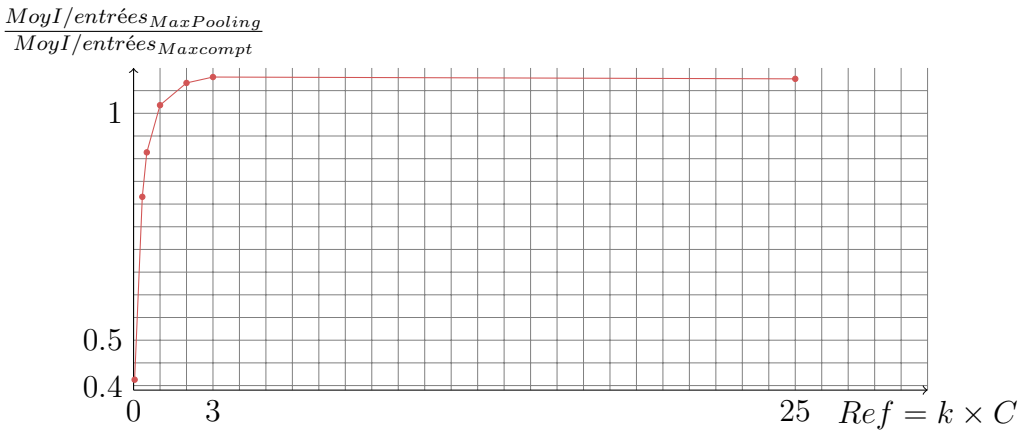

FIG. 4.19: Rapport entre les nombres d'impulsions moyennes pré-synaptique par entrées $MoyI/entrées$ d'un réseau utilise le modèle MaxPooling et un réseau utilisant le modèle Maxcompt en fonction de la référence Ref. La valeur de la référence correspond à k fois le nombre d'éléments du champ récepteur des neurones de la couche.

réseaux classiques, ce qui nous a permis de confirmer ces résultats de perte et la valeur de la Ref à $3 \times C$.

Sachant que pour le bon fonctionnement le compteur d'activité doit être initialisé à environs $3 \times C$, ceci nous donne une contrainte sur le nombre de bits du compteur d'activité différentielle N qui doit être au minimum de $\lceil log_2(3 \times C) \rceil$.

Les chaînes précédemment vues sont donc utilisables pour la mesure de l'impact d'une approximation matérielle. Au final, le modèle MaxCompt est une alternative moins coûteuse en matière d'implémentation de la couche MaxPooling pour les réseaux à impulsions positives. Nous allons ensuite nous intéresser à l'impact sur la complexité calculatoire de l'utilisation de cette approximation.

4.4.2 Impact du MaxCompt en termes d'opérateur atomique

Il est possible d'exprimer le nombre d'opérations atomiques que fait un neurone de MaxCompt à la réception d'une impulsion et la quantité mémoire que représente ce neurone. Ainsi le neurone de MaxCompt contient un additionneur, un soustracteur et deux comparateurs de $N = \lceil log_2(3 \times C) \rceil$ bits ce qui représente, selon notre définition d'opérateurs atomiques $4 \times \lceil log_2(3 \times C) \rceil$ opérations atomiques. Au niveau de l'empreinte mémoire, le neurone de MaxCompt contient deux blocs mémoire : un de N bits pour l'activité A et un de $\lceil log_2(C) \rceil$ pour l'adresse $@S_{Max}$. En considérant une couche de MaxCompt avec un Stride de 1 et pas de ZéroPading et pour 1 tenseur d'entrée et 1 tenseur de sorties ne contenant qu'une seule matrice 2D, la complexité calculatoire en fonction du nombre d'impulsions reçues $NbImp$ de cette couche est donnée par l'équation 4.2 et l'empreinte mémoire est donnée par l'équation 4.3.

$$Op_{MaxCompt_{impul}} = NbImp_{\text{pré}} \times 4 \times \lceil log_2(C) \rceil \times O \tag{4.2}$$

$$Mem_{MaxCompt_{impul}} = (\lceil log_2(3 \times C) \rceil + \lceil log_2(C) \rceil) \times O \tag{4.3}$$

À partir de là, on peut exprimer le nombre d'impulsions maximales en entrée de cette couche pour qu'elle soit plus efficace en matière de complexité calculatoire que son équivalent formel de MaxPooling. Ainsi en utilisant les équations 4.2 et 2.8, nous obtenons la relation 4.4 qui fixe la limite en nombre d'impulsions en entrée d'une couche de MaxCompt impulsionnelle pour obtenir un gain en matière de complexité calculatoire par rapport au MaxPooling.

$$NbImp_{Pré_{Max}}(MaxCompt) = \frac{C^2 \times \lceil log_2(C) \rceil}{4 \times \lceil log_2(3 \times C) \rceil} \tag{4.4}$$

Ainsi en considèrent $C = 3 \times 3$ alors $NbImp_{Pré_{Max}}(MaxCompt) = 64$, ce qui est un gain non négligeable par rapport au MaxPooling avec $NbImp_{Pré_{Max}}(MaxPooling) = 1$. En considérant le codage fréquentiel de l'information, l'utilisation du MaxCompt permet potentiellement de réduire le nombre d'opérations à effectuer en inférence par rapport au MaxPooling formel. Cette logique de calcul du MaxCompt a fait l'objet d'un brevet [LB17].

4.5 Conclusion : La modélisation et validation

Nous avons défini une chaîne complète d'exploration et de validation générique en utilisant l'environnement N2D2 et les simulateurs matériels. Cette chaîne a été conçue pour être parcourue dans les deux sens c.-à-d. pour valider le fonctionnement de l'architecture et pour valider des choix techniques. Nous avons parcouru le détail de cette chaîne et montré sa flexibilité en matière d'exportation et de modélisation du matériel de haut niveau. Nous avons proposé notre solution de calcul approximé de l'opération de MaxPooling. Cet outil nous permet donc de valider toutes les étapes du passage du formel vers l'impulsionnel, de l'impulsionnel vers le matériel et les étapes du matériel vers les simulations impulsionnelles.

En conclusion, l'utilisation de cet outil nous permet de valider et de mesurer les performances d'une architecture à traitement unitaire des impulsions. Dans le prochain chapitre, nous présenterons le détail de notre architecture finale dont le fonctionnement a été validé par cette chaîne et nous estimerons ses performances pour nous comparer à l'état de l'art.

Chapitre 5

Architecture et performances

Contents

5.1 Introduction

Nous avons vu dans le chapitre 3 que les architectures impulsionnelles de l'état de l'art ne sont pas adaptées pour le calcul de l'inférence d'un CNN complet. Cependant, les approches spécialisées pour la convolution et le multiplexage des neurones sont prometteuses.

Dans ce chapitre, nous allons analyser comment répartir le calcul d'une couche de convolution pour maximiser le multiplexage des neurones. Cette analyse nous permettra d'introduire notre répartition des neurones et du calcul, conçu pour optimiser le nombre de calculateurs. En partant de cette répartition nous définirons la structure générale de notre architecture le NeuronSpike (NS) en décrivant les différents blocs clés de l'architecture. Pour finir cette partie théorique, nous introduirons les adaptations de notre architecture pour l'implémentation de différents types de couches et les différentes topologies de connexions entre couches et le multi-couches, nous permettant d'implémenter un CNN complet.

Puis, nous analyserons les performances du NeuronSpike, ce qui nous permettra de nous comparer aux différentes architectures impulsionnelles et formelles de l'état de l'art et ainsi de positionner et de discuter notre approche.

5.2 Stratégie de distribution des calculs

Dans l'objectif de proposer une architecture la plus compacte et optimisée possible pour le calcul des CNN, nous avons posé un certain nombre de contraintes : Premièrement, les pondérations d'une couche ne doivent pas être redondantes afin d'optimiser les besoins mémoires et potentiellement implémenter un apprentissage en ligne. Deuxièmement, pour minimiser la surface du silicium de l'architecture, il faut regrouper le plus possible de calculateurs de modèle de neurones et le plus possible de calculs sur les adresses d'impulsions. Et finalement, la partie calculatoire du modèle de neurone et les pondérations de notre architecture doivent être compatibles avec une instanciation analogique pour une potentielle utilisation de RRAM avec une implémentation faible consommation.

Dans cet objectif, nous avons effectué une étude à gros grains de l'architecture pour une couche de convolution. Cette étude va nous permettre de poser les idées de base de notre approche. Puis nous rentrerons progressivement dans le détail de notre implémentation et nous généraliserons notre approche pour différents types de couche.

5.2.1 Logique générale de distribution des calculs

Pour éviter la redondance des pondérations dans la mémoire, les pondérations devront être partagées. Une impulsion d'entrées n'active jamais deux fois la même

synapse dans les couches que l'on considère. Ainsi pour éviter la lecture concurrentielle des pondérations, nous avons opté pour une communication unitaire des impulsions de type AER, voir le détail de la communication dans la section 4.3.1.

Pour minimiser le temps de traitement d'une impulsion d'entrée, il faut pouvoir calculer la réponse à une impulsion de tous les neurones d'une même matrice de sortie parallèlement. Or, le nombre de neurones qui peuvent recevoir une impulsion d'entrée simultanément correspond à la taille de leur champ récepteur, qui correspond à la taille du filtre de convolution C. Pour faire ce calcul en parallèle, il nous faut donc au minimum C calculateurs. Lors de l'arrivée d'une autre impulsion, il est possible de réutiliser ces mêmes C calculateurs avec des valeurs différentes d'intégration, pour calculer la réponse d'autres neurones, ceci nous évite d'implémenter un calculateur par neurones. Nous noterons ce calculateur partagé entre plusieurs neurones SPE (Spike Process Element).

Ce calcul parallèle de la réponse de C neurones implique plusieurs consequences : Premièrement, la réponse des SPE à une impulsion d'entrée est simultanée, nous avons donc mis en place un sérialiseur d'impulsions de sortie notée "PE" connecté au bloc de communication AER de sortie. Deuxièmement, lorsqu'un neurone calcule sa réponse à une impulsion, il a besoin de la pondération associée à la synapse stimulée. Ainsi pour que le calcul de tous les neurones soit parallèle, il faut que toutes les pondérations soient lues parallèlement.

au regard de ces contraintes, nous nous sommes orientés vers une architecture avec deux types mémoires partagée et privée et un bloc calculatoire central avec tous les SPE et un système de contrôle global, voir la figure 5.1. La mémoire partagée est en lecture seule, elle est utilisée pour stocker les pondérations. Une ligne de cette mémoire correspond à un ou plusieurs filtres et cette ligne est accessible par tous les SPE simultanément. La mémoire privée est interne aux SPE, elle est utilisée pour stocker la valeur de l'intégration des neurones contenues dans un même SPE.

Les bases de notre architecture posée, nous allons exposer la logique de répartition et d'adressages des neurones dans les SPE. Puis nous expliquerons les extensions de cette logique pour plusieurs types de couches et plusieurs topologies de réseau.

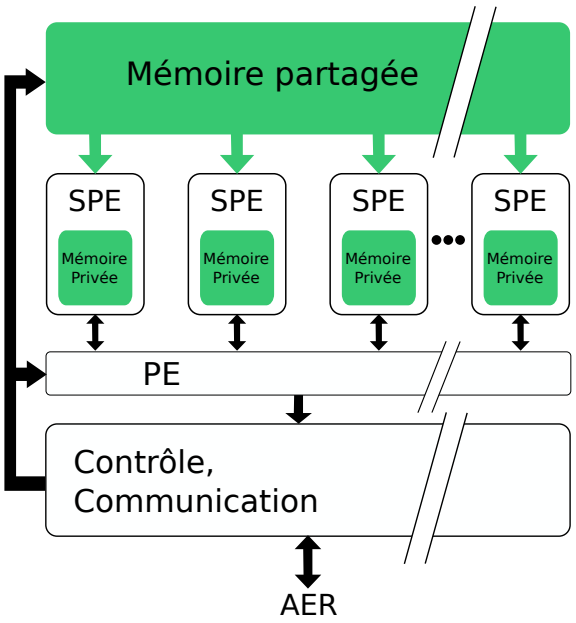

FIG. 5.1: Représentation simplifiée de l'architecture. On retrouve en vert les deux types de mémoire, une partagée et une privée. En blanc les calculateurs, le contrôle et la communication.

5.2.2 Les neurones indépendants dans une couche de convolution

Dans une convolution impulsionnelle, lors de l'arrivée d'une impulsion, C neurones sont sollicités, dans le cas le plus connecté Stride = 1. Dans une couche de convolution soumise à une impulsion à la fois, lors de la réception d'une impulsion en entrées, il existe des neurones dont le champ récepteur n'est pas connecté à cette entrée, mais également des neurones dont le champ récepteur est connecté à cette entrée. Ces deux types de neurones ne peuvent pas se déclencher en même temps. On dit de ces neurones qu'ils sont "indépendants", voir figure 5.3. Ainsi, pour des groupes de neurones indépendants d'une même matrice de sortie, il est possible de mutualiser leurs ALU (unité arithmétique et logique) de calcul du modèle IF, autrement dit on peut les placer dans le même SPE.

La topologie de notre architecture, à la différence des architectures impulsionnelles génériques ou des architectures impulsionnelles dédiées à la convolution non multiplexée, a été conçue pour profiter de cette propriété, dans le but de maximiser l'utilisation matérielle. Ainsi, nous utilisons le SPE pour regrouper plusieurs intégrations de neurones autour d'un même calculateur de modèle. Le SPE est constitué d'une unité ALU pour le calcul du modèle de neurone et d'une mémoire privée permettant de stocker la valeur interne de plusieurs neurones (l'intégration des neurones). Le SPE ne traite qu'un neurone à la fois, il y a donc autant de SPE que de pondération dans le filtre. Ceci implique aussi que tous les neurones dans un même

SPE doivent être indépendants, voir figure 5.3.

Pour que deux neurones soient indépendants, il ne faut pas que leur champ réceptif ait d'éléments communs. Dans une couche de convolution, il faut donc que les neurones soient séparés d'une distance au moins supérieure à la taille du filtre sur la matrice de sortie. Nous avons donc introduit une fonction de partitionnement en neurones indépendants (5.1). Cette fonction définit la relation entre un neurone,= identifié par sa position sur la matrice de sortie (o_x, o_y) et les numéros du SPE dans lequel il est placé pour que les neurones du même SPE soitent indépendants.

$$\begin{aligned} SPE_x &= o_x \mod C_{elmX} \\ SPE_y &= o_y \mod C_{elmY} \\ SPE &= SPE_x + C_{elmX} \times SPE_y \end{aligned} \tag{5.1}$$

Il est possible de visualiser le numéro du SPE associé au neurone sur la matrice de sortie, voir la figure 5.2. Toutes les coordonnées dans la matrice de sortie correspondent à un neurone, la valeur associée à un élément de la matrice quant à lui correspond aux numéros du SPE dans lequel le neurone est placé.

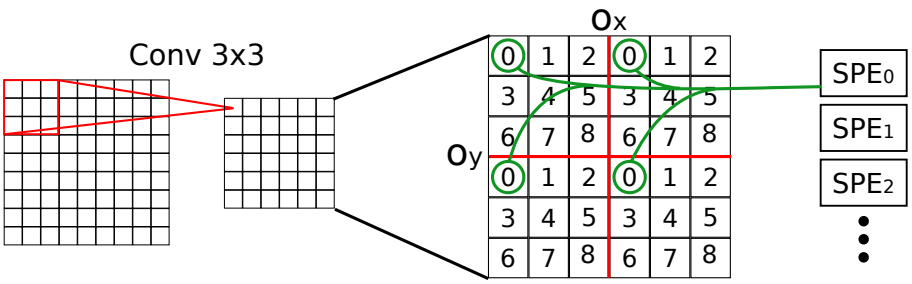

FIG. 5.2: Répartition des SPE sur la matrice de sortie avec $C_{elmX} = C_{elmY} = 3$ et $O_{elmX} = O_{elmY} = 6$.

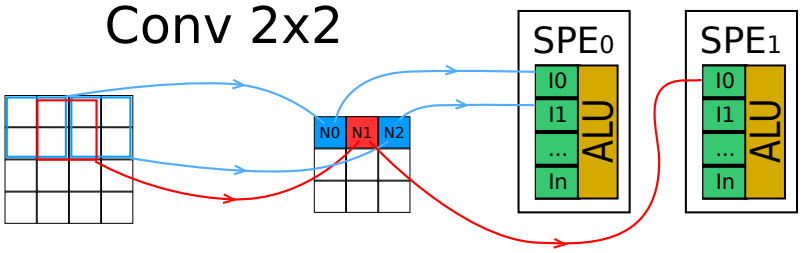

FIG. 5.3: Les neurones indépendants. Le champ récepteur des neurones bleus ne se chevauche pas, ces neurones sont indépendants, ils sont placés dans le même SPE. Le champ récepteur du neurone en rouge chevauche les champs des deux neurones en bleu, il y a donc une dépendance avec les neurones bleus, le neurone rouge doit être placé dans un autre SPE.

Cette notion de neurones indépendants nous permet de réduire le nombre de

calculateurs nécessaires et de réutiliser une grande partie du matériel entre deux impulsions. Cependant, cette répartition a des conséquences sur l'adressage des pondérations et sur celle des intégrations. Nous allons étudier l'impact de cette logique de répartition sur le calcul des adresses mémoires.

5.2.3 L'adressage des mémoires de pondérations et d'intégrations

Adressage des pondérations

Dans une couche de convolution, les neurones partagent les mêmes pondérations. Ces pondérations correspondent aux valeurs du filtre de convolution. Lors de l'arrivée d'une impulsion, ce sont toutes les pondérations du filtre qui sont lues et distribuées aux neurones, à raison d'une pondération par neurone. Pour une impulsion d'entrée, de coordonnée (i_x, i_y), et pour $s_x \in [0, C_{elmX}[$ et $s_y \in [0, C_{elmY}[$ coordonnées de la pondération dans le filtre. L'équation (5.2) donne la relation entre un neurone (o_x, o_y) et une pondération d'adresse $@W$ en fonction d'une impulsion d'entrées de coordonné (i_x, i_y) et en considérant les pondérations alignées en haut à gauche.

$$\begin{cases} i_x - s_x = o_x \\ i_y - s_y = o_y \\ @W = C_{elmX} \times s_y + s_x \end{cases} \tag{5.2}$$

Pour implémenter cet accès mémoire, il nous est nécessaire d'étudier la relation entre les pondérations, l'impulsion en entrée et les SPE. Pour exprimer cette relation, nous avons utilisé les équations (5.2) pour exprimer s_x et s_y, voir l'exemple pour s_x :

$$s_x = i_x - o_x$$

Et nous utilisons l'équation (5.1) pour exprimé s_x en fonction de SPE_x et SPE_y, voir l'exemple pour SPE_x :

$$s_x \mod C_{elmX} = s_x = (i_x - o_x) \mod C_{elmX} \text{ car } 0 \leq s_x \leq C_{elmX} - 1$$

Ainsi nous avons une nouvelle expression de s_x :

$$s_x = (i_x - o_x) \mod C_{elmX} = (i_x \mod C_{elmX} - o_x \mod C_{elmX}) \mod C_{elmX}$$

Or, $SPE_x \mod C_{elmX} = SPE_x$ car $0 \leq SPE_x \leq C_{elmX} - 1$, ainsi $s_x = (i_x \mod C_{elmX} - SPE_x) \mod C_{elmX}$. De plus, $i_x \mod C_{elmX} - SPE_x \geq 1 - C_{elmX}$ donc au final on obtient l'équation (5.3) pour s_x et s_y.

$$
\begin{aligned}
&@W = C_{elmX} \times s_y + s_x \\
&s_x = \begin{cases} (i_x \mod C_{elmX}) - SPE_x \text{ si } i_x \mod C_{elmX} \geq SPE_x \\ (i_x \mod C_{elmX}) - SPE_x + C_{elmX} \text{ sinon} \end{cases} \\
&s_y = \begin{cases} (i_y \mod C_{elmY}) - SPE_y \text{ si } i_y \mod C_{elmY} \geq SPE_y \\ (i_y \mod C_{elmY}) - SPE_y + C_{elmY} \text{ sinon} \end{cases}
\end{aligned} \tag{5.3}
$$

On remarque que l'on peut isoler le traitement sur (i_x, i_y) de l'adresse du SPE. Ainsi, il nous est possible de regrouper le calcul de $i_x \mod C_{elmX}, i_y \mod C_{elmY}$ pour tous les SPE. Seule la soustraction en fonction des coordonnées du SPE et la comparaison sont à la charge du SPE. Ce regroupement des fonctions d'adressage pour tous les SPE suit la logique de minimisation de la surface du silicium.

Dans l'architecture, la taille du filtre est fixée et elle correspond à la taille du filtre maximal de $C_{Max} = C_{MaxX} \times C_{MaxY}$ éléments. Le nombre d'éléments dans le filtre de dimension maximale correspond au nombre de SPE implémenté dans le système. En analysant l'état de l'art des réseaux de neurones convolutionnels : GoogLeNet [67], VGG [13], SqueezeNet [68], AlexNet [14], il nous a semblé que 11×11 est une valeur judicieuse pour $C_{MaxX} \times C_{MaxY}$, par la suite nous considérerons donc cette valeur. Mais l'évolution récente des DNNs montre que les filtres sont souvent limités à 1×1 et 3×3, ce qui permet de rendre le coût de cette logique d'adressage quasiment nul.

Comme la mémoire des pondérations contenant les filtres doit être adressable en parallèle, nous avons choisi de la représenter sous forme d'un vecteur. Ce vecteur de $C_{MaxX} \times C_{MaxY}$ éléments est la concaténation des lignes de la matrice du filtre maximal, voir la figure 5.4 pour une représentation simplifiée avec $C_{MaxX} \times C_{MaxY} = 4 \times 4$.

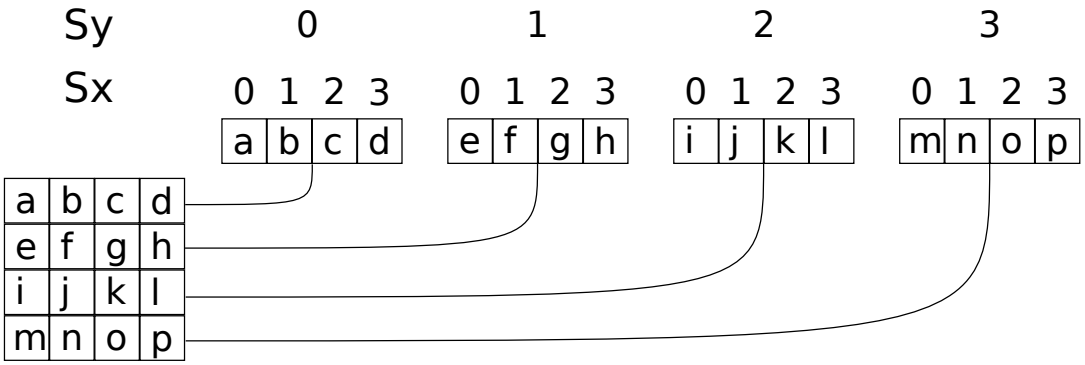

FIG. 5.4: Représentation en vecteur du filtre de dimension maximale $C_{MaxX} = C_{MaxY} = 4$. Chaque ligne du filtre est concaténée pour former un vecteur.

Pour une utilisation optimale des ressources, il est possible de stocker plusieurs

filtres dans un vecteur de mémoire de pondération. Dans ce cas, les filtres d'un même vecteur ont la même taille et sont soumis aux mêmes impulsions d'entrées. Les SPE sont alors utilisés pour calculer des matrices de sortie différentes.

Le nombre de filtres implémentables dans un vecteur est égal à : $Nb_{filtre/vec} = \lfloor \frac{C_{MaxX}}{C_{elmX}} \rfloor \times \lfloor \frac{C_{MaxY}}{C_{elmY}} \rfloor$ Les différents filtres sont alors concaténés ligne par ligne dans le vecteur de pondération. L'équation (5.3) est alors appliquée indépendamment pour chaque filtre.

Adressage des intégrations

Dans la logique de répartition en neurones indépendants, ces derniers sont distribués dans les SPE. Il faut donc déterminer en fonction de cette logique de distribution, leur logique d'adressage de leurs intégrations d'adresse $@I$ dans les SPE. Ces deux intégrations dans deux SPE peuvent avoir la même adresse. C'est donc le couple $(@I, SPE)$ qui représente alors le numéro du neurone qui est unique pour chaque neurone d'une couche. En sachant que le numéro de SPE est défini par l'équation (5.1), nous avons utilisé l'équation (5.4) pour définir un couple $(@I, SPE)$.

$$@I = (\lfloor \frac{o_y}{C_{elmY}} \rfloor, \lfloor \frac{o_x}{C_{elmX}} \rfloor) \tag{5.4}$$

Cette équation distribue les intégrations en fonction des neurones de sortie. On obtient alors un découpage de la matrice de sortie, représentée dans la figure 5.5, chaque élément représente une $@I$ du neurone. (o_x, o_y)

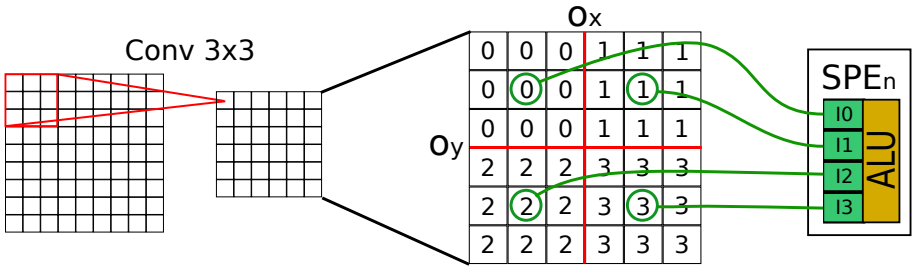

FIG. 5.5: Répartition des intégrations sur une matrice de sortie avec $C_{elmX} = C_{elmY} = 3$ et $O_{elmX} = O_{elmY} = 6$.

Pour l'intégration dans notre architecture, il faut exprimer cette équation en fonction de (i_x, i_y) et de (SPE_x, SPE_y). Pour commencer, nous partons de l'équation (5.3) et (5.2) pour exprimer o_x. Le calcul avec o_y est identique.

$$o_x = SPE_x + C_{elmX} \lfloor \frac{i_x - SPE_x}{C_{elmX}} \rfloor$$

Or, $\frac{SPE_x}{C_{elmX}} < 0$, donc $\frac{i_x - SPE_x}{C_{elmX}} = \frac{i_x}{C_{elmX}}$ si $frac(\frac{i_x}{C_{elmX}}) \geq \frac{SPE_x}{C_{elmX}}$ sinon $\frac{i_x - SPE_x}{C_{elmX}} = \frac{i_x}{C_{elmX}} - 1$. Or, $frac(\frac{i_x}{C_{elmX}}) \geq \frac{SPE_x}{C_{elmX}}$ si $i_x \mod C_{elmX} \geq SPE_x$, ce qui nous donne l'équation (5.5).

$$o_x = \begin{cases} SPE_x + C_{elmX}\lfloor\frac{i_x}{C_{elmX}}\rfloor & \text{si } i_x \mod C_{elmX} \geq SPE_x \\ SPE_x + C_{elmX}(\lfloor\frac{i_x}{C_{elmX}}\rfloor - 1) & \text{sinon} \end{cases} \tag{5.5}$$

Pour $@I$ on exprime alors $\lfloor\frac{o_x}{C_{elmX}}\rfloor$ en fonction de i_x et de SPE_x. Or, $C_{elmX} > SPE_x \geq 0$ et $i_x \geq 0$ alors ce qui nous donne pour $\lfloor\frac{o_x}{C_{elmX}}\rfloor$ l'équation (5.6).

$$\lfloor\frac{o_x}{C_{elmX}}\rfloor = \begin{cases} \lfloor\frac{i_x}{C_{elmX}}\rfloor & \text{si } i_x \mod C_{elmX} \geq SPE_x \\ \lfloor\frac{i_x}{C_{elmX}}\rfloor - 1 & \text{sinon} \end{cases} \tag{5.6}$$

On peut donc exprimer $@I$ en fonction de (SPE_x, SPE_y) et (i_x, i_y) :

$$\begin{aligned} @I &= I_x + \lfloor\frac{O_{elmX}}{C_{elmX}}\rfloor I_y \\ I_x &= \begin{cases} \lfloor\frac{i_x}{C_{elmX}}\rfloor \text{ si } i_x \mod C_{elmX} \geq SPE_x \\ \lfloor\frac{i_x}{C_{elmX}}\rfloor - 1 \text{ sinon} \end{cases} \\ I_y &= \begin{cases} \lfloor\frac{i_y}{C_{elmY}}\rfloor \text{ si } i_y \mod C_{elmY} \geq SPE_y \\ \lfloor\frac{i_y}{C_{elmY}}\rfloor - 1 \text{ sinon} \end{cases} \end{aligned} \tag{5.7}$$

On remarque que pour une impulsion d'entrée, il n'existe que 4 possibilités de $@I$ pour tous les SPE. Cette propriété nous permet donc d'envisager de regrouper le calcul de $@I$ pour tous les SPE. Les SPE sélectionneront celle qui leur convient en fonction de leur SPE_x, SPE_y et de la valeur de $i_x \mod C_{elmX}, i_y \mod C_{elmY}$, voir figure 5.6.

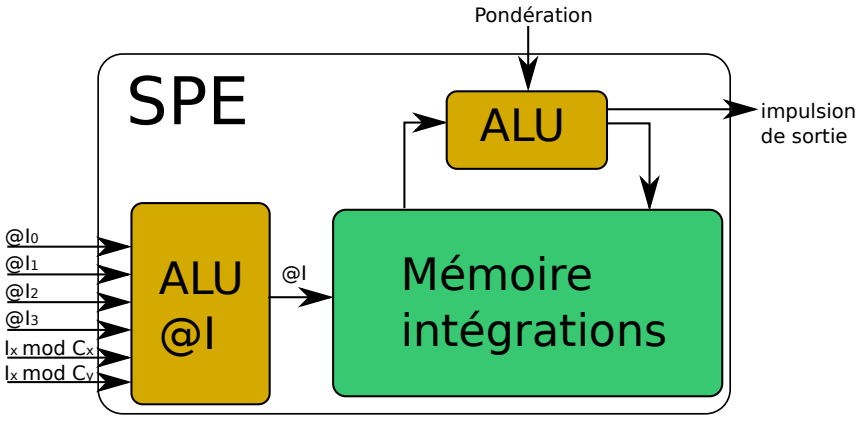

FIG. 5.6: Représentation schématique d'un SPE avec ses blocs logiques en jaune et de sa mémoire privée d'intégrations en vert. Le bloc ALU $@I$ est utilisé pour adressé l'intégration et le bloc ALU fait le calcule de la réponse d'un neurone IF.

Conclusion des adressages

Comme nous avons pu le voir, le calcul d'adresse dans un système à neurones indépendants est plus complexe là où classiquement la relation entre l'adresse d'un neurone et d'une pondération est déterminée par une opération de soustraction, voir équation (5.2). Cependant, de nombreuses fonctions d'adressage peuvent être regroupées pour tous les SPE. Ce regroupement nous permet d'éviter la redondance d'une partie des fonctions d'adressage dans les SPE, et ainsi minimise la surface du silicium.

Nous avons donc défini le bloc de pré-calcul utilisé pour fournir à tous les SPE les 4 $@I$ possibles et les valeurs de $i_x \mod C_{elmX}, i_y \mod C_{elmY}$, voir la figure 5.7. Les SPE sélectionnent une des 4 $@I$ en fonction de leur coordonnée et de la valeur du modulo. La distribution des pondérations est faite directement par une structure connectant les pondérations au SPE, voir la section 5.3.1.

En conclusion, la notion de neurone indépendant nous permet de mutualiser une grande partie du matériel, calculer la réponse d'une impulsion d'entrée en parallèle et de décorréler le nombre de calculateurs et le nombre du neurone implémentables.

Cette logique d'implémentation d'une opération de convolution est donc pertinente dans le cadre de la réduction du coût d'implémentation et l'optimisation des performances de calcul. Nous allons maintenant nous intéresser à la généralisation de cette architecture pour différente Strides et couches.

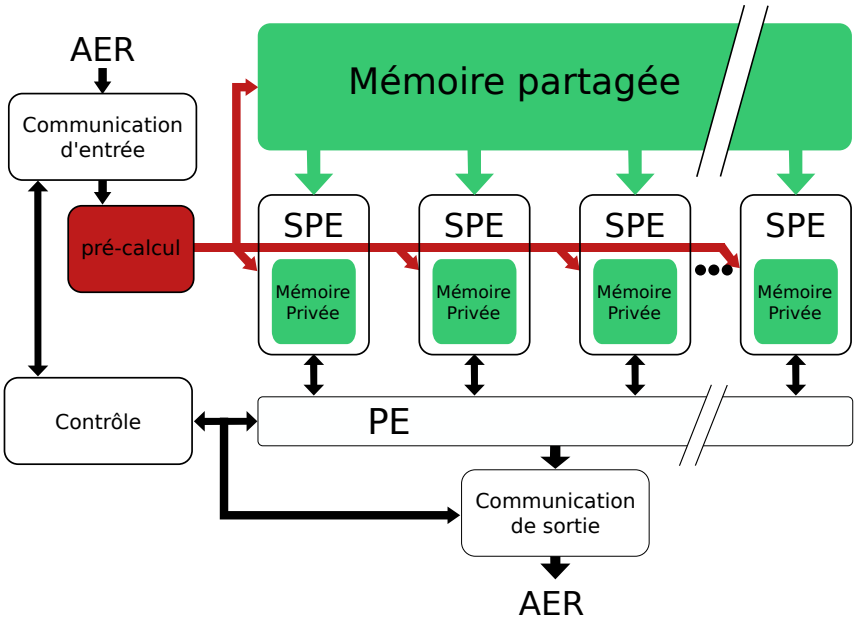

FIG. 5.7: Représentation en bloc de l'architecture avec la représentation du pré-calcul et du bus d'adressage (en rouge).

5.2.4 Les extensions de fonctionnalité : Stride, FC et Max-Pooling

Nous allons étudier les conditions d'implémentation du Stride et des couches FC et MaxPooling dans notre architecture. Nous exposerons dans cette section les méthodes d'instanciation de ces généralisations leurs limites, leurs conditions d'implémentation.

Stride

Le Stride permet d'appliquer les filtres sur une matrice d'entrée avec un pas différent de 1, voir section 2.2.2 . Ce décalage peut se faire sur x et/ou sur y d'un nombre d'éléments notés (St_x, Sty). Il est possible de générer une matrice de sortie avec un Stride quelconque en utilisant la matrice de sortie avec un Stride de 1, voir figure 5.8. Pour cela, il faut trier les impulsions de sortie et recalculer leurs adresses, voir équation (5.8).

$$\begin{cases} new(o_x) & = \lfloor \frac{o_x}{St_x} \rfloor \text{ si } o_x \mod St_x = 0 \\ new(o_y) & = \lfloor \frac{o_y}{St_y} \rfloor \text{ si } o_y \mod St_y = 0 \end{cases} \tag{5.8}$$

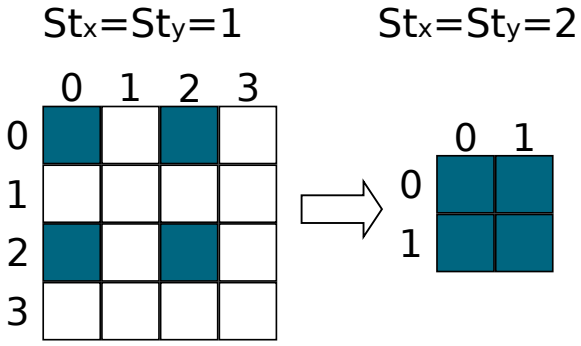

FIG. 5.8: Passage d'une matrice de sortie avec $St_x = St_y = 1$ à sa version $St_x = St_y = 2$. Les éléments en bleus représentent les éléments de la matrice avec $St_x = St_y = 1$ concaténés pour former la matrice avec $St_x = St_y = 2$.

Une première solution est de n'utiliser que les SPE ne contenant que des neurones de sorties utiles après application du Stride. Pour cela il nous faut une relation entres (o_x, o_y) et (SPE_x, SPE_x) indépendante des entrées (constante) dans le but d'identifier les SPE utiles. Or, si l'on prend le cas du o_x et en utilisant les équations 5.8 et 5.6 on obtient l'équation 5.9.

$$\lfloor \frac{o_x}{C_{elmX}} \rfloor = \begin{cases} \lfloor \frac{i_x}{C_{elmX}} \rfloor \mod St_x & \text{si } i_x \mod C_{elmX} \geq SPE_x \\ \lfloor \frac{i_x}{C_{elmX}} \rfloor - 1 \mod St_x & \text{sinon} \end{cases} \tag{5.9}$$

On remarque, que dans le cas ou $St_x \neq C_{elmX}$ la relation entre o_x et SPE_x dépend de i_x. Avec notre stratégie de répartition en neurone indépendant on ne peut pas isoler les SPE inutiles après application du Stride. Pour appliquer cette logique de calcul du Stride, il faut donc repenser la logique de répartition des neurones dans les calculateurs, ce qui impacte toute la structure de l'architecture.

Une autre stratégie consiste à trier les neurones utiles dans les SPE. Au vu de la théorie du calcul du Stride, deux choix de logique de calcul du Stride sont envisageables dans notre architecture :

Un premier choix de calculateur pour le calcul du Stride commun à tous les SPE à la suite du PE, qui va ainsi limiter le coût matériel du Stride. Ce choix engendre potentiellement un surcoût calculatoire et un surcoût de sérialisation des impulsions de sortie. En effet tous les SPE vont effectuer le calcul puis le PE va sérialiser les impulsions de sortie, mais seule une partie va finalement être prise en compte.

Un second choix qui évite le calcul et la propagation dans le PE d'impulsions inutiles consiste à instancier ce calculateur de Stride dans chaque SPE. Ce calculateur va inhiber le SPE si $o_x \mod St_x \neq 0$ et $o_y \mod St_y \neq 0$, évitant ainsi le calcul et le déclenchement inutile des neurones. Mais la répétition de ces deux opérateurs mod dans tous les SPE représente un fort coût matériel.

Sachant que dans les CNN, la valeur du Stride reste limitée pour les couches de convolution (rarement au-dessus de 2), que le Stride n'est pas utilisé pour les couches de FC et que seules les couches de MaxPooling l'utilisent avec un Stride classiquement égal à la taille du filtre. Sachant que le surcoût de la sérialisation d'impulsions inutiles est limité par le fait qu'il y a peu de neurones dans une matrice de sortie qui se déclenchent simultanément. La solution du calculateur commun m'a paru pertinente, car elle permet d'optimiser le coût matériel du calcul du Stride. Pour implémenter cette fonction on ajoute un calculateur à la suite du PE, que l'on note post-calcul et qui est utilisé pour modifier et trier les adresses d'impulsion avant l'envoi, voir la figure 5.9.

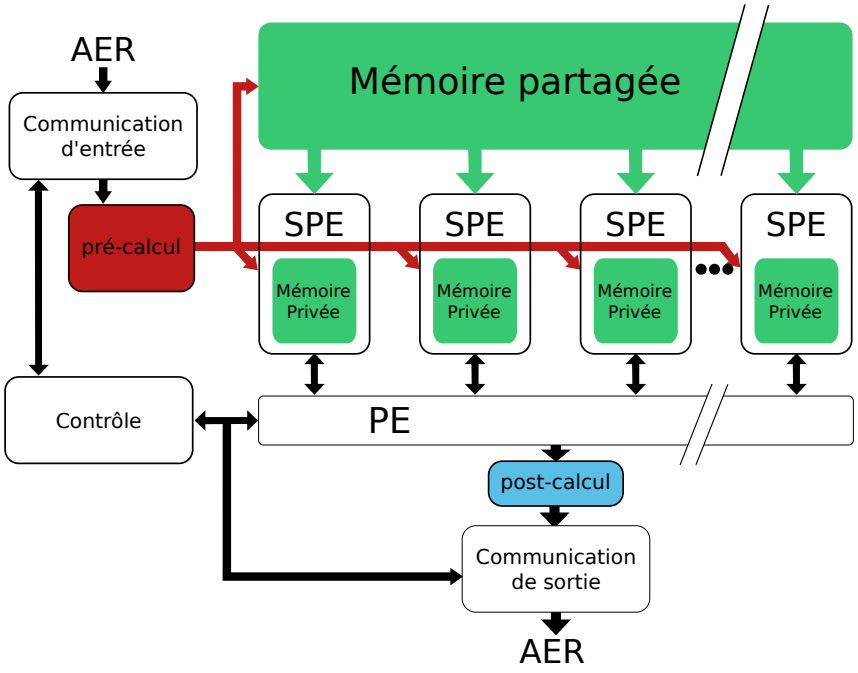

FIG. 5.9: Représentation en bloc de l'architecture avec la représentation du post-calcul en bleu à la suite du PE, avant la communication de sortie. Le post-calcul est utilisé pour effectuer le Stride.

Le cas de la couche entièrement connectée (FC)

Classiquement, les structures CNN se terminent par une ou plusieurs couches entièrement connectées (FC, fully connected). Dans une couche de FC, les neurones sont connectés à toutes les entrées.

La connectivité d'un neurone FC peut être vue comme une convolution dont la taille du filtre est égale à la taille de la matrice d'entrée voir la figure 5.10. Dans le cas de la première couche FC, le filtre doit être égal à la taille des matrices d'entrées. Or, la structure de l'architecture impose un nombre fixe de SPE, égal au nombre maximal de pondérations du filtre C_{Max}. Il y a donc limitation matérielle au niveau de la taille des matrices d'entrées du FC qui doit être au maximum égale à la taille du filtre maximal soit 11×11. Dans le cas des couches FC, après la première couche FC, la taille des matrices d'entrée de ces couches est de 1×1. Ces couches sont donc implémentable dans l'architecture.

Malgré les imitations matérielles de la première couche de FC sur la matrice d'entrée qui doit être d'au maximum de 11×11, la couche de FC est implémentable dans notre architecture. Une étude de la répartition de la première couche de FC sur plusieurs architectures dans le but de dépasser cette limite de taille de matrice d'entrée de 11×11, est envisagée pour une future optimisation de l'architecture.

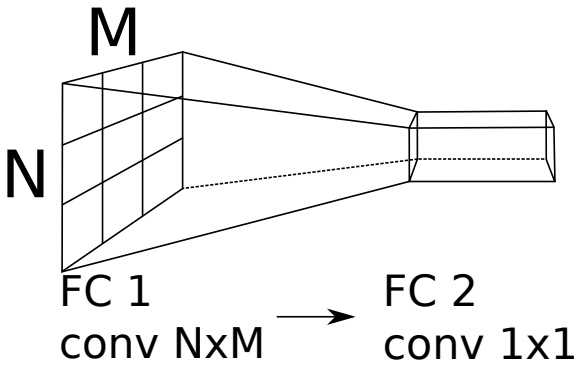

FIG. 5.10: Représentation des deux couches de FC qui se trouvent à la fin de la majorité des CNN. La première couche de FC permet de passer d'une matrice 2D à un élément, elle peut être vue comme une convolution dont la taille est égale à la taille de l'image d'entrée $N \times M$. La deuxième couche de FC est assimilable à une convolution 1×1.

Le cas de la couche de MaxPooling

Un neurone de MaxPooling détermine sa synapse la plus active à chaque nouvelle impulsion d'entrée, la fonction de MaxPooling est donc différente de la fonction de la couche de convolution, mais sa connectivité à la couche précédente est semblable, nous devons donc seulement modifier l'ALU dans le SPE pour effectuer un MaxPooling.

Comme nous l'avons vu dans la section 4.4, la fonction de MaxPooling est couteuse en termes de mémoire et d'opération. C'est pour cela que nous avons mis en place une approximation, le MaxCompt. Pour rappel, nous utilisons une activité de synapse différentielle notée A. Cette activité est une activité différentielle entre l'adresse de la synapse Max, notée $@Smax$ et toutes les autres synapses. Lors de la réception d'une impulsion, l'adresse de la synapse excitée par cette impulsion $@S$ est comparée à $@Smax$. Si $@S = @Smax$ alors $A = A + 1$ et le neurone se déclenche. Sinon $@S \neq @S_{max}$ et si $A > 0$ alors $A = A - 1$ sinon $A = Ref$ et $@Smax = @S$. Avec Ref une valeur fixée à la programmation dont la valeur minimale est fonction du nombre d'éléments dans le champ récepteur, a été fixée à $3 \times C$ après de nombreuses simulations sur les réseaux classiques.

Nous allons donc voir les conditions d'implémentation du MaxCompt dans un SPE, pour le cas du filtre maximal C_{Max}. Premièrement, MaxCompt ne nécessite qu'une seule mémoire dont la dimension est égale à $\lceil log_2(C_{Max}) \rceil + \lceil log_2(3 \times C_{Max}) \rceil$ bits, avec $\lceil log_2(C_{Max}) \rceil$ bits pour stoker l'adresse de la synapse la plus active et $\lceil log_2(3 \times C_{Max}) \rceil$ bits pour le compteur d'activité. En considérant un filtre max de 11×11, le besoin du MaxCompt et de $\lceil log_2(121) \rceil + \lceil log_2(3 \times 121) \rceil = 16$ bits. Or, la précision choisie pour les pondérations Nbi_{bit} est de 8 bits. Donc la taille de la mémoire d'intégration $2 \times Nbi_{bit}$ est alors de 16 bits, voir la relation entre la précision des pondérations et la précision des intégrations dans la section 2.4.3. Le MaxCompt

est donc compatible avec la quantité de mémoire disponible (d'intégration) dans un SPE pour un neurone.

Deuxièmement, la détermination de @S. Dans notre architecture les SPE ne reçoivent que l'adresse d'intégration à utiliser et la pondération à intégrer, ils n'ont donc pas connaissance de l'adresse de la synapse stimulée par l'impulsion d'entrée @S , essentielle pour le MaxCompt. Cependant, cette adresse de la synapse @S est liée à la pondération reçue par un SPE. En initialisant le filtre avec des valeurs de pondérations toutes différentes, il est donc possible d'identifier @S. Ainsi, la valeur de la pondération est considérée ici comme l'adresse de la synapse @S stimulée par une impulsion en entrée. Pour ce faire, il faut que les précisions des pondérations soient suffisantes pour que chaque élément du filtre soit différent. En considérant un filtre max de 11×11, une précision de pondération sur 8 bits et comme $2^8 > 121$, cette condition est satisfaite.

Pour l'implémentation du MaxCompte dans un SPE, on ajout le modèle à l'ALU du SPE et un signal Sel permet de choisir entre un neurone IF et un neurone MaxCompt voir figure 5.11.

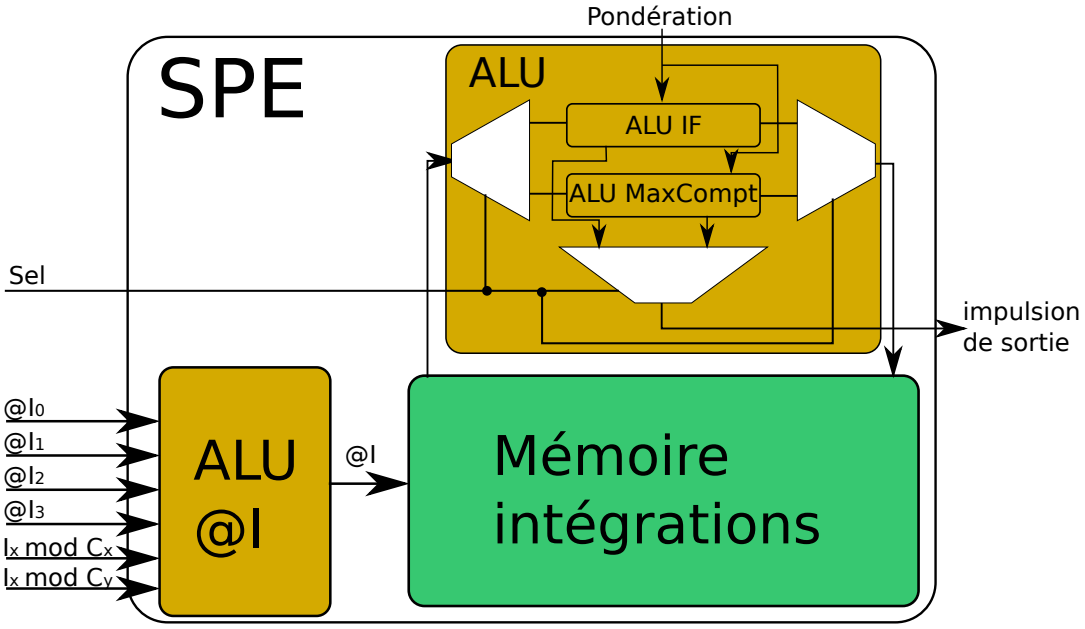

FIG. 5.11: Représentation schématique d'un SPE avec le modèle de neurone IF et le modèle de neurone MaxCompt, les blocs logiques sont en jaune et de la mémoire privée d'intégration en vert.

Au final, il est possible d'instancier des couches de MaxCompte dans notre architecture, de plus, cette implémentation ne nécessite pas plus de mémoire ni un autre adressage qu'un neurone IF.

Conclusion des extensions

Pour le calcul d'un CNN, il est nécessaire de pouvoir alterner différents types de couches. Comme nous avons vu, il est possible d'adapter le calculateur pour le calcul de différentes couches. Ce MaxCompt ne nécessite que quelques adaptations, mais ne rentre pas en conflit avec la notion de neurone indépendant ni avec la structure de base du calculateur. Pour conclure, notre approche de la répartition des neurones est assez flexible pour l'implémentation des différents modèles de couches des CNN. Nous allons nous intéresser maintenant à la réalisation de différentes topologies de connexion et au multi-couche.

5.2.5 Le passage à l'échelle mutli-filtres, multi-couches

mutli-filtres

Les mémoires des pondérations contiennent plusieurs vecteurs de pondération, on note @w l'adresse du vecteur. Chaque vecteur de pondération contient un ou plusieurs filtres qui s'appliquent sur la même matrice d'entrée. Cependant, si le nombre de filtres connectés à une même entrée est supérieur aux capacités d'un vecteur de pondérations ($Nb_{filtre} > Nb_{filtre/vec}$), le calcul se fait séquentiellement. Pour ce faire, on multiplexe temporellement le calcul, l'impulsion d'entrée de coordonnées (i_x, i_y, i_z) est découpée en plusieurs impulsions avec des coordonnées de profondeurs i_z croissantes, voir figure 5.12. Le nombre de découpes est égal à $Nb_{dec} = \lceil \frac{Nb_{filtre}}{Nb_{filtre/vec}} \rceil$, avec Nd le numéro de la découpe. Le système génère aussi un offset des mémoires d'intégration pour adresser les bonnes couches de sortie. Pour cela, on calcule le nombre de neurones d'une couche par SPE avec :

$$Max_{@I} = \lceil \frac{I_{elmX} - C_{elmX} + 1}{C_{elmX}} \rceil \times \lceil \frac{I_{elmY} - C_{elmY} + 1}{C_{elmY}} \rceil$$

Le numérateur correspond au nombre de neurones total dans la couche sans Stride et le dénominateur correspond au nombre de SPE dans lesquels les neurones sont répartis. On définit alors l'offset comme étant $I_{offset} = Nd \times Max_{@I}$.

Dans le cas où il y a plusieurs matrices d'entrée sur lesquelles sont appliqués plusieurs filtres, la logique est identique. Les différentes couches d'entrées sont découpées. Cependant, pour éviter le recouvrement, une variable d'offset en profondeur W_{offset} est utilisée. La valeur de cet offset peut être précalculée avec $W_{offset} = Nb_{dec} \times N_{couche}$ et avec N_{couche} le numéro de la couche d'origine, voir figure 5.13 pour un exemple de découpe multi-entrées.

Dans le cas de couches partiellement connectées, nous générons une connexion complète dans laquelle certains filtres sont à 0. Cette solution impacte peu la consommation de l'architecture, car un SPE recevant une pondération à 0 n'est pas activé. Cependant, cette solution est sous-optimale du point de vue de la mémoire, mais

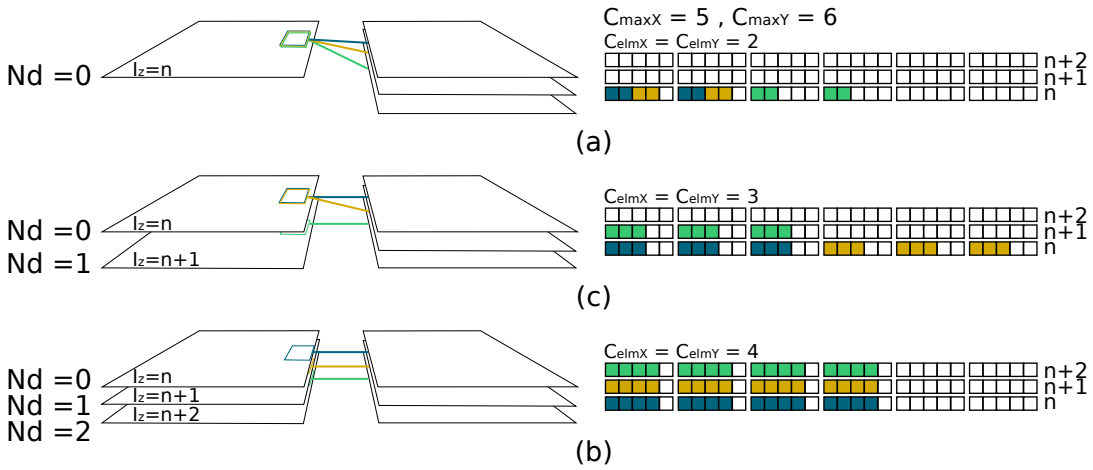

FIG. 5.12: Répartition de plusieurs filtres dans un vecteur de pondération. (a) Connexion initiale. (b) cas partiellement parallèle $Nb_{filtre/vec} > 1$. (c) cas séquentiel $Nb_{filtre/vec} = 1$. C_{elmX} et C_{elmY} la taille des filtres considérés et C_{maxX} et C_{maxY} la taille du filtre maximal

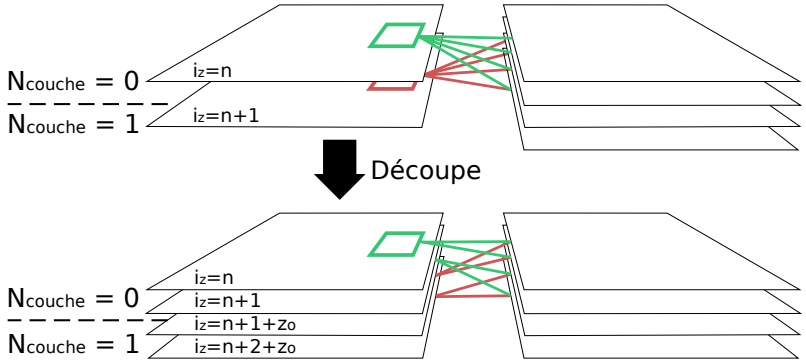

FIG. 5.13: Cas d'une entrée multiple avec découpe. Les filtres sont appliqués sur les deux matrices d'entrées. Celle-ci étant trop grandes pour être stockées dans un seul vecteur, on découpe ces matrices d'entrées afin d'utiliser plus de vecteurs de pondération.

dans le cadre d'une optimisation future il est envisagé d'étudier la concaténation de différents filtres de différentes couches dans un même vecteur de pondérations.

Le système présenté nous permet de créer une connexion complète entre deux couches quelconques du réseau. Le principe de mutli-filtres est compatible avec les autres modèles de couches : FC, MaxPooling. Ce qui nous permet par exemple de calculer un nombre de neurones quelconque dans les couches FC. Cette découpe d'une impulsion en plusieurs impulsions est effectuée par un bloc noté "découpe" qui se situe à la suite du bloc de communication d'entrée, voir figure 5.14.

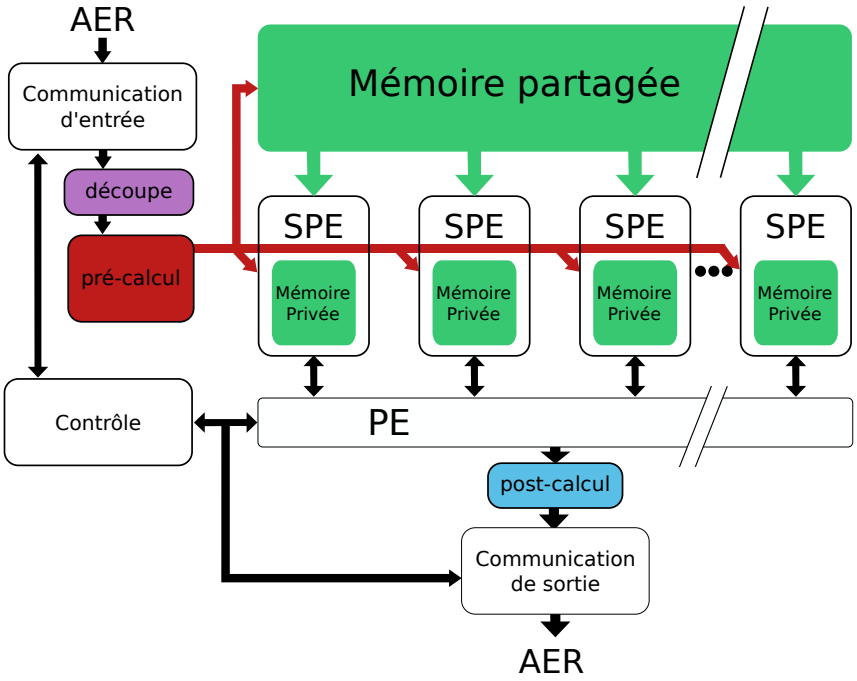

FIG. 5.14: Représentation en bloc de l'architecture avec la représentation du bloc de découpe en violet à la suite du bloc de communication de sortie. Le bloc de découpe est utilisé pour effectuer le mutli-filtres.

multi-couches

Après avoir généralisé les connexions entre deux couches quelconques, nous allons expliquer le principe du multi-couches. Pour le calcul multi-couches, l'idée est de reboucler les impulsions dans l'architecture. Nous avons donc ajouté une coordonnée aux impulsions, la coordonnée de couche i_c. Cette nouvelle coordonnée est interne au système et n'est utilisée que pour le rebouclage. Les impulsions rebouclées sont stockées dans une FIFO spécifique qui permet à notre architecture de continuer à lire les impulsions en entrées tout en traitant une impulsion rebouclée.

Ainsi, le système a deux possibilités d'entrée : une impulsion extérieure ou une impulsion rebouclée. Nous avons choisi de prioriser les impulsions rebouclées pour plusieurs raisons. Premièrement, nous voulons optimiser le temps d'inférence, pour cela, les impulsions les plus avancées dans le réseau, donc les plus proches de la sortie, sont les plus importantes. Deuxièmement, pour éviter l'encombrement, si l'activité en entrée est trop importante, le système ne pourra pas stocker toutes les impulsions à reboucler.

La coordonnée $i_c \in [0 : La]$ est ajoutée aux impulsions d'entrées, cette coordonnée i_c est initialisée à 0, les impulsions de sortie sont celles avec $i_c = La$. À chaque passage dans l'architecture, la coordonnée de couche de l'impulsion est modifiée.

La relation entre la nouvelle et l'ancienne coordonnée de couche est définie dans une table de correspondance (Lookup table, LUT) initialisée lors de la program-

mation. Cette coordonnée de couche est utilisée pour appliquer des offsets sur les mémoires des pondérations et des intégrations. Elle est aussi utilisée pour charger une nouvelle configuration de couche, filtres et caractéristique de couche.

Nous avons donc un nouvel offset de pondération $W_{offsetCouche}$ et un nouvel offset pour les intégrations $I_{offsetCouche}$. Les valeurs de $W_{offsetCouche}$ et $I_{offsetCouche}$ sont respectivement la somme des W_{offset} et I_{offset} des couches précédentes plus 1. La répartition d'un réseau en mémoire ainsi que les différents offset pour un réseau de deux couches sont présentés figure 5.15.

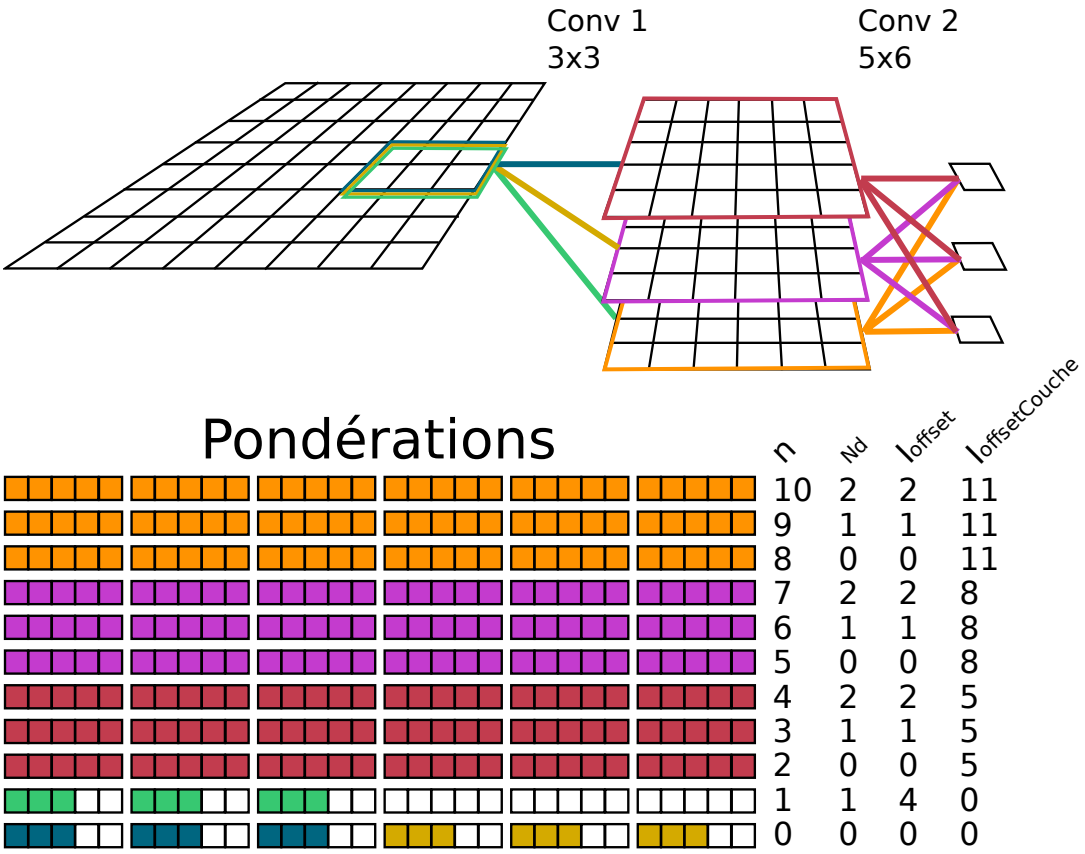

FIG. 5.15: Exemple d'instanciation d'un réseau et représentation de la mémoire des pondérations associées pour $C_{MaxX} = 5$ et $C_{MaxY} = 6$. On remarque que les valeurs des filtres sont stockées à des profondeurs n différentes et les offsets permettent d'adresser les bons filtres en fonction de la couche en cours de calcul.

Au niveau de l'architecture, cette fonction impose un rebouclage entre le bloc de communication d'entrée et le bloc de communication de sortie. La figure 5.16 est une représentation en bloc de l'architecture avec la boucle des impulsions. Dans cette figure sont représentés tous les blocs calculatoires de l'architecture.

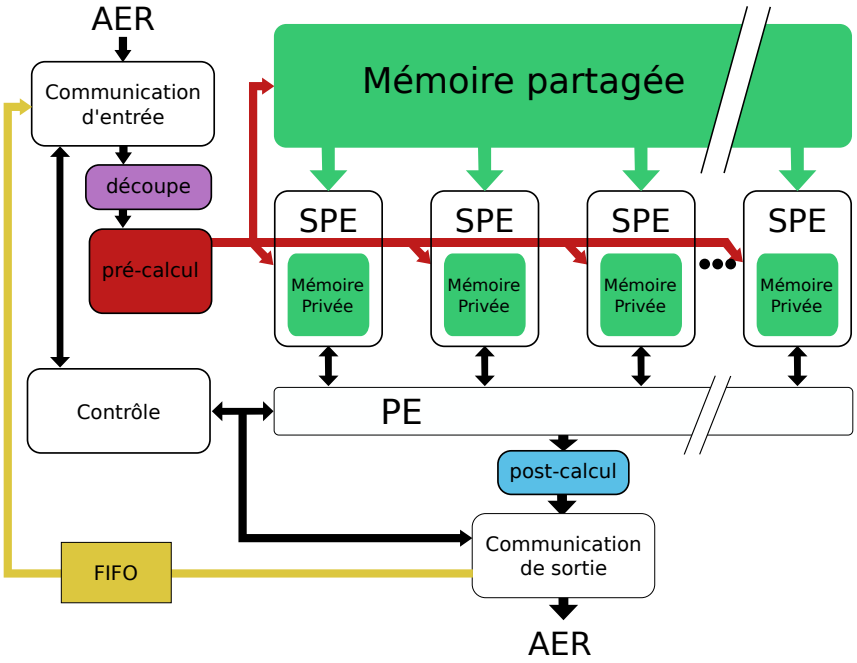

FIG. 5.16: Représentation en bloc de l'architecture avec le rebouclage en jaune, cette représentation correspond à l'architecture finale avec tous ses blocs de calcul.

Conclusion du passage à l'échelle

Notre architecture est donc capable d'utiliser les mémoires de pondérations et d'intégrations pour implémenter n'importe quelle topologie de CNN. La notion de neurone indépendant a engendré une séparation claire entre les mémoires et les calculateurs, augmentant la flexibilité d'instanciation et la réutilisation matérielle. Pour finir, le calculateur est capable d'instancier un CNN quelconque. Cette architecture et sa logique de répartition des neurones font l'objet d'un brevet [LBD16a] et d'un poster [BCB+17].

Nous n'avons traité que les parties calculatoires et d'adressages des intégrations de notre architecture. Nous avons considéré que le SPE reçoit la bonne pondération pour effectuer son calcul. Cependant, nous avons vu dans la section 5.2.3 les aspects théoriques de l'adressage des pondérations, mais pas son implémentation matérielle. Dans la prochaine section, nous allons voir la solution technique développée pour la lecture parallèle des pondérations dans notre architecture.

5.3 Réalisation de la distribution des pondérations

Nous avons vu l'équation qui nous donne la relation entre une pondération et un SPE pour une impulsion d'entrée. J'ai choisi d'utiliser un système de distribution des pondérations entre la mémoire des pondérations et les SPE pour résoudre

cette lecture des pondérations, voir figure 5.17. Nous allons dans cette section détailler la solution matérielle que nous avons mise en place pour cette distribution des pondérations, compatible avec les mémoires numériques ou analogiques.

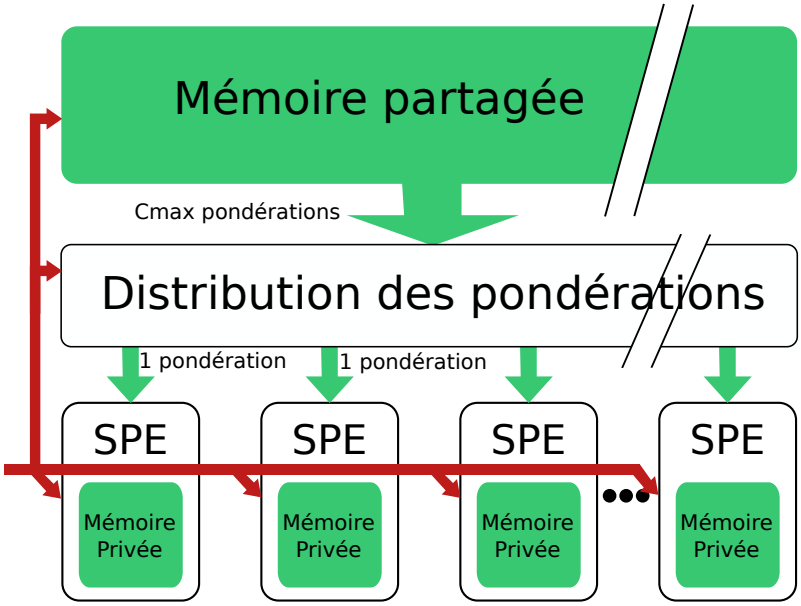

FIG. 5.17: Représentation générale du système de distribution des pondérations.

5.3.1 Problématique de distribution parallèle des pondérations

Nous considérons ici que les valeurs de SPE_x, SPE_y des différents SPE sont fixées matériellement. Nous considérons aussi que la position des pondérations dans le vecteur est fixée lors de la programmation. En étudiant la partie s_x de l'équation (5.3), on remarque son effet sur le sous-vecteur de pondérations que l'on note V_p. Cet effet peut être décomposé en deux modifications de V_p : Premièrement, une réorganisation du vecteur V_p en Vr_p pour $i_x \mod C_{elmX} = 0$. Deuxièmement, pour $i_x \mod C_{elmX} \neq 0$ une permutation circulaire du vecteur Vr_p de $i_x \mod C_{elmX}$ éléments à droite, voir figure 5.18.

Dans le cas multi-filtres, les différents filtres sont soumis aux mêmes impulsions d'entrées. Cependant, la réorganisation initiale ainsi que le vecteur SPE sont différents du cas du filtre unique. Pour créer les vecteurs initiaux et SPE, il faut prendre le cas du filtre unique pour tous nos sous-filtres et concaténer leurs vecteurs, voir figure 5.19.

Pour s_y la logique est identique, la seule différence est que cette permutation n'agit que sur les différentes lignes du filtre. Donc les i_y permutent les sous-vecteurs de pondération, eux-mêmes permutés par i_x.

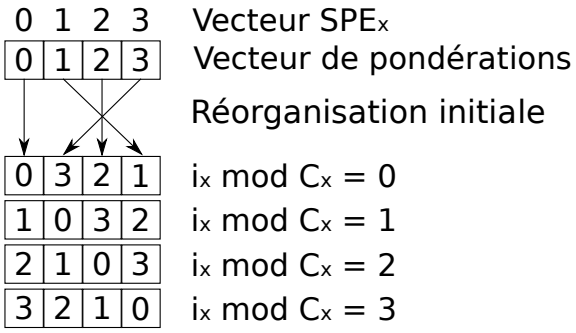

FIG. 5.18: Réorganisation initiale et effet de permutation circulaire sur le vecteur en fonction de l'adresse des impulsions d'entrée.

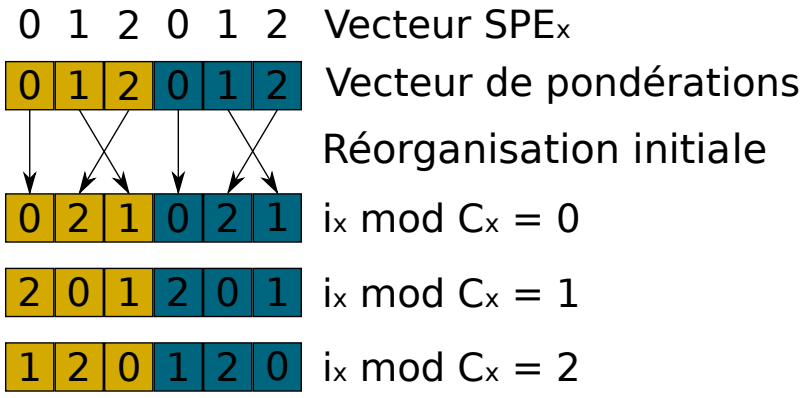

FIG. 5.19: Réorganisation initiale et effet de permutation circulaire sur le vecteur contenant les valeurs de plusieurs filtres en fonction de l'adresse des impulsions d'entrée.

Pour réaliser cette fonction de distribution parallèle, nous avons besoin d'une structure compacte, générique, passant facilement à l'échelle et compatible avec des signaux analogiques. Je me suis donc orienté vers les réseaux de permutation composés de permutateurs deux états. Ce réseau de permutation doit pouvoir être implémenté sur un bus de taille quelconque pour maximiser la flexibilité d'instanciation. Il doit minimiser le nombre de permutateurs pour réduire la surface du silicium. Et enfin, il doit être contrôlable avec un algorithme de routage local au permutateur, ce qui facilitera le passage à l'échelle.

5.3.2 Les réseaux de permutation existants

Les réseaux de permutation sont utilisés pour transmettre et distribuer des informations. Les exemples communs incluent les systèmes d'interconnexions téléphoniques et les ordinateurs à calculs parallèles. L'application typique consiste à connecter N entrées à N sorties, par un réseau où chaque nœud est un commutateur. Nous allons présenter différents réseaux de permutation classiques, nous discuterons de leur adéquation à notre problème, avant d'introduire notre solution.

Nous allons rapidement présenter un premier type de réseau de permutation dit

"bloquant". Les plus utilisés sont les réseaux Butterfly ,Omega et Baseline. Tous ces réseaux ont une faible complexité en nombre de nœuds égal à $\frac{N}{2}log_2(N)$ et des algorithmes de routage efficaces. Ces réseaux ne peuvent traiter des bus dont la taille est égale à 2^N.

Mais, le fait qu'ils soient bloquants implique qu'il n'est pas possible de distribuer parallèlement toutes les valeurs. Ils ne sont donc pas adaptés à notre problématique, car incapables d'effectuer toutes les permutations nécessaires, voir figure 5.20.

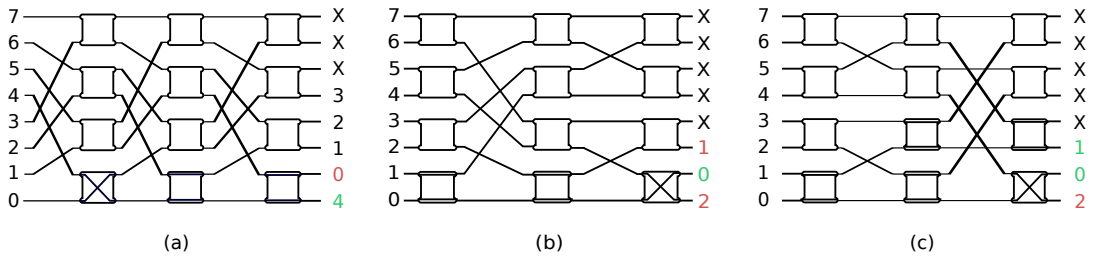

FIG. 5.20: Les permutations non possibles (a) réseaux Omega,(b) réseaux Baseline, (c) réseaux Butterfly. Les éléments en rouge sont les places inatteignables par ces réseaux.

Nous allons maintenant présenter les réseaux de permutations non bloquants. Nous nous focaliserons sur deux structures le Beneš [69] et le Waksman[70] ainsi que leurs extensions pour un bus de taille quelconque As-Beneš [71], AS-Waksman [72] (Arbitrary Size : AS). Ce type de réseaux est capable de résoudre notre problématique de routage parallèle des pondérations.

Dans notre, cas la taille du filtre maximale peut être quelconque, ce qui nous impose des réseaux de taille quelconque. Comme nous pouvons le voir sur le tableau 5.1, la complexité en nombre de nœuds du As-Waksman est la plus faible. Sachant que la taille fait partie d'un de nos critères pour la réduction de la surface du silicium, nous nous focaliserons sur le réseau As-Waksman.

Malgré l'existence d'algorithmes pour les réseaux Beneš [73, 74], nous n'avons pas trouvé d'algorithmes locaux pour le réseau as-Waksman.

Nous constatons que les réseaux actuels ne répondent pas totalement à notre problématique car ils ne peuvent pas implémenter d'algorithmes locaux. Nous avons donc décidé de créer un réseau de permutation pour un bus de taille quelconque avec un algorithme local implémentable, le tout avec une complexité en nombre de nœuds au maximum égale aux réseaux As-Waksman.

n	Beneš	Waksman	As-Beneš	As-Waksman
2	1	1	1	1
3			3	3
4	6	5	6	5
5			8	8
6			12	11
7			15	14
8	20	17	20	17
9			22	21
10			26	25
11			30	29
12			36	33
13			39	37
14			44	41
15			49	45
16	56	49	56	49
17			58	54
18			62	59
19			66	64
20			72	69

TAB. 5.1: Comparatif du nombre de permutateurs des différents réseaux de permutations non bloquants.

5.3.3 Notre réseau de permutation

En partant du vecteur réorganisé que l'on note V_r, nous avons défini un réseau permettant de faire toutes les permutations et les sous-permutations circulaires d'un bus quelconque. Pour ce faire, le réseau est constitué d'un empilement de couches différenciées par leurs degrés *deg*.

Il y a $\lceil log_2(C_{MaxX}) \rceil$ couches, ordonnées par ordre décroissant de degrés par rapport au sens de parcours des données. Dans une couche, les permutateurs ont tous la même taille et connectent deux bus distants de $taille = 2^{deg}$. Les permutateurs d'une couche sont décalés d'un élément entre eux, il n'y a pas deux permutateurs connectés aux deux mêmes bus. Les permutateurs sont ordonnés sur les lignes du bus, les permutateurs les plus à droite sont les premiers par rapport au sens du parcours des données, voir figure 5.21.

L'idée dans ce réseau est de pouvoir déplacer une série contiguë d'éléments de gauche vers n'importe quelle position à droite. Les éléments rejetés à gauche pourront être replacés avec les permutateurs non utilisés, selon la même logique. Le réseau est conçu pour que lors d'un déplacement de B éléments à droite, il est possible de faire par la suite un déplacement de $B-1$ éléments à droite. Ce faisant, le réseau est capable d'effectuer n'importe quelle permutation circulaire, n'importe où sur le bus.

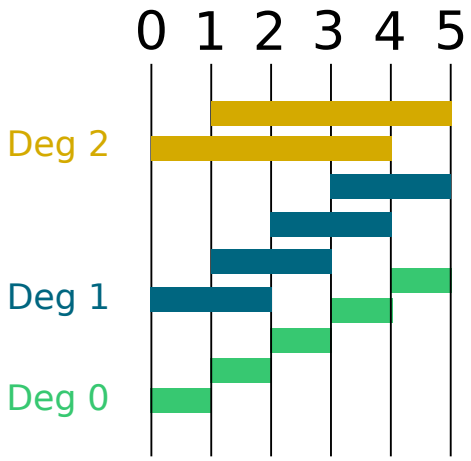

FIG. 5.21: Structure d'une couche du réseau de permutation proposée. Chaque couche est définie par son degré deg qui caractérise la distance entre les deux lignes connectées à un même permutateur.

Comme chaque couche est implémentable pour des bus de taille quelconque, il en est de même pour le réseau. Au niveau du nombre de nœuds du réseau, il est égal à :

$$C_{MaxX}.\lceil log_2(C_{MaxX}) \rceil - 2^{\lceil log_2(C_{MaxX}) \rceil} + 1$$

La complexité en nombre de nœuds de ce réseau est égale à celle du réseau As-

Waksman. Cette structure satisfait donc les exigences en matière de nombre de permutateurs et de flexibilité d'implémentation. Nous allons donc nous intéresser à l'algorithme de routage.

L'algorithme de routage local utilise un vecteur d'adresse avec une adresse par pondération. Ce vecteur est défini par la différence entre la position de la pondération et sa destination, par exemple :

$$diff \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 3 & 4 & 0 & 1 & 2 \end{pmatrix} \quad = \quad 2 \quad 2 \quad 2 \quad -3 \quad -3$$

Chaque permutateur compare le MSB (most significant bit) de l'adresse reçue à sa taille, s'il y a égalité alors le permutateur permute les données et les adresses. Les adresses sont aussi modifiées lors d'une permutation : on soustrait la taille à l'adresse de gauche et on l'additionne à celle de droite. Ce routage peut être directement implémenté dans un permutateur, voir figure 5.22 pour un exemple d'implémentation.

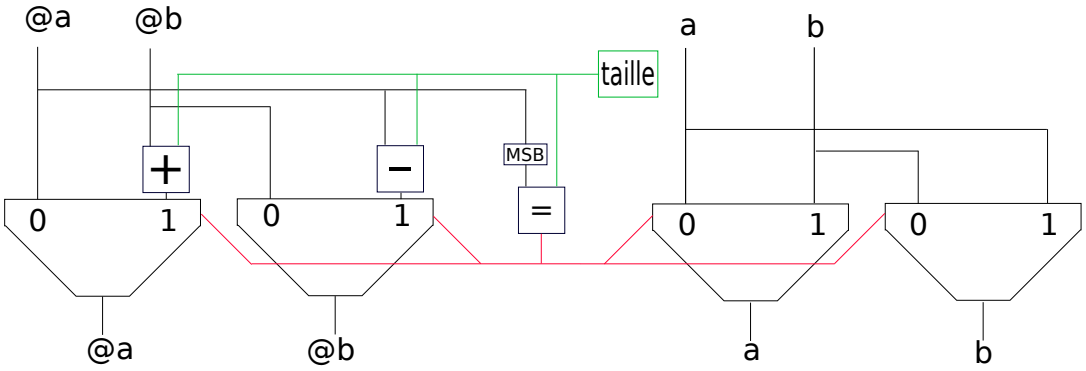

FIG. 5.22: Permutateur numérique avec la logique de l'algorithme de routage.

L'algorithme s'applique sur les bus de façon identique, il peut être implémenté avec un circuit combinatoire pour avoir un temps de routage de 1 cycle. Lorsque le routage s'est effectué correctement, toutes les adresses du vecteur d'adresse sont égales à 0. La figure 5.23 présente le déroulement du routage et les changements dans le vecteur d'adresse couche par couche.

Ainsi ce réseau de permutation permet de faire toutes les permutations circulaires. De plus, ce réseau peut utiliser un algorithme de routage local. Nous allons voir comment utiliser ce réseau pour résoudre le partage des pondérations.

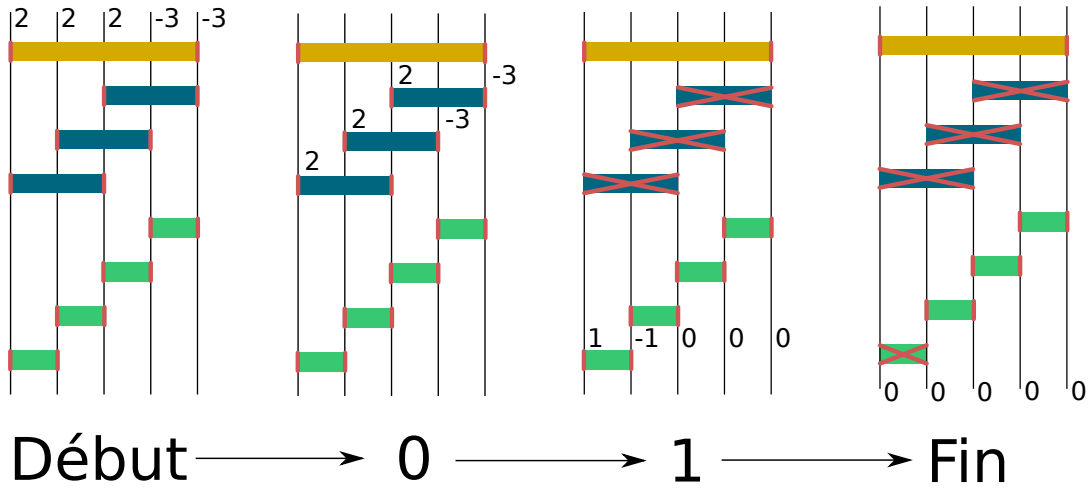

FIG. 5.23: Déroulement de l'algorithme de routage. L'état des permutateurs d'une même couche est déterminé simultanément.

5.3.4 Utilisation en arbre de notre réseau de permutation

Comme nous l'avons vu, le réseau de permutation proposé peut effectuer toutes les permutations circulaires d'un vecteur et de ses sous-vecteurs. Cependant, pour effectuer toute la fonction de routage, il faut pouvoir permuter les vecteurs entre eux. Comme la fonction de permutation des vecteurs est identique à la fonction de permutation des sous-vecteurs de pondération, il est possible d'utiliser le même réseau de permutation. Le réseau final utilise plusieurs réseaux de permutation avec une topologie en arbre, voir la figure 5.24. Cette structure peut être divisée en deux parties, les réseaux X et le réseau Y. Cette structure nous donne une complexité en nombre de permutateurs de :

$$C_{MaxX}(C_{MaxX}\lceil log_2(C_{MaxX})\rceil - 2^{\lceil log_2(C_{MaxX})\rceil} + 1)$$
$$+(C_{MaxY}\lceil log_2(C_{MaxY})\rceil - 2^{\lceil log_2(C_{MaxY})\rceil} + 1)$$

L'algorithme de routage est identique, mais utilise deux vecteurs d'adresse. Le premier est le vecteur d'adresse pour la partie s_x de l'équation, il est identique pour tous les réseaux X. Le deuxième vecteur pour la partie s_y, est utilisé pour piloter le réseau Y. Le routage est fini quand les deux vecteurs d'adresse sont nuls.

Cette structure est compatible avec les autres réseaux de permutation. L'utilisation de l'arbre permet une réduction du nombre de permutateurs par rapport à l'implémentation d'un réseau en un seul bloc, voir figure 5.25.

Nous avons donc une structure en arbre d'acheminement des pondérations parallèles qui peut résoudre notre problématique de lecture des pondérations. Cette structure minimise le nombre de permutateurs utilisés et elle est compatible avec un algorithme de routage local. De plus, elle est compatible avec d'autres types de réseaux de permutation. Cette structure de routage des pondérations a fait l'objet d'un brevet [LBD16b].

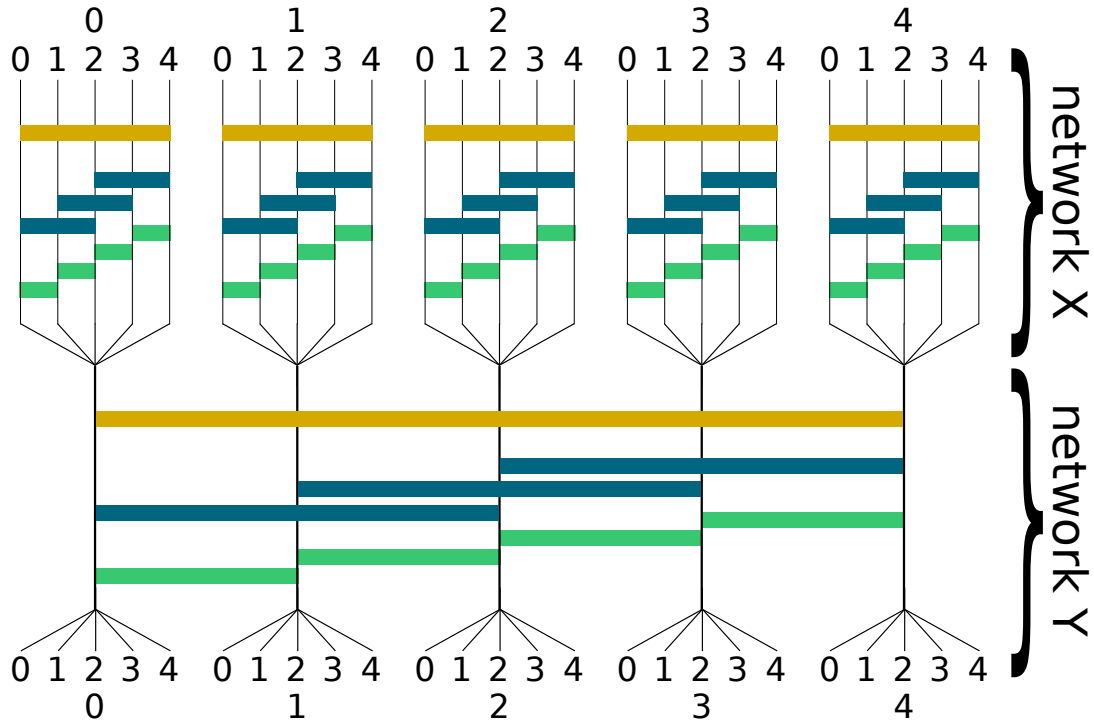

FIG. 5.24: Structure en arbre de réseaux de permutation utilisée pour router les pondérations et noté Routeur. Ce routeur est composé de deux sous-réseaux le X et le Y

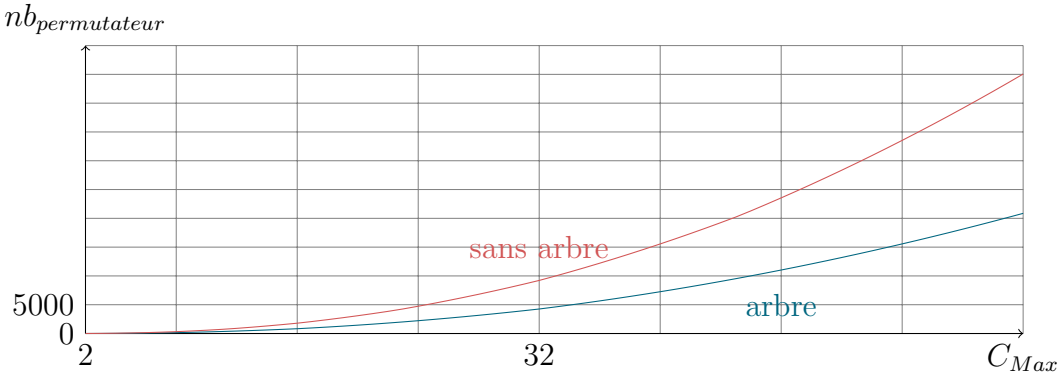

FIG. 5.25: Différence en matière de nombre de permutateurs entre un réseau Waksman (ou notre réseau) avec la structure en arbre et sans cette structure pour $C_{MaxX} = C_{MaxY}$.

5.3.5 Les limites

Le système d'acheminement des pondérations que nous proposons présente quelques limitations pour son passage à l'échelle. Premièrement, l'asymétrie du nombre de permutateurs traversés par les données dans les réseaux. En effet, dans les réseaux, la différence entre le chemin le plus long et le plus court en matière de permutateurs traversés est égale à $C_{Max} - 2$. Cette propriété peut devenir problématique pour la montée en fréquence du système.

La structure en arbre apporte une autre limitation : La taille des bus permutés par les réseaux X et le réseau Y sont différents. Ainsi, pour une précision de pondé-

ration sur N_{pond} bits, les permutateurs des réseaux X sont des entrées de N_{pond} bits et les permutateurs du réseau Y des entrées de $C_{MaxX} \times N_{pond}$. Cette différence peut engendrer une consommation plus importante, car les permutateurs des réseaux Y ont une grande capacité de sortie.

5.4 Implémentation et évaluation de performances et comparaisons

Après avoir défini la chaîne d'exploration et de validations dans le chapitre 4 ainsi que le fonctionnement de l'architecture, nous allons pouvoir évaluer sa performance. Nous avons appliqué cette chaîne à notre architecture et ainsi nous avons montré l'équivalence en termes d'impulsion entre notre architecture et la simulation SystemC. Ceci nous a permis de certifier l'équivalence applicative entre la simulation formelle et l'architecture. Pour finir, nous avons utilisé les différentes simulations matérielles pour estimer les performances et caractéristiques dynamiques de notre système.

Nous allons présenter, dans un premier temps, l'estimation des performances en OP/s et OP/J obtenue par notre architecture pour une configuration donnée. Nous utiliserons cette estimation pour nous comparer aux architectures de l'état de l'art. Nous pourrons ainsi conclure sur la pertinence de notre approche.

5.4.1 Résultat des simulations

Comme nous l'avons vu dans la section 4.3.2, à la fin de notre chaîne de validation nous sommes en mesure d'estimer les performances calculatoires et énergétiques de notre architecture. L'architecture a été dimensionnée pour implémenter un réseau CNN pour la classification de chiffre manuscrit, voir figure 5.26. Les valeurs des différents paramètres de l'architecture utilisées lors des simulations sont données dans le tableau 5.2.

L'architecture ainsi définie, nous avons effectué la simulation RTL pour mesurer le temps de traitement d'une impulsion. Nous avons utilisé la synthèse DC pour obtenir la fréquence maximale de fonctionnement, la puissance moyenne et la surface. Et pour finir la simulation PrimeTime pour la puissance pour un réseau donné, voir figure 5.26. Les résultats des simulations matérielles sans mémoire sont présentés

nombre de couches	$C_{MaxX} \times C_{MaxY}$	C_{MaxZ}	$I_{elmX} \times I_{elmY}$	nombre de neurones
8	11×11	2048	32×32	30 976

TAB. 5.2: Configuration du NeuroSpike utilisée lors des simulations matérielles.

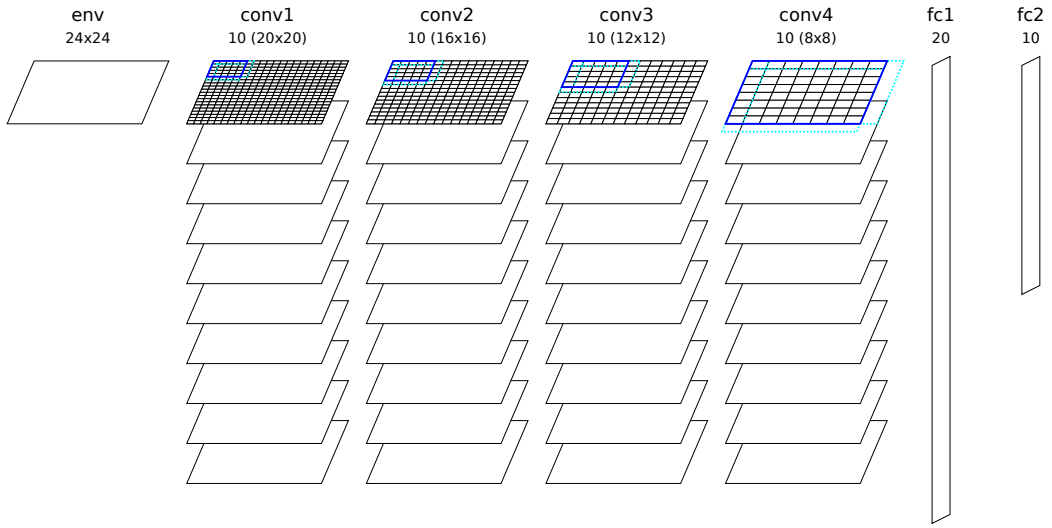

FIG. 5.26: Topologie de réseau conventionnel utilisé pour résoudre la problématique de classification de chiffre manuscrit de la base Mnist et l'estimation de consommation PrimeTime.

	Puissance (mW)	Puissance pique (mW)	surface (mm^2)	freq (MHz)	$t_{traitement}$	$Nb_{neurones}$
dc	70,9483		0,309	400	7,5 ns	121
pt	67,8	168,3	0,309	400	0,3ms/14k impulsions	4,3M

TAB. 5.3: Résultat des simulations matérielles du calculateur seul. La synthèse DC donne des valeurs pour un taux d'utilisation matérielle moyen de 50%. Les valeurs de la simulation PrimeTime (pt) sont en fonction d'un réseau.

dans le tableau 5.3. La répartition de la consommation moyenne dans l'architecture est présentée dans le tableau 5.4.

Pour avoir une estimation de l'architecture mémoire comprise, nous utilisons une estimation de la consommation mémoire. La puissance consommée par ces mémoires est fonction d'un taux d'utilisation en lecture et en écriture. Ce taux est déduit de l'utilisation moyenne de ces mémoires lors du calcul d'une couche, voir le tableau pour les résultats de cette estimation 5.5.

En analysant les résultats en surface, on remarque que la surface est principalement due aux mémoires, environ 87.53% de l'architecture. Ce résultat est cohérent avec les besoins des neurones impulsionnels, comme nous l'avons montré dans la section 2.4.1, le modèle impulsionnel IF utilise des opérateurs simples, mais au détriment de la sauvegarde d'un état interne (l'intégration). Ceci explique que l'on retrouve cette particularité lors de l'estimation matérielle.

Arbre	$2,1\%$
FIFO	$1,8\%$
121 SPE	$\approx 93,17\%$
Autres	$\approx 2,93\%$
Total	100%

TAB. 5.4: répartition de la consommation entre les différents blocs de l'architecture mémoire non comprise.

Mémoire	Puissance (mW)	Surface (mm^2)	freq (MHz)	Lecture (%)	Écriture (%)
Pondérations	19,096	1,503 522	400	20	0
Intégrations	64,149	0,393 492	400	20	20
Paramètres	0,854	0,270 867	400	3	0

TAB. 5.5: Estimation de la surface et de la puissance des mémoires FDSOI 28nm pour une fréquence de fonctionnement de 400 MHz en fonction d'un taux de lecture/écriture.

Au niveau de la consommation énergétique, la répartition est plus équitable avec $54,798\%$ de la consommation due à la mémoire. Cette particularité peut s'expliquer par l'utilisation limitée des lectures mémoire et écritures mémoire qui représente au maximum 20% du temps de traitement et l'utilisation de mémoires faible consommation (ST SPREG BODYBIAS). On remarque aussi que la consommation de l'architecture seule est principalement due aux SPEs, notre logique de recoupement d'adressage, l'arbre de pondérations, le pré-calcul et le post-calcul sont assez performants pour que la consommation de ces blocs soit négligeable par rapport au calcul du modèle IF par les SPE.

En analysant le temps moyen de traitement d'une impulsion soit $7,5ns$ et le temps de traitement moyen d'une impulsion pour le réseau de la figure 5.26 soit $20,7ns$ on trouve une différence de $13.2ns$. Cette différence s'explique par le fait que la performance de la simulation PrimeTime prend en compte les communications et la récursivité des impulsions pour le multi-couches. Cette différence est variable en fonction de la topologie du réseau utilisé, ce rebouclage est à étudier cas par cas pour trouver le meilleur compromis possible, car son coût sur la vitesse de calcul est non négligeable. Pour jouer sur le rebouclage, il est possible d'utiliser plusieurs calculateurs dans lesquels les différentes couches du réseau seront distribuées.

Pour la puissance moyenne et la puissance PrimeTime, on retrouve un écart de $3,14mW$ qui est dû au fait que la synthèse DC ne considère qu'une activité générale de 50%. Or, lorsqu'une impulsion est reçue, seul un nombre limité de neurones va se déclencher et donc seule une petite partie du matériel va être utilisé. Les résultats de

	NeuroSpike
Technologie	28 nm FDSOI
Fréquence	400MHz
Surface	2,466 mm^2
Nombres de neurones	30k
Tailles du filtre max	11×11
Nombre de filtres max	121
Type de calcul	numérique
Impulsions en parallèle max	1
Nombres de neurones parallèles	121
Temps de traitement	7,5 ns
Précision	8 bits
Consommation moy	153,4 mW

TAB. 5.6: Résumé des caractéristiques estimées en simulation de notre calculateur le NeuroSpike.

la simulation PrimeTime sont donc plus réalistes sur cet aspect de consommation.

Au final, nous avons résumé les différentes caractéristiques technologiques, calculatoires et la consommation de notre architecture NeuroSpike dans le tableau 5.6. Ce sont ces valeurs qui seront utilisées pour les comparaisons avec l'état de l'art.

Au vu de ces analyses, pour calculer les performances de notre architecture nous ne considérerons que le traitement d'une seule couche pour le filtre maximal, ici de 11 éléments, comme nous l'avons fait pour les autres architectures impulsionnelles, soit $7,5ns$ par impulsion pré-synaptique pour 121 ACC. Nous considérerons une consommation totale de $153,4mW$ qui correspond à la consommation des mémoires et du matériel donné par PrimeTime. En définitive, les performances de notre architecture en nombre d'opérations atomiques (définies dans la section 2.4.3) sont de $387GOP/s$ et de $2,58TOP/J$.

Pour conclure, le fonctionnement de notre architecture a été validé. Les simulations matérielles nous ont permis de déterminer ses caractéristiques, ses performances. Nous allons pouvoir désormais nous comparer à l'état de l'art et conclure sur notre approche.

5.4.2 Comparaison avec les architectures impulsionnelles

Dans cette section, nous allons nous concentrer sur la comparaison avec les architectures impulsionnelles identifiées dans la section 3.3. Comme nous avons vu dans la section 3.3, les architectures spécialisées sont plus efficaces que les architectures génériques et tout à fait pertinentes dans le cadre du calcul de l'inférence des CNN

impulsionnels. Nous allons donc limiter la comparaison de notre architecture aux architectures spécialisées.

Pour faire cette comparaison, nous avons vu dans la section 2.5.2 qu'il est possible de comparer les architectures impulsionnelles directement en considérant le même nombre d'impulsions pré-synaptiques pour toutes les architectures. Il faut alors exprimer leur performance en fonction du nombre d'opérations atomiques traitées. Nous obtenons alors deux comparatifs, un en vitesse de calcul en OP/s et un en efficacité énergétique de calcul en OP/J. Le comparatif des architectures impulsionnelles spécialisées avec notre architecture est représenté dans la figure 5.27.

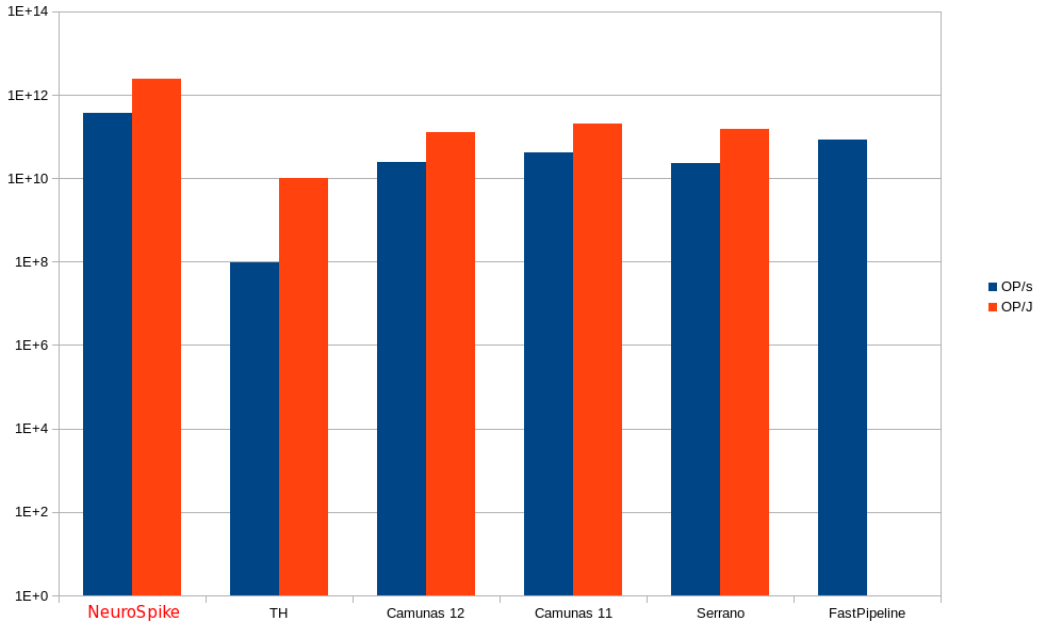

FIG. 5.27: Comparaison en matière de performance calculatoire d'une opération atomique des différentes architectures impulsionnelles. Deux métriques sont utilisées, une de vitesse de calcul en nombre d'opérations atomiques par seconde OP/s et une d'efficacité de ce calcul en nombre d'opérations atomiques par Joule OP/J. On considère la même application pour toutes les architectures et le même nombre d'impulsions en entrées.

Les résultats obtenus sont des estimations en fonction de notre système de calcul d'une opération atomique. Cependant, on peut dire que notre logique de multiplexage complet des neurones ainsi que la distribution parallèle des pondérations nous donnent des résultats intéressants par rapport aux autres architectures impulsionnelles.

Ainsi, au vu des résultats, on remarque qu'au niveau de la performance calculatoire en vitesse de calcul OP/s notre architecture nous permet d'obtenir un $\times 4$ par rapport à la solution la plus rapide de l'état de l'art.

La réduction matérielle apportée par le multiplexage des neurones et le regroupement de la logique d'adressage a un impact en matière de consommation. Ainsi au vu des résultats, on constate que notre architecture nous permet d'obtenir un gain en matière de performance énergétique du calcul d'environs $\times 11$ par rapport à la solution la plus efficace de l'état de l'art. Notre architecture offre une meilleure efficacité énergétique pour le calcul des CNN par rapport aux architectures de l'état de l'art.

En considérant la flexibilité de notre approche en matière de types de couches, de connectivité entre couches et de nombre de couches, nous pouvons affirmer que notre approche calculatoire et architecturale est tout à fait pertinente pour le calcul des CNN impulsionnels au regard de l'état de l'art des architectures impulsionnelles. Nous allons maintenant nous intéresser à la comparaison entre notre architecture et les architectures formelles.

5.4.3 Comparaison aux architectures formelles

Cette comparaison entre architectures formelles et architectures impulsionnelles, n'est pas une chose évidente à faire, principalement du fait des différences d'opérateurs et de signaux d'entrées entre ces deux modèles. Nous avons donc mis en place une logique de comparaison de la section 2.5.2, en considérant le nombre d'impulsions pré-synaptiques par entrées pour qu'une architecture impulsionnelle soit aussi rapide et/ou consomme la même énergie qu'une architecture formelle donnée. La figure 5.28 représente le nombre d'impulsions pré-synaptiques par entrées pour que notre architecture soit aussi rapide qu'une architecture formelle donnée. La figure 5.29 représente le nombre d'impulsions pré-synaptiques par entrées pour que notre architecture consomme la même énergie qu'une architecture formelle donnée.

On constate qu'au niveau du temps de traitement notre approche ne permet pas toujours d'obtenir au moins une impulsion pré-synaptique par entrées. Ainsi en matière de vitesse de traitement notre architecture a une limite basse d'environ $0,1$ impulsion pré-synaptique par entrées et en matière d'efficacité énergétique d'environ $0,2$ impulsion pré-synaptique par entrée. Cependant, on constate que comparé à la solution nn-X, notre architecture est possiblement plus efficace en matière d'efficacité énergétique, avec $3,8$ impulsions pré-synaptiques par entrées.

Pour obtenir plus d'une impulsion pré-synaptique par entrées en matière de temps de traitement et d'efficacité énergétique pour toutes les architectures formelles notre architecture devrait traiter au moins 15 impulsions pré-synaptiques parallèlement, et/ou pour obtenir plus d'une impulsion pré-synaptique par entrées en matière d'efficacité énergétique notre architecture devrait consommer environ $40mW$. Or, on estime le surcoût énergétique de la mémoire d'intégration à 42% de la consommation totale et logique d'adressage des intégrations consomme environs $35mW$ soit un surcoût total de $99,26mW$. Ainsi, notre architecture serait en mesure

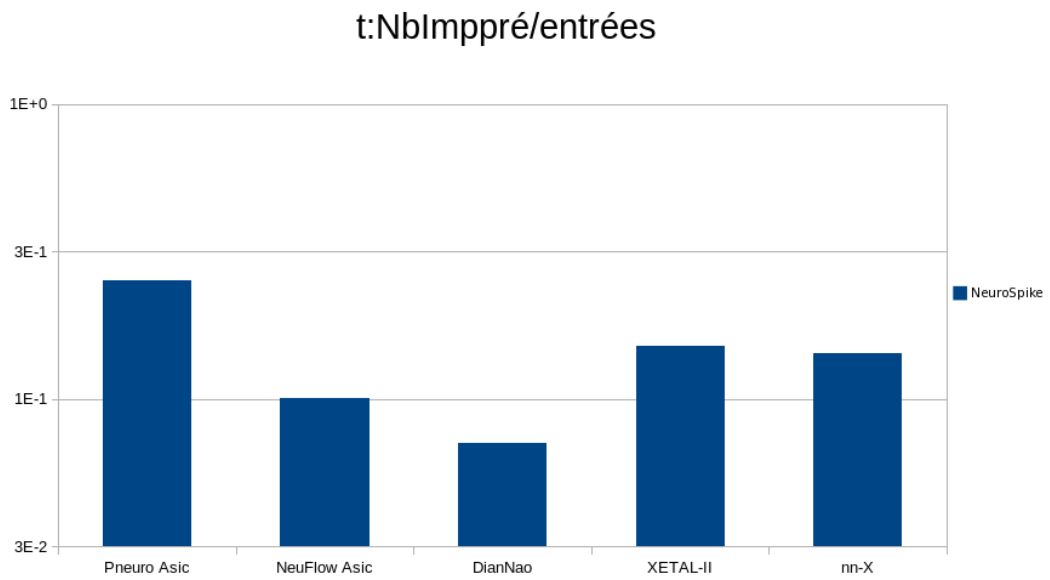

FIG. 5.28: Le nombre d'impulsions pré-synaptiques par entrées pour une équivalence en matière de temps de traitement. Le nombre d'impulsions pré-synaptiques par entrées est exprimé pour chaque architecture formelle.

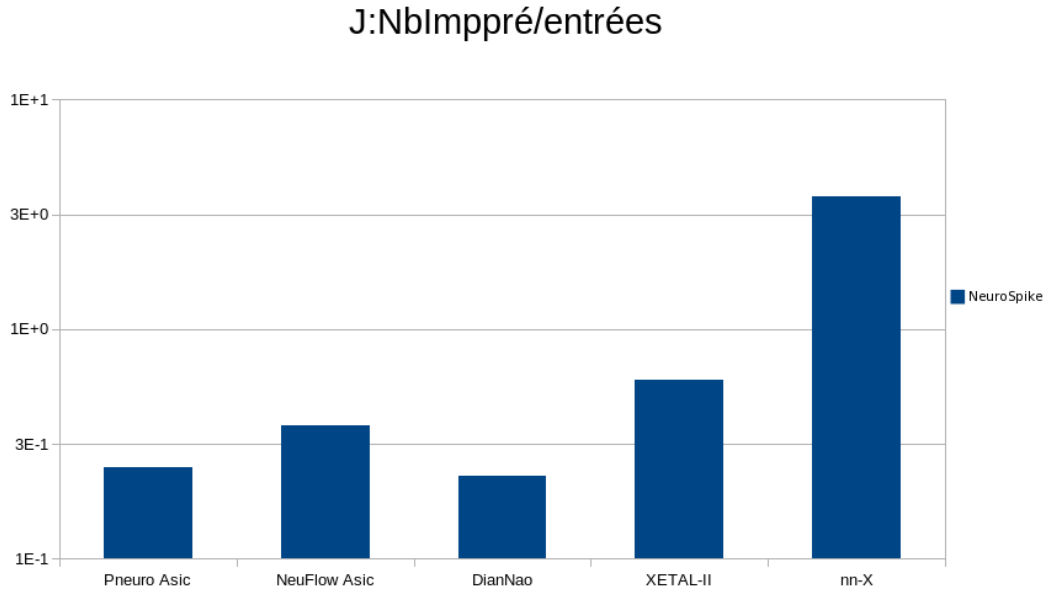

FIG. 5.29: Le nombre d'impulsions pré-synaptiques par entrées pour une équivalence en matière d'énergie. Le nombre d'impulsions pré-synaptiques par entrées est exprimé pour chaque architecture formelle.

de rivaliser avec les architectures formelles en matière d'efficacité énergétique si elle n'avait pas à sauvegarder les intégrations de tous les neurones du réseau.

5.5 Conclusion

Nous avons présenté notre approche d'une architecture spécialisée dans le calcul de l'inférence des CNN impulsionnels. Nous avons montré que notre approche est assez flexible pour instancier différents types de couches et différentes topologies de CNN. Finalement, notre architecture, à la différence des architectures impulsionnelles de l'état de l'art, est capable d'instancier un CNN complet.

Nous avons expliqué, notre logique de répartition des neurones dans les SPE, la mutualisation des adressages et notre approche de la lecture des pondérations. Cette logique de lectures parallèles des pondérations en utilisant un réseau de permutation nous permet d'envisager l'utilisation de mémoire analogique, voir un apprentissage en ligne. Toutes ces optimisations nous ont permis d'obtenir de meilleures performances en OP/s et OP/J que les architectures impulsionnelles de l'état de l'art. Notre approche du point de vue des architectures impulsionnelles de l'état de l'art apparait donc pertinente.

En utilisant notre méthode de comparaisons entre les architectures formelles et les architectures impulsionnelles, nous avons montré que notre architecture est probablement moins performante en temps de traitement et en énergie consommée que les architectures formelles les plus performantes de l'état de l'art. Nous avons montré que ce résultat est surement dû au surcoût de la sauvegarde de l'intégration indispensable pour les modèles impulsionnels.

Cependant, nos métriques ne prennent pas en compte des aspects de chargement mémoire, de sauvegarde des états intermédiaires ou encore de distribution du calcul dans les architectures formelles, car ces aspects sont difficilement évaluables sans un réseau fixé. Or, ces aspects n'existent pas ou ont été pris en comptes pour l'estimation des architectures impulsionnelles. Cela implique que la performance en MAC/s des architectures formelles est surestimée par rapport aux estimations des architectures impulsionnelles. Pour statuer sur l'utilisation d'une architecture impulsionnelle à la place d'une architecture formelle il faut donc obligatoirement les comparer pour un réseau donné.

De plus, dans le cas de notre architecture, certaines optimisations pourraient être faites pour améliorer ses performances. Ainsi, une implémentation mixte analogique-numérique avec des SPE analogiques nous permettrait potentiellement de réduire notre consommation énergétique. Une meilleure gestion des connexions partielle entre couches réduirait nos besoins mémoire. Une étude plus poussée sur la répartition des calculs dans les SPE nous permettrait d'envisager dans certains cas le traitement de plusieurs impulsions simultanément. L'implémentation d'un apprentissage en ligne offrirait la possibilité d'envisager une indépendance à la transposition et au codage fréquentiel.

Pour conclure, selon notre logique de comparaison l'utilisation des modèles impulsionnels avec un codage fréquentiel pour le calcul de l'inférence des CNN ne

constitue pas, dans tous les cas, une solution pertinente par rapport aux architectures formelles, ceci est principalement dû au surcoût des mémoires d'intégration. Mais cette logique ne prend pas en compte tous les aspects du calcul formel et au vu des potentielles améliorations de notre approche, elle reste une piste intéressante pour de futures études. Nous allons dans le prochain chapitre synthétiser notre approche et discuter des calculateurs impulsionnels spécialisés ainsi que leurs possibles évolutions et utilisations.

Chapitre 6

Conclusion générale

Contents

6.1 Synthèse

Nous avons vu dans un premier temps que les applications des CNN sont nombreuses. Les CNN sont des structures de calcul efficaces, régulières, mais complexes en matière de nombre d'opérations et de partage de données. Nous avons vu qu'il existait deux modèles de neurones, les modèles formels et les modèles impulsionnels. Le but a été d'étudier la possible réduction de complexité calculatoire des CNN par l'utilisation du modèle impulsionnelle.

Nous avons ensuite présenté les CNN, les deux modèles de neurones et les conditions de leur utilisation. J'ai par la suite mis en place une logique de comparaisons entre ces deux modèles et leurs architectures pour pouvoir statuer sur les performances de ces architectures. Nous avons montré que l'utilisation du modèle impulsionnel permet en théorie une réduction de la complexité calculatoire des réseaux conventionnels.

Nous avons ensuite appliqué notre logique de comparaisons aux architectures de l'état de l'art. Cette étude a montré que les architectures impulsionnelles de l'état de l'art ne sont pas aussi performantes que leur homologue impulsionnelle. Nous avons aussi constaté qu'il n'existait pas de structure matérielle capable de calculer l'inférence impulsionnelle pour un CNN complet. Cependant, nous avons identifié de possibles gains par l'utilisation d'une architecture spécialisée pour les CNN avec multiplexage des neurones.

Partant de cette constatation, nous avons décidé de proposer notre approche d'une architecture de calcul impulsionnel spécialisé dans les CNN. Nous avons alors introduit notre chaîne de validation générique et nos résultats en matière de simplification matérielle de l'opération de MaxPooling. Notre chaîne ainsi définie nous permet de valider et de mesurer notre approche.

Par la suite, nous avons développé notre approche architecturale. Nous avons montré que notre approche évite la redondance des pondérations et limite le nombre de calculateurs nécessaire et permet un temps constant dans le traitement d'une impulsion d'entrée. Au final nous avons proposé une architecture impulsionnelle compatible avec l'instanciation d'un CNN complet. Nous avons par la suite estimé les performances de cette architecture pour une instanciation numérique en FDSOI 28 nm à 400 mHz.

Nous avons constaté que notre approche est pertinente et représente une bonne solution par rapport à l'état de l'art des architectures impulsionnelles. Cependant, il reste des réserves quant à sa pertinence du point de vue des architectures formelles.

6.2 Réflexions et questions ouvertes

Les travaux développés tout au long du mémoire ont permis la mise en place et l'évaluation d'une architecture CNN impulsionnelle. Nous avons montré que par rapport aux architectures formelles notre architecture est moins performante. Ce qui tend à montrer que les architectures impulsionnelles ne sont pas forcement pertinentes pour les problématiques de l'inférence.

Cependant, avant de statuer sur la pertinence des architectures impulsionnelles pour l'inférence, il me semble intéressant d'évoquer plusieurs aspects du calcul impulsionnel que nous n'avons pas exploités lors de cette thèse et qui potentiellement améliorent les performances en inférence des architectures impulsionnelles.

6.2.1 L'approche analogique du calcul

L'architecture proposée dans ce mémoire a été évaluée dans une forme exclusivement numérique. Cependant, lors de la conception de l'architecture, la chaîne de calcul a été conçue pour être compatible avec des signaux analogiques. Ce qui signifie que la valeur des pondérations ainsi que les modèles du neurone peuvent être implémentés avec un circuit analogique. La conséquence de l'utilisation d'un circuit analogique serait la réduction de la consommation de la partie mémoire et calculatoire. Cependant, le choix de multiplexage des neurones pose plusieurs questions pour l'instanciation analogique.

Premièrement : l'implémentation du SPE et particulièrement l'instanciation de plusieurs intégrations pour une unité de calcul. Classiquement, dans le modèle analogique l'intégration est effectuée à l'aide d'une capacité. Chaque neurone est composé de son circuit de déclenchement et de sa capacité. Cependant, dans notre approche de neurones multiplexés, plusieurs neurones partagent le même circuit de déclenchement. Ceci impose l'utilisation de plusieurs capacités adressables pour chaque circuit de déclenchement. Le coût de cette mémoire capacitif est difficilement évaluable.

Deuxièmement : l'implémentation analogique de la fonction de MaxCompt. Si cette fonction a l'avantage d'utiliser peu de ressources mémoires et calculatoire, sa transposition analogique reste à effectuer. Il est important d'évaluer les ressources matérielles nécessaires et leur adéquation avec les ressources disponibles dans un neurone impulsionnel analogique.

6.2.2 Codage de l'information et la transposition

Nous avons proposé au cours de ce mémoire une chaîne de transposition des réseaux formels vers les réseaux impulsionnels. Cette chaîne de transposition nous permet de faire une étude de la topologie du réseau formelle, de la réduction des dynamiques, ainsi que l'apprentissage, puis de convertir ce réseau en impulsionnel.

Ainsi, la chaîne classique d'estimation des réseaux formels est maintenue et seule la méthode d'inférence est modifiée. Cependant, plusieurs points de cette méthode sont sous-optimaux.

Premièrement, la problématique de la transposition matérielle des images. Ainsi, dans l'approche proposée le traitement d'une image nécessite dans un premier temps sa transposition en impulsions en utilisant un codage fréquentiel. Cette approche nécessite donc la mise en place d'un convertisseur images vers impulsions, en début de la chaîne de traitement. Ce convertisseur doit parcourir tous les pixels d'une image pour lire leur valeur, convertir cette valeur en impulsions et les ordonnancer. Ainsi, la fonction de convertisseur risque d'avoir un impact non négligeable sur les performances globales avec l'augmentation de la taille des images d'entrée.

Deuxièmement, la question de l'optimisation du codage impulsionnel. Ainsi, la transposition du formel vers l'impulsionnel nous garantit une équivalence dans le traitement de l'information et le maintien de la topologie du réseau et du jeu de pondération. Ceci nous permet de profiter de l'apprentissage formel en impulsionnel. Cela nous permet également de mettre en place des techniques de comparaison entre solution formelle et impulsionnelle. Cependant cette transposition ne nous garantit pas que le réseau impulsionnel obtenu soit optimal en matière d'efficacité du codage impulsionnel. Ainsi sans condition sur le nombre d'impulsions propagées, le nombre d'impulsions traitées est sûrement supérieur à l'optimal.

Troisièmement, l'inadéquation des modèles impulsionnels avec les calculateurs classiques (CPU, GPU) rend le calcul plus long. La transposition, mais surtout l'estimation de grand réseau (AlexNet, Google etc.) en impulsionnel est difficilement accélerable. Il serait alors possible d'utiliser des architectures génériques. Cependant, elles nécessiteraient une très grande flexibilité en matière de modèles implémentables et de logique d'apprentissage réduit ainsi leur efficacité énergétique.

En définitive, la problématique de transposition, l'optimisation du traitement impulsionnel sont des points-clés à étudier. Ceci est d'autant plus vrai que l'efficacité des architectures impulsionnelles est liées au nombre d'impulsions traitées. Ainsi, l'amélioration du codage impulsionnel aura des conséquences immédiates sur les performances générales des architectures impulsionnelles.

6.3 Perspectives

Pour ma part et au regard des propriétés des solutions impulsionnelles, les architectures et la chaîne d'apprentissage impulsionnel proposent une approche d'avenir dans les systèmes fortement contraints. Je suis convaincu que ces types d'architectures sont destinés à se développer, voir devenir plus pertinents que les solutions formelles actuelles. Les différents travaux effectués m'ont permis de cerner quelques problématiques qui à mon sens sont indispensables à étudier pour péren-

niser notre approche calculatoire des CNN. Pour conclure ce mémoire, je voudrais exposer quelques pistes qui me semblent intéressantes à étudier.

6.3.1 Perspectives à court terme

L'étude du calcul analogique qui nous permettrait d'estimer le gain par utilisation du calcul analogique qui nous permettrait potentiellement de surpasser les architectures formelles.

L'optimisation de l'architecture au niveau des SPE pour réduire leur consommation avec une logique de clock gating, au niveau de la concaténation des filtres dans le vecteur de pondération pour optimiser la quantité mémoire utilisée et au niveau du séquencement des traitements pour optimiser le pipeline.

L'optimisation du modèle impulsionnel par réduction du coût de la mémoire d'intégrations, la réduction de la précision d'intégration nécessaire au fonctionnement des modèles impulsionnels permettrait de réduire l'écart d'efficacité énergétique entre les architectures formelles et les architectures impulsionnelles.

6.3.2 Perspectives à moyen terme

L'implémentation en NOC nous permettrait d'envisager un traitement parallèle de plusieurs impulsions. Ce traitement parallèle multi-impulsions, permettrait potentiellement d'augmenter l'efficacité de traitement. Cette étude nécessite également une analyse de la répartition d'un réseau sur plusieurs modules NeuroSpike.

Le passage à l'échelle impulsionnelle est une problématique due à l'inadéquation du calcul impulsionnel et des architectures classiques. Cette inadéquation rend difficile l'estimation de performance de très grands réseaux impulsionnels. Cependant, il est possible d'imaginer la création d'une approximation calculatoire plus facilement parallélisable ou utilisation d'architectures fortement génériques (type TrueNorth) dédiées à l'estimation de réseau. L'implémentation de ces réseaux pourrait alors être faite sur des architectures spécialisées (telle que la notre) pour obtenir une plus grande efficacité en inférence.

6.3.3 Perspectives à long terme

L'indépendance totale au formel et l'utilisation d'autre codage pour les impulsions nous permettrait de minimiser le nombre d'impulsions à traiter pour une problématique donnée. Ceci nécessitera la création d'une chaîne complète impulsionnelle et donc d'un apprentissage impulsionnel (STDP etc). Cette chaîne

nous permettrait d'avoir un contrôle plus fin sur le nombre d'impulsions. Cependant, la comparaison avec les solutions formelles serait complexifiée, car complètement dépendante de la problématique traitée.

La création d'un traitement impulsionnel complet composé de capteurs impulsionnels (ex. dvs128) et de calculateur impulsionnel. Ceci permettrait une indépendance complète avec les données vectorielles et un traitement asynchrone de l'information. Cette logique permettrait, à mon avis, de réduire le coût énergétique en n'activant que les parties calculatoires uniquement lors de la réception d'une information.

Publications personnelles

[BCB+17] O. Bichler, A. Carbon, D. Briand, V. Lorrain, and J. M. Philippe. N2d2 : A framework for deep neural networks exploration, spike transcoding and code generation for cots and dedicated hw ips. In *DAC 2017*, 2017.

[LB17] V. Lorrain and O. Bichler. Brevet : Calculateur pour reseau de neurones impulsionnel avec agregation maximale, juil 2017.

[LBD16a] V. Lorrain, O. Bichler, and A. Dupret. Brevet : Dispositif et procede de calcul de convolution d'un reseau de neurones convolutionnel, avr 2016.

[LBD16b] V. Lorrain, O. Bichler, and A. Dupret. Brevet : Dispositif et procede de distribution de donnees de convolution d'un reseau de neurones convolutionnel, avr 2016.

PUBLICATIONS PERSONNELLES

Bibliographie

[1] J. N. Heemskerk, "Overview of neural hardware," *Neurocomputers for Brain-Style Processing. Design, Implementation and Application*, 1995.

[2] T. Y. W. Choi, P. A. Merolla, J. V. Arthur, K. A. Boahen, and B. E. Shi, "Neuromorphic implementation of orientation hypercolumns," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 52, pp. 1049–1060, June 2005.

[3] L. Camunas-Mesa, A. Acosta-Jimenez, C. Zamarreno-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 32, *times*, 32 pixel convolution processor chip for address event vision sensors with 155 ns event latency and 20 meps throughput," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 58, pp. 777–790, April 2011.

[4] A. Yousefzadeh, T. Serrano-Gotarredona, and B. Linares-Barranco, "Fast pipeline 128 x 128 pixel spiking convolution core for event-driven vision processing in fpgas," in *2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pp. 1–8, June 2015.

[5] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 10–14, IEEE, 2014.

[6] NVIDIA, "Gpu-based deep learning inference :a performance and power analysis." `https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf`, 2015.

[7] C. Farabet, R. Paz, J. Pérez-Carrasco, C. Zamarreño-Ramos, A. Linares-Barranco, Y. LeCun, E. Culurciello, T. Serrano-Gotarredona, and B. Linares-Barranco, "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers in neuroscience*, vol. 6, 2012.

[8] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free

event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 2706–2719, Nov 2013.

[9] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[10] K. Fukushima, "Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., DTIC Document, 1985.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[15] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," *Artificial Neural Networks–ICANN 2010*, pp. 92–101, 2010.

[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of mathematical biology*, vol. 5, no. 4, pp. 115–133, 1943.

[17] P. J. Werbos, *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[18] Y. Le Cun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School*, pp. 21–28, CMU, Pittsburgh, Pa : Morgan Kaufmann, 1988.

[19] J. L. Holt and T. E. Baker, "Back propagation simulations using limited precision calculations," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. ii, pp. 121–126 vol.2, Jul 1991.

[20] P. Moerland and E. Fiesler, "Neural network adaptations to hardware implementations," tech. rep., IDIAP, 1997.

[21] F. Choudry, E. Fiesler, A. Choudry, and H. J. Caulfield, "A weight discretization paradigm for optical neural networks," in *in Proceedings of the International Congress on Optical Science and Engineering*, pp. 164–173, SPIE, 1990.

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression : Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[23] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 4409–4412, IEEE, 2012.

[24] L. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, no. 5–6, pp. 303 – 304, 1999.

[25] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, 1952.

[26] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical Journal*, vol. 1, no. 6, pp. 445 – 466, 1961.

[27] R. B. Stein, "A theoretical analysis of neuronal variability," *Biophysical Journal*, vol. 5, no. 2, pp. 173 – 194, 1965.

[28] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013.

[29] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[31] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 431–438, June 2008.

[32] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950, May 2010.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[34] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3642–3649, IEEE, 2012.

[35] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow : A runtime reconfigurable dataflow processor for vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pp. 109–116, IEEE, 2011.

[36] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao : A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49, pp. 269–284, ACM, 2014.

[37] M. Duranton, "L-neuro 2.3 : A vlsi for image processing by neural networks," in *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on*, pp. 157–160, IEEE, 1996.

[38] A. A. Abbo, R. P. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen, and M. Heijligers, "Xetal-ii : a 107 gops, 600 mw massively parallel processor for video scene analysis," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 192–201, 2008.

[39] F. Rosenblatt, "The perceptron : A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[40] C. Mead, *Analog VLSI and Neural Systems*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1989.

[41] C. A. Mead and M. Rem, "Cost and performance of vlsi computing structures," *IEEE Transactions on Electron Devices*, vol. 26, pp. 533–540, Apr 1979.

[42] C. Mead and M. Rem, "Minimum propagation delays in vlsi," *IEEE Journal of Solid-State Circuits*, vol. 17, pp. 773–775, Aug 1982.

[43] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, pp. 1629–1636, Oct 1990.

[44] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, pp. 2454–2467, Dec 2013.

[45] G. Indiveri and S. C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, pp. 1379–1397, Aug 2015.

[46] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid : A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, pp. 699–716, May 2014.

[47] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth : Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, pp. 1537–1557, Oct 2015.

[48] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, Sept 2011.

[49] M. A. Petrovici, B. Vogginger, P. Müller, O. Breitwieser, M. Lundqvist, L. Muller, M. Ehrlich, A. Destexhe, A. Lansner, R. Schüffny, J. Schemmel, and K. Meier, "Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms," *PLOS ONE*, vol. 9, pp. 1–30, 10 2014.

[50] C. Mayr, J. Partzsch, M. Noack, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schüffny, "A biological-realtime neuromorphic system in 28 nm cmos using low-leakage switched capacitor circuits," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, pp. 243–254, Feb 2016.

[51] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic architectures for spiking deep neural networks," in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 4.2.1–4.2.4, Dec 2015.

[52] P. Venier, A. Mortara, X. Arreguit, and E. A. Vittoz, "An integrated cortical layer for orientation enhancement," *IEEE Journal of Solid-State Circuits*, vol. 32, pp. 177–186, Feb 1997.

[53] D. Hubel, *Eye, Brain and Vision*. Freeman, 1987.

[54] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[55] D. H. Hubel and T. N. Wiesel, "Ferrier lecture : Functional architecture of macaque monkey visual cortex," *Proceedings of the Royal Society of London B : Biological Sciences*, vol. 198, no. 1130, pp. 1–59, 1977.

[56] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II : Analog and Digital Signal Processing*, vol. 47, pp. 416–434, May 2000.

[57] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike-based event processing vision systems," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 53, pp. 2548–2566, Dec 2006.

[58] L. Camunas-Mesa, C. Zamarreno-Ramos, A. Linares-Barranco, A. J. Acosta-Jimenez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-kernel convolution processor module for event-driven vision sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, pp. 504–517, Feb 2012.

[59] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow : Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[60] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7 : A matlab-like environment for machine learning."

[61] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe : Convolutional architecture for fast feature embedding," *arXiv preprint arXiv :1408.5093*, 2014.

[62] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60, pp. 53–60, Perth, Australia, 1995.

[63] Itseez, *The OpenCV Reference Manual*, 2.4.9.0 ed., April 2014.

[64] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1058–1066, 2013.

[65] J. Costas-Santos, T. Serrano-Gotarredona, R. Serrano-Gotarredona, and B. Linares-Barranco, "A spatial contrast retina with on-chip calibration for neuromorphic spike-based aer vision systems," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 54, pp. 1444–1458, July 2007.

[66] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A five-decade dynamic-range ambient-light-independent calibrated signed-spatial-contrast aer retina with 0.1-ms latency and optional time-to-first-spike mode," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 57, pp. 2632–2643, Oct 2010.

[67] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[68] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet : Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[69] V. E. Beneš, "Permutation groups, complexes, and rearrangeable connecting networks," *The Bell System Technical Journal*, vol. 43, pp. 1619–1640, July 1964.

[70] A. Waksman, "A permutation network," *J. ACM*, vol. 15, pp. 159–163, Jan. 1968.

[71] C. Chang and R. Melhem, "Arbitrary size benes networks," *Parallel Processing Letters*, vol. 07, no. 03, pp. 279–284, 1997.

[72] B. BEAUQUIER and E. DARROT, "On arbitrary size waksman networks and their vulnerability," *Parallel Processing Letters*, vol. 12, no. 03n04, pp. 287–296, 2002.

[73] J. Lenfant, "Parallel permutations of data : A benes network control algorithm for frequently used permutations," *IEEE Transactions on Computers*, vol. C-27, pp. 637–647, July 1978.

[74] D. Nassimi and S. Sahni, "Parallel algorithms to set up the benes permutation network," *IEEE Transactions on Computers*, vol. C-31, pp. 148–154, Feb 1982.