Approches connexionnistes pour la vision par ordinateur embarquée

Robin Danilo

THESE DE DOCTORAT DE

L'UNIVERSITE BRETAGNE SUD
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Electronique*

Par
« Robin DANILO »

« Approches connexionnistes pour la vision par ordinateur embarquée »

Thèse présentée et soutenue à « Lorient », le « 18 décembre 2018 »
Unité de recherche : Lab-STICC
Thèse N° : 518

Rapporteurs avant soutenance :

Damien QUERLIOZ – Chargé de recherche CNRS HDR – Université Paris Sud – Centre de Nanosciences et Nanotechnologies
Michel PAINDAVOINE – Professeur – Université de Bourgogne – LEAD

Composition du Jury :

Président : **Bernard GIRAU** – Professeur – Université de Lorraine – LORIA

Examinateurs : **Damien QUERLIOZ** – Chargé de recherche CNRS HDR – Université Paris Sud – Centre de Nanosciences et Nanotechnologies
Michel PAINDAVOINE – Professeur – Université de Bourgogne – LEAD
Benoît MIRAMOND – Professeur – Université de Nice – LEAT

Dir. de thèse : **Philippe COUSSY** – Professeur – Université Bretagne Sud – Lab-STICC

Co-dir. : **Laura CONDE CANENCIA** – Professeur – Université Bretagne Sud – Lab-STICC
Vincent GRIPON – Chargé de Recherche – IMT Atlantique

Titre : Approches connexionnistes pour la vision par ordinateur embarquée

Mots clés : Réseaux de neurones, traitement d'image, systèmes embarqués, vision par ordinateur

Résumé : Pour concevoir des systèmes de vision embarquée, deux axes peuvent être considérés. Le premier se focalise sur la conception de nouveaux dispositifs numériques plus puissants capables de mettre en œuvre de manière efficace des algorithmes complexes. Le second se concentre sur l'élaboration de nouveaux algorithmes de vision, moins gourmands en ressources et qui peuvent efficacement être mis en œuvre sur des systèmes numériques embarqués. Nous privilégions dans ces travaux le second axe avec comme approche l'utilisation de modèles connexionnistes. Parmi les différents modèles existants, nous nous intéressons à deux modèles de réseaux de neurones artificiels, les réseaux à clusters et les réseaux convolutifs.

Le premier modèle que nous utilisons, appelé réseau à clusters, n'avait jamais été utilisé pour réaliser des tâches de vision par ordinateur. Cependant, il paraissait être un bon candidat pour être utilisé sur des systèmes embarqués, notamment par des mises en œuvre sur des architectures matérielles dédiées. L'objectif a été tout d'abord de trouver les types de tâches pouvant être réalisées à l'aide de ce modèle de réseau. Ce modèle a été conçu pour réaliser des mémoires associatives. En vision par ordinateur, cela peut se rapprocher de problèmes tels que la recherche d'images par le contenu. Ce type d'application utilise massivement des algorithmes de recherche de plus proches voisins approchée et c'est donc sur ce type de tâches que nous nous sommes concentrés.

Le second type de réseau étudié appelé réseau convolutif, est lui très populaire pour concevoir des systèmes de vision par ordinateur. Notre objectif a été ici de trouver des manières de simplifier ces réseaux tout en conservant des performances élevées. Nous proposons notamment une technique qui consiste à ré-entrainer des réseaux quantifiés.

Title : Connectionist approaches for embedded computer vision

Keywords : Neural network, image processing, embedded systems, computer vision

Abstract : To design embedded computer vision systems, two axes can be considered. The first focuses on designing new, more powerful, digital devices that can efficiently implement complex algorithms. The second targets the development of new, lightweight computer vision algorithms that can be effectively implemented on digital embedded systems. In this work, we favor the second axis by using connectionist models. In this context, we focus on two models of artificial neural networks: cluster-based networks and convolutional networks.

The first model we use, i.e. cluster-based network, was never been used to perform computer vision tasks before. However, it seemed to be a good candidate to design embedded systems, especially through dedicated hardware architectures implementation. The goal was first to find out the kinds of tasks that could be performed using this network model. This model has been designed to implement associative memories which can come close to problems such as content-based image retrieval in computer vision domain. This type of application massively uses approximated nearest neighbor search algorithms which makes it a good candidate to focus on.

The second type of network studied in this work, called convolutional network, is very popular to design computer vision systems. Our goal here was to find different ways to simplify their complexity while maintaining high performance. In particular, we proposed a technique that involves re-training quantified networks.

Remerciements

Je souhaite tout d'abord remercier Philippe pour l'aide et le soutien qu'il m'a apporté pendant toute la durée de cette thèse. Un grand merci également à Vincent et Laura qui n'ont pas été avares de conseils. Merci à toute l'équipe du Lab-STICC pour la bonne ambiance qui règne à Lorient. Je tiens aussi à remercier Thomas de m'avoir accueilli pendant trois mois dans son laboratoire à Providence. Enfin, tout mon amour à Julie pour m'avoir encouragé tout au long de ces années.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Avant propos

1.1.1 Vision par ordinateur embarquée

Dès les débuts de l'informatique, approcher l'intelligence humaine à l'aide d'un système numérique est apparu comme un objectif à atteindre [1]. Cela concerne notamment les capacités cognitives de haut niveau telles que le raisonnement critique, l'abstraction ou la créativité ; mais également la capacité des humains à percevoir leur environnement à partir de données complexes telles que les informations sensorielles. La perception sensorielle consiste à capter ces informations et à les traiter pour offrir une représentation cohérente de l'environnement dans lequel nous évoluons. Parmi les différentes informations sensorielles, l'être humain est particulièrement performant pour traiter l'information visuelle. Une courte séquence d'images est par exemple suffisante à une personne pour qu'elle puisse rapidement identifier les objets présents, les positionner dans l'espace et analyser leurs mouvements. Il n'est donc pas étonnant que la vision par ordinateur soit

actuellement l'un des domaines les plus actifs de la recherche en informatique.

Si les fonctions visuelles telles que l'analyse des mouvements ou la reconnaissance des objets sont réalisées sans effort par les humains, ces fonctions sont extrêmement complexes à mettre en œuvre à l'aide d'un système numérique. Au cours de ces dernières années, même si les performances humaines n'ont pas été atteintes, d'importants progrès ont été faits en vision par ordinateur. Ces progrès sont d'une part dus à l'émergence de nouveaux algorithmes de vision toujours plus performants, d'autre part, au développement de composants numériques tel que les GPUs (*Graphics Processing Units*) qui permettent de les exécuter. Ces composants offrent en effet d'importantes puissances de calcul, nécessaires à la plupart de ces algorithmes. Ils ne sont cependant pas adaptés pour des applications mobiles qui disposent de ressources limitées : mémoires, calcul, surface et énergie.

La conception de systèmes de vision capables d'interpréter le contenu des images captées et pouvant être contenus dans notre poche est donc un défi majeur pour les chercheurs en systèmes embarqués et en vision par ordinateur. De tels systèmes peuvent être utilisés dans une grande gamme d'applications telles que la navigation autonome ou les interactions homme machine. Dans cette thèse, nous nous intéressons à la conception de systèmes de vision embarquée à l'aide de modèles connexionistes, c'est-à-dire mettant en œuvre des réseaux de neurones artificiels.

1.1.2 Connexionisme

Le connexionisme est une approche utilisée dans différents champs de recherche allant de la psychologie à l'informatique. Les modèles connexionistes

utilisent des réseaux de neurones artificiels qui s'inspirent de l'organisation et du fonctionnement des systèmes neuronaux biologiques. Ces modèles ont tout d'abord été utilisés dans les sciences cognitives pour modéliser et étudier des fonctions telles que la mémoire, la perception ou l'attention. Leurs mises en œuvre sur des systèmes informatiques a rapidement orienté leur utilisation dans un champ purement applicatif. Ils sont actuellement utilisés dans des applications de classification d'images [2] ou de traduction automatique [3].

Un réseau de neurones artificiels consiste en un ensemble d'unités élémentaires de calcul reliées les unes aux autres par des connexions pondérées appelées *poids*. L'état du système est caractérisé par l'état de chacun des neurones qui le composent. L'état d'un neurone dépend de l'état des neurones auxquels il est connecté ainsi que des poids de ses connexions. Un réseau de neurones artificiels est également défini par son mécanisme d'apprentissage, c'est-à-dire la méthode utilisée pour mettre à jour les poids des connexions. Suite au premier modèle de réseau de neurones artificiels, le *perceptron*, proposé par le psychologue Frank Rosenblatt [4], il existe maintenant une large gamme de réseaux de neurones différents.

1.2 Objectifs

Pour concevoir des systèmes de vision embarquée, deux axes orthogonaux peuvent êtres considérés. Le premier se focalise sur la conception de nouveaux appareils numériques capables de mettre en œuvre de manière efficace des algorithmes complexes. Le second se concentre sur l'élaboration de nouveaux algorithmes de vision, moins gourmands en ressources et qui peuvent efficacement être mis en œuvre sur des systèmes numériques embarqués. Nous nous intéressons ici au second axe avec comme approche l'utilisation de modèles connexionistes.

Parmi les différents modèles existants, nous nous intéressons à deux modèles de réseaux de neurones artificiels, les réseaux à clusters et les réseaux convolutifs.

1.2.1 Réseaux à clusters

Le premier modèle que nous utilisons, appelé réseau à clusters, n'avait jamais été utilisé pour réaliser des tâches de vision par ordinateur. Cependant, il paraissait être un bon candidat pour être utilisé sur des systèmes embarqués, notamment par des mises en œuvre sur des architectures matérielles dédiées. L'objectif a été tout d'abord de trouver les types de tâches pouvant être réalisées à l'aide de ce modèle de réseau. Ce modèle a été conçu pour réaliser des *mémoires associatives*. De manière synthétique, une mémoire associative est une mémoire dont l'accès aux données ne se fait pas à l'aide d'une adresse mais par les données elles-mêmes. En vision par ordinateur, cela peut se rapprocher de problèmes tels que la recherche d'images par le contenu. Ce type d'application utilise massivement des algorithmes de recherche de plus proches voisins approchée et c'est donc sur ce type de tâches que nous nous sommes concentrés.

Projet SENSE

Les travaux portant sur les réseaux à clusters s'inscrivent dans un projet de recherche du labex CominLabs [5] appelé SENSE (Sparse Neural Coding and Bionic Vision System). Ce projet avait pour ambition de concevoir un système de vision neuro-inspiré, basé entièrement sur ce modèle connexioniste. Allant de l'acquisition de l'information jusqu'à des traitements de haut niveau, ce système devait être constitué de différentes couches technologiques. La première couche analogique devait être responsable de l'acquisition de l'information visuelle et être

composée d'une rétine artificielle ainsi que d'une mise en œuvre analogique d'un réseau à cluster. La deuxième couche devait mettre en œuvre des réseaux à clusters sur des architectures matérielles de type FPGA (*Field Programmable Gate Array*) et être responsable du traitement de l'information de bas niveau. Enfin, la dernière couche logicielle devait être en charge de l'analyse des informations extraites. Avant le projet SENSE, la majeure partie des travaux portant sur ce modèle a considéré son utilisation uniquement dans un cadre théorique.

1.2.2 Réseaux convolutifs

Le second type de réseau étudié appelé réseau convolutif, est lui très populaire pour concevoir des systèmes de vision par ordinateur. Depuis 2012, il est même l'approche privilégiée par la communauté scientifique pour des applications telles que la classification d'images, application pour laquelle ils offrent de meilleurs résultats qu'un humain. Ce type de réseau a largement bénéficié du développement des GPUs. En effet, ce type de réseau est connu pour consommer un grand nombre de ressources, aussi bien en termes de calcul qu'en termes de mémoire. Pour ces raisons, il reste encore peu utilisé pour des applications embarquées. Notre objectif a été ici de trouver des manières de simplifier ces réseaux tout en conservant des performances élevées. Nous proposons notamment une technique qui consiste à ré-entraîner des réseaux quantifiés.

1.3 Plan de la thèse

Cette thèse est organisée de la manière suivante. Dans le premier chapitre, nous présenterons différents modèles connexionnistes de mémoire associative et plus particulièrement le réseau à clusters. Nous réaliserons également une étude

des différentes évolutions que ce modèle a connu.

Le second chapitre est consacré à l'utilisation des réseaux à clusters pour la recherche de plus proche voisin dans l'espace euclidien et l'espace de Hamming. Cette tâche est massivement utilisée pour résoudre des problèmes de vision par ordinateur tels que la recherche d'images par le contenu ou la classification non-paramétrique. Les réseaux à clusters manipulent des vecteurs ayant une structure particulière. Cette structure ne correspond pas aux vecteurs couramment utilisés en vision par ordinateur. Nous proposerons dans ce chapitre une solution permettant de stocker des vecteurs binaires ou réels dans ce type de réseau ainsi que des modifications à l'algorithme de décodage. Ces modifications permettent de retrouver le plus proche voisin d'un vecteur requête fourni en entrée.

Dans le troisième chapitre, nous nous intéresserons au stockage de vecteurs corrélés dans les réseaux à clusters. Ces réseaux offrent une grande capacité de stockage lorsque les vecteurs sont uniformément et indépendamment distribués. Cette capacité diminue lorsque ces deux conditions ne sont pas respectées, ce qui est le cas pour des vecteurs issus de données naturelles, extraits d'images par exemple. Afin de répondre au problème de stockage de vecteurs corrélés, nous proposons une version hétéro-associative du réseau à clusters appelée réseau à clusters restreint.

Dans le quatrième chapitre, nous proposerons d'utiliser des techniques de quantification vectorielle afin d'accélérer et de compresser des réseaux convolutifs. Nous montrerons également comment l'algorithme de rétropropagation du gradient peut être modifié afin de ré-entraîner un réseau convolutif quantifié.

Chapitre 2

Introduction aux réseaux à clusters

Le réseau à clusters est un modèle connexioniste de mémoire associative proposé par Gripon et Berrou [6]. Ce modèle utilise des neurones et des connexions binaires, permettant ainsi des mises en oeuvre sur des architectures numériques dédiées particulièrement efficaces. Dans une première partie, nous présenterons les modèles connexionnistes de mémoires associatives les plus courants et plus particulièrement le modèle de Willshaw [7] dont sont inspirés les réseaux à clusters. La seconde partie de ce chapitre sera consacrée aux réseaux à clusters. Dans la troisième partie, nous présenterons différentes architectures matérielles mettant en œuvre des réseaux à clusters. Enfin, dans la dernière partie nous proposerons une étude comparative des différentes évolutions des réseaux à clusters.

2.1 Modèles connexionistes de mémoires associatives

2.1.1 Formulation

De manière formelle, une mémoire associative est un système permettant de stocker M couples de vecteurs $(\mathbf{x}^z, \mathbf{y}^z)$ avec $z \in [1, ..., M]$. Une fois les couples stockés, un tel système est capable de retrouver $\mathbf{y}^z$ à partir de $\tilde{\mathbf{x}}^z$ une version bruitée de $\mathbf{x}^z$. On distingue deux types de mémoire associative. Les mémoires **hétéro-associatives**, pour lesquelles $\mathbf{x}^z \neq \mathbf{y}^z$ et les mémoires **auto-associatives**, pour lesquelles $\mathbf{x}^z = \mathbf{y}^z$. Une méthode naïve pour concevoir une mémoire associative consiste à stocker les données sous la forme d'une liste. La recherche se fait alors de manière linéaire en comparant l'entrée avec chaque élément de la liste. Une approche plus intéressante consiste à utiliser un modèle connexioniste, c'est-à-dire un réseau de neurones.

Un réseau de neurones hétéro-associatif est composé de deux couches de neurones : une couche d'entrée et une couche de sortie. Le nombre de neurones dans la couche d'entrée et dans la couche de sortie est égal respectivement au nombre de dimensions des vecteurs $\mathbf{x}^z$ et $\mathbf{y}^z$. Chaque neurone de la couche d'entrée est connecté avec l'ensemble des neurones de la couche de sortie. Ces connexions sont regroupées dans une matrice W, cette matrice est calculée lors de la phase d'apprentissage. L'état du réseau est donné par les deux vecteurs d'état $\mathbf{e}_e$ et $\mathbf{e}_s$ correspondant respectivement à l'état des neurones de la couche d'entrée et de sortie. Lors de la phase de décodage, l'état des neurones de la couche d'entrée est initialisé à l'aide du vecteur dégradé fourni en entrée $\mathbf{e}_e = \tilde{\mathbf{x}}^z$. L'état des neurones de la couche de sortie est alors calculé à partir de $\mathbf{e}_e$ et W. Un réseau de neurones

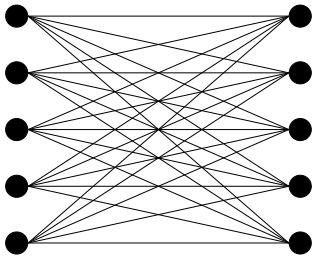


FIGURE 2.1 – Représentation graphique d'un réseau hétéro-associatif.

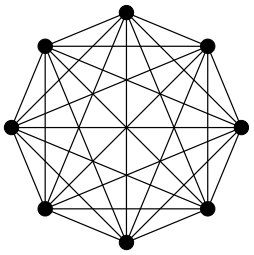


FIGURE 2.2 – Représentation graphique d'un réseau auto-associatif.

auto-associatif est composé d'une seule couche entièrement interconnectée qui sert d'entrée et de sortie au système. L'état du réseau est alors donné par un unique vecteur d'état $\mathbf{e}$. Les figures 2.1 et 2.2 présentent ces deux types de réseau. La majorité des réseaux associatifs sont utilisés pour stocker des vecteurs binaires, codés soit de manière binaire, $e_j \in \{0, 1\}$, soit de manière polaire, $e_j \in \{-1, 1\}$.

2.1.2 Modèles à pondération à valeurs réelles

Différents modèles de réseaux de neurones associatifs ont été proposés. La majorité d'entre eux utilisent des connexions à pondération à valeurs réelles. Chaque connexion $w_{i,j}$ reliant le neurone i avec le neurone j appartient à l'ensemble des réels, i.e. $\mathbb{R}$. Parmi ces modèles, on peut citer les matrices de corrélation [8]. Ces réseaux peuvent être hétéro ou auto-associatifs et utilisent la règle de Hebb [9] pour trouver W. Cette règle consiste à renforcer le poids d'une connexion lorsque les deux neurones qu'elle connecte sont excités conjointement. Pour un réseau hétéro-associatif, cette règle s'applique de la manière suivante :

$$w_{ij}^{z} = w_{ij}^{z-1} + x_i^z y_j^z, \tag{2.1}$$

avec w_{ij}^z égal au poids de la connexion w_{ij} lors de l'apprentissage du couple $(\mathbf{x}^z, \mathbf{y}^z)$, w_{ij}^0 étant initialisé à zéro. Considérant les matrices X et Y, dont chaque

ligne z de X est égale à $\mathbf{x}^z$ et chaque ligne z de Y est égale à $\mathbf{y}^z$. La règle de Hebb peut alors s'écrire à l'aide de l'équation matricielle suivante :

$$W = X^\intercal Y = \sum_{z=1}^{M} \mathbf{x}^{z\intercal} \mathbf{y}^z \tag{2.2}$$

le symbole $^\intercal$ étant utilisé pour la transposée d'une matrice ou d'un vecteur.

Lors de la phase de décodage, après avoir initialisé l'état des neurones d'entrée $\mathbf{e}_e$, l'état des neurones de sortie $\mathbf{e}_s$ est calculé de la manière suivante :

$$\mathbf{e}_s = f(\mathbf{s}) = f(\mathbf{e}_e W), \tag{2.3}$$

avec f une fonction d'activation. Pour des vecteurs polaires, on utilisera par exemple la fonction *signe*. Les matrices de corrélation offrent des performances optimales lorsque les vecteurs $\mathbf{x}^z$ sont orthogonaux deux à deux. Des vecteurs presque orthogonaux peuvent être obtenus à partir d'une distribution uniforme sur un hypercube non trivial de centre le vecteur nul. Pour favoriser l'orthogonalité des vecteurs, on privilégie le codage polaire de ces derniers.

Lorsque les vecteurs ne sont pas orthogonaux deux à deux, la méthode de Hebb peut être remplacée par une méthode utilisant une matrice pseudo-inverse [10]. Une règle d'apprentissage a pour objectif de trouver W tel que $XW = Y$ et donc de minimiser l'erreur quadratique $\|XW - Y\|^2$. Ce minimum est atteint pour $W = X^+ Y$, avec X^+ la matrice pseudo-inverse de X. Cette méthode offre de meilleurs résultats que ceux obtenus avec la règle de Hebb, cependant elle est moins flexible. En effet, la totalité des vecteurs doit être connue au début de l'apprentissage.

L'un des réseaux de neurones associatifs les plus connus est sans doute le réseau de Hopfield [11]. Un réseau de Hopfield est un réseau auto-associatif dans lequel un neurone ne peut être connecté avec lui même (les valeurs de la diagonale

de W sont égales à zéro). L'apprentissage se fait à l'aide de la règle de Hebb (équation 2.1). La principale différence entre le réseau de Hopfield et les matrices de corrélation auto-associatives, est que le réseau de Hopfield utilise un algorithme de décodage itératif. Après avoir initialisé l'état du réseau avec le vecteur fourni en entrée, chaque itération calcule un nouvel état pour le réseau jusqu'à atteindre un état stable :

$\mathbf{e}(0) = \tilde{\mathbf{x}}^i$;

$it = 1$;

do

  $\mathbf{e}(it) = f(\mathbf{e}(it-1)W)$;

  $it = it + 1$;

while $\underline{\mathbf{e}(it) \neq \mathbf{e}(it-1)}$;

Le réseau de Hopfield fut étendu aux réseaux hétéro-associatifs par Kosko [12]. Dans les réseaux de Kosko, après avoir calculé $\mathbf{e}_s$ à l'aide de l'équation 2.3, $\mathbf{e}_e$ est calculé à partir de $\mathbf{e}_s$ et ainsi de suite. De manière générale, les réseaux à décodage itératif tels que les modèles de Hopfield [11] et de Kosko [12] permettent de stocker et de retrouver un plus grand nombre de vecteurs et sont moins sensibles au bruit. Ce gain est fait au détriment de la vitesse de décodage.

2.1.3 Le réseau de Willshaw

Parallèlement aux modèles à pondération, Willshaw propose un modèle connexionniste de mémoire associative utilisant des connexions binaires [7]. Ce modèle retiendra plus particulièrement notre attention car les réseaux à clusters auxquels nous nous intéressons dans cette thèse en sont une dérivée. Plus particulièrement,

nous nous intéresserons ici à la forme auto-associative du réseau de Willshaw.

Formulation

Considérons l'ensemble de vecteurs binaires $\mathcal{X} = \{\mathbf{x}^1, ..., \mathbf{x}^M\}$, avec pour tout $z \in [1, ..., M]$, $\mathbf{x}^z \in \{0, 1\}^d$. Le réseau de Willshaw permettant de stocker $\mathcal{X}$ est alors constitué de d neurones pouvant potentiellement tous s'interconnecter. Ces connexions sont regroupées dans la matrice $W^{d \times d}$ dont chaque élément w_{ij} appartient à l'ensemble $\{0, 1\}$. Les poids w_{ij} étant binaires, on peut considérer que, soit deux neurones sont connectés, soit ils ne le sont pas.

Apprentissage

Pour apprendre ces connexions, on utilise une règle d'apprentissage dérivée de la règle de Hebb qui peut s'écrire de la manière suivante :

$$w_{ij}^z = w_{ij}^{z-1} \vee x_i^z x_j^z \equiv W = \bigvee_{z=1}^{M} \mathbf{x}^{z\intercal} \mathbf{x}^z \tag{2.4}$$

où $\vee$ représente la fonction logique OR. En d'autres termes, cette règle sature les connexions à un. La figure 2.3 représente un réseau de Willshaw après l'apprentissage de trois vecteurs. Chacun d'entre eux contient trois composantes à un.

Dégradations

Une fois la matrice W apprise, ce système est capable de retrouver $\mathbf{x}^z$ lorsque $\tilde{\mathbf{x}}^z$ une version dégradée de $\mathbf{x}^z$ lui est présenté. Les vecteurs étant binaires, nous définissons deux types de dégradation :

— l'**effacement**, une ou plusieurs des composantes de $\mathbf{x}^z$ égales à un ont été mises à zéro ;

$$\mathcal{X} = \left\{ \begin{array}{l} \mathbf{x}^1 = [0,0,0,1,0,1,0,1], \\ \mathbf{x}^2 = [1,0,1,0,0,0,1,0], \\ \mathbf{x}^3 = [0,1,0,0,1,0,0,1] \end{array} \right\}$$

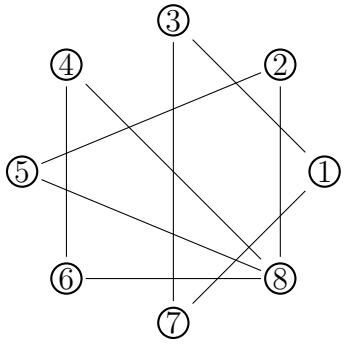


FIGURE 2.3 – Représentation graphique d'un réseau de Willshaw ($d = 8$) après l'apprentissage de trois vecteurs. A gauche les vecteurs, à droite le réseau, seules les connexions $w_{ij} = 1$ sont représentées.

— l'**intrusion**, une ou plusieurs des composantes de $\mathbf{x}^z$ égales à zéro ont été mises à un.

Le type de dégradation va avoir une influence sur le choix de la méthode de décodage. En effet, dans le cas de l'effacement, le processus de décodage va avoir pour objectif de retrouver les éléments manquants, tandis que dans le cas de l'intrusion, l'objectif est d'effacer les intrus. Ces deux types de dégradation peuvent être appliqués simultanément pour produire le vecteur $\tilde{\mathbf{x}}^i$, auquel cas le processus de décodage devra satisfaire ces deux objectifs.

Processus de décodage

Lors de la phase de décodage, le réseau est initialisé à l'aide du vecteur fourni en entrée $\mathbf{e}(0) = \tilde{\mathbf{x}}^z$. Puis un nouvel état du réseau est calculé de la manière suivante :

$$\mathbf{e}(it) = f(\mathbf{s}(it)) = f\left(\mathbf{e}(it-1)W\right), \tag{2.5}$$

avec $\mathbf{s}$ un vecteur de score de dimension d et f une fonction à seuil :

$$f(x) = \begin{cases} 0 \text{ si } x < \theta, \\ \\ 1 \text{ si } x \geq \theta \end{cases} . \tag{2.6}$$

Ce processus peut être itéré pour améliorer le résultat. Si le nombre M de vecteurs stockés et la dégradation ne sont pas trop grands, alors $\mathbf{e}(\text{fin}) = \mathbf{x}^z$ et le décodage est réussi.

Choix du seuil θ

Le choix du seuil θ est déterminant sur les performances et dépendra du type d'application. Nous allons maintenant énoncer deux méthodes utilisées pour trouver ce seuil. La méthode proposée par Willshaw [7] est de donner comme valeur à θ le poids du vecteur d'état $\mathbf{e}$ (le nombre de composantes égales à un) :

$$\theta(it) = \sum_{i=1}^{d} e_i(it-1). \tag{2.7}$$

Cette méthode donne de bons résultats pour des dégradations de type effacement lorsqu'une seule itération est utilisée pour le décodage. Cependant, dans le cas de dégradations de type intrusion, le poids du vecteur d'entrée est plus grand que le score des neurones appartenant au vecteur stocké. En reprenant l'exemple de la figure 2.3 et si $\mathbf{e}(0) = \tilde{\mathbf{x}}^1 = [0,1,0,1,0,1,0,1]$, les neurones deux, quatre, six et huit sont activés lors de l'initialisation avec le neurone deux comme intrus. A la première itération, le vecteur de score $\mathbf{s}(1)$ est égal à $[0,2,0,3,0,3,0,4]$, en fixant $\theta = 4$ le poids du vecteur $\mathbf{e}(0)$, le nouvel état du réseau est alors $\mathbf{e}(1) = [0,0,0,0,0,0,0,1]$.

Afin de régler les problèmes d'intrusion, une seconde méthode consiste à activer les neurones ayant les k meilleurs scores. Le seuil est fixé de la manière

suivante :

$$\theta(it) = \hat{s}_k, \text{ avec } \hat{\mathbf{s}} = sortDec(\mathbf{s}(it)), \tag{2.8}$$

avec $sortDec$ une fonction de tri décroissant. Si l'on applique cette règle à l'exemple précédent, $sortDec(\mathbf{s}(1)) = [4, 3, 3, 2, 0, 0, 0, 0]$. Pour $k = 1$, le seuil θ est égal à quatre et comme pour la règle de Willshaw le décodage échoue. Pour $k = \{2, 3\}$, le seuil θ est égal à trois, le nouvel état du réseau est alors $\mathbf{e}(1) = [0, 0, 0, 1, 0, 1, 0, 1]$ et le décodage est réussi. Si $k = 4$, le seuil θ est égal à deux et l'état du réseau ne change pas. Si k est bien choisi, cette règle permet de répondre au problème d'intrusion. Une bonne valeur pour k est le poids du vecteur $\mathbf{x}^z$ que l'on cherche à retrouver (ici le poids de $\mathbf{x}^1$ est égal à trois). Il est donc préférable de stocker dans le réseau des vecteurs de même poids.

Dans la littérature sur les réseaux de Willshaw [13], on nomme cette règle *l-max*. Elle sera toutefois nommée dans cette thèse k-Global-Winner-take-All (k-GWtA) afin de correspondre au vocabulaire utilisé pour les réseaux à clusters.

Parcimonie

Un des autres paramètres importants pour les réseaux de Willshaw est la densité de W. Nous appelons densité le nombre de valeurs à un dans la matrice divisée par la taille de la matrice. La densité augmente avec M, le nombre de vecteurs stockés et p, le poids des vecteurs stockés. Afin de stocker un grand nombre de vecteurs, il est donc important que ces derniers soient parcimonieux ($p \ll d$ avec d la taille des vecteurs) [14].

2.2 Les réseaux à clusters

Le réseau à clusters proposé par Gripon et Berrou [6] est un nouveau modèle de réseau de neurones auto-associatif dérivé du modèle de Willshaw [7]. Comme ce dernier, un réseau à clusters utilise des neurones et des connexions binaires. À la différence d'un réseau de Willshaw, une structure particulière est imposée au réseau. Cette structure permet d'utiliser des processus de décodage plus robustes.

2.2.1 Structure du réseau et des vecteurs

Un réseau à clusters équivaut à un réseau de Willshaw auto-associatif pour lequel les d neurones sont organisés en c clusters de taille $\ell = d/c$ (considérant que d est un multiple de c). Chaque neurone est identifié à l'aide d'un couple d'index (i, j) où $i \in [1, ..., c]$ est l'index du cluster auquel il appartient et $j \in [1, ..., \ell]$, est utilisé pour identifier le neurone à l'intérieur du cluster i. Chaque neurone (i, j) peut être connecté à l'aide d'une connexion binaire $w_{(i,j)(i',j')}$ avec n'importe quel neurone (i', j') situé dans un autre cluster $(i \neq i')$.

Afin d'être stockés dans un réseau à clusters, les vecteurs $\mathbf{x}$ ont eux-même une structure particulière. Cette structure est la suivante :

— chaque vecteur $\mathbf{x}^z$ est divisé en c sous-vecteurs $\mathbf{x}_i^z$ de dimension ℓ,

— chaque sous-vecteur $\mathbf{x}_i^z$ possède au maximum une seule composante à un,

l'équation suivante résume les différentes notations utilisées dans ce chapitre pour les vecteurs $\mathbf{x}^z$:

$$\mathbf{x}^z = [\mathbf{x}_1^z, ..., \mathbf{x}_c^z] = [x_{(1,1)}^z, ..., x_{(i,j)}^z, ..., x_{(c,\ell)}^z] \equiv I(\mathbf{x}^z) = [I_1^z, ..., I_c^z] \tag{2.9}$$

avec I_i^z l'index de la composante à un dans le sous-vecteur $\mathbf{x}_i^z$. La structure des vecteurs a pour conséquence l'activation, au maximum, d'un unique neurone dans

$$X = \left\{ \begin{array}{l} \mathbf{x}^1 = [0001|1000|0010|0100] \\ \mathbf{x}^2 = [0010|0001|1000|0001] \\ \mathbf{x}^3 = [1000|0010|0001|1000] \end{array} \right\}$$

$$I = \left\{ \begin{array}{l} I(\mathbf{x}^1) = [4, 1, 3, 2] \\ I(\mathbf{x}^2) = [3, 4, 1, 4] \\ I(\mathbf{x}^3) = [1, 3, 4, 1] \end{array} \right\}$$

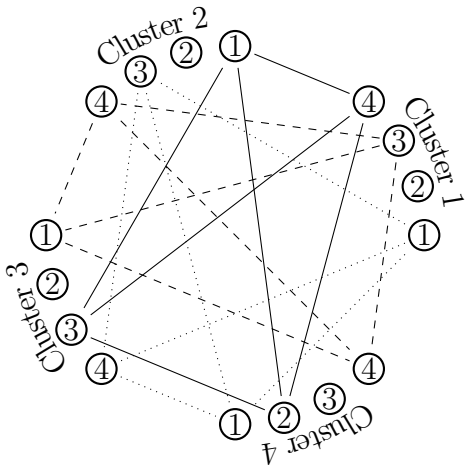


FIGURE 2.4 – Représentation graphique d'un réseau à clusters ($d = 16$, $c = 4$) après l'apprentissage de trois vecteurs. A gauche les vecteurs, à droite le réseau, seules les connexions $w_{(ij)(i'j')} = 1$ sont représentées.

chaque cluster lors de la phase de décodage. Cette caractéristique sera notamment utilisée pour rendre plus robuste le processus de décodage.

On distingue deux types de vecteurs $\mathbf{x}$. Les vecteurs dits **pleins** pour lesquels chaque sous-vecteur $\mathbf{x}_i$ possède une composante égale à un : ces vecteurs ont un poids égal à c. L'équation suivante présente un exemple de vecteur plein ($d = 16$ et $c = 4$) :

$$[0001|0100|0100|1000] \equiv [4, 2, 2, 1]. \tag{2.10}$$

Les vecteurs dits **creux** [15] pour lesquels certains des sous-vecteurs $\mathbf{x}_i$ ne possèdent pas de composante égale à un : ces vecteurs ont un poids inférieur à c. L'équation suivante présente un exemple de vecteur creux dont le troisième sous-vecteur ne possède pas de composante à un ($d = 16$ et $c = 4$) :

$$[0100|0001|0000|0001] \equiv [2, 4, \_, 4], \tag{2.11}$$

dans la représentation $I(\mathbf{x}^z)$, le symbole _ est utilisé pour indiquer l'absence de composante à un. Dans les deux cas, le stockage se fait de la même manière que

pour les réseaux de Willshaw en utilisant l'équation 2.4 pour trouver W. La figure 2.4 représente de manière graphique un réseau à clusters pour $d = 16$ et $c = 4$ dans lequel trois vecteurs pleins sont stockés.

2.2.2 Processus de décodage

Lors du décodage, l'objectif est de retrouver $\mathbf{x}^z \in \mathcal{X}$ à partir de $\tilde{\mathbf{x}}^z$. Après avoir initialisé le réseau, $\mathbf{e}(0) = \tilde{\mathbf{x}}^z$, un processus de décodage itératif est utilisé pour retrouver $\mathbf{x}^z$. Chaque itération it permet de calculer un nouvel état $\mathbf{e}(it)$ pour le réseau. Dans la littérature sur les réseaux à clusters, une itération de décodage est divisée en deux étapes :

1. l'application d'une règle dynamique permettant de calculer à partir de $\mathbf{e}(it-1)$ un score $s(it)_{(i,j)}$ pour chacun des neurones (i, j),
2. l'application d'une règle d'activation permettant de calculer un nouveau vecteur d'état $\mathbf{e}(it)$ à partir de $\mathbf{s}(it)$.

Si cette terminologie est appliquée au réseau de Willshaw (voir équation 2.5), la règle dynamique correspond au produit matriciel et la règle d'activation aux différentes fonctions à seuil (Willshaw, GWtA, k-GWtA).

Règles dynamiques

Deux règles dynamiques sont couramment utilisées. La première, la règle du **Sum-of-Sum** (SoS) initialement proposée dans [6], est équivalente au produit matriciel eW :

$$\forall i, j, s_{(i,j)} = \sum_{i'=1}^{c} \sum_{j'=1}^{\ell} \left(e_{(i',j')} w_{(i,j)(i',j')} \right) \tag{2.12}$$

Si cette règle offre de bonnes performances pour des dégradations de type effacement, elle ne permet pas (ou peu) de répondre à des dégradations de type

intrusion.

Afin de tirer parti de la structure particulière du réseau, une seconde règle, nommée **Sum-of-Max** (SoM), a été proposée [16]. La règle SoM exploite le fait que chaque sous-vecteur $\mathbf{x}_i$ possède au maximum une seule composante à un (un neurone actif par cluster i). Donc la contribution du cluster i' au calcul du score $s_{(i,j)}$ (avec $i' \neq i$), ne peut dépasser un. La règle SoM s'écrit de la manière suivante :

$$\forall i, j, s_{(i,j)} = \sum_{i'=1}^{c} \max_{j'=1}^{\ell} \left(e_{(i',j')} w_{(i,j)(i',j')} \right) \tag{2.13}$$

Règles d'activation

Comme pour le réseau de Willshaw, les règles d'activation utilisées pour les réseaux à clusters reviennent à trouver un seuil θ permettant d'activer les neurones dans le réseau. Les premiers travaux portant sur les réseaux à clusters [6, 16] ne considéraient que des vecteurs pleins. Dans le cas de vecteurs pleins, tous les vecteurs $\mathbf{x}^z$ stockés dans le réseau possèdent une composante à un dans chaque sous-vecteur $\mathbf{x}_i^z$. Au lieu d'utiliser un seuil θ, global à tous les neurones, il est possible de déterminer un seuil θ^i, local à chaque cluster i. Le score maximal du cluster i est choisi pour θ^i :

$$\forall i \in [1, ..., c], \theta^i = \max_{j=1}^{\ell} s_{(i,j)}, \tag{2.14}$$

cette règle est appelée Local-Winner-take-All (LWtA).

Lorsqu'un réseau à clusters est utilisé pour stocker des vecteurs creux [15], certains clusters ne possèdent pas de neurone actif, la règle LWtA ne peut donc pas être utilisée. La règle proposée dans [15] est celle du k-GWtA (voir section 2.1.3) avec $k = 1$. Dans [17], les auteurs étudient différentes règles d'activation pour les réseaux à clusters lorsque des vecteurs creux sont stockés. Ils proposent également

une nouvelle règle d'activation appelée Global-Loser-Kicked-Out (GLkO).

La règle GLkO fonctionne différemment des règles discutées précédemment et est utilisée pour de dégradations des type intrusion. Dans le contexte de dégradations de type intrusion, certaines composantes égales à zéro du vecteur $\mathbf{x}^z$ ont été mises à un pour créer le vecteur $\tilde{\mathbf{x}}^z$. Après avoir initialisé le réseau ($\mathbf{e}(0) = \tilde{\mathbf{x}}^z$), l'objectif est donc de désactiver les neurones erronés. Pour cela, la règle GLkO se concentre sur les scores des neurones actifs à la précédente itération. Considérons $\mathcal{E} = \{(i_1, j_1), ..., (i_m, j_m)\}$ l'ensemble des m neurones actifs et $\mathcal{S} = \{s_{(i_1,j_1)}, ..., s_{(i_m,j_m)}\}$ leurs scores. La règle GLkO met à zéro les neurones de $\mathcal{E}$ ayant un score égal à $\min(\mathcal{S})$.

2.2.3 Stockage des réseaux à clusters

Stocker un réseau à clusters revient à mémoriser sa matrice d'interconnexion $W \in \{0,1\}^{c.\ell \times c.\ell}$. Étant donné que cette matrice est binaire, le coût mémoire total de stockage est donc de $(c.\ell)^2$ bits. Cependant les propriétés du réseau permettent de réduire ce coût. Tout d'abord, si l'on considère les connexions intra-clusters un neurone n'est connecté qu'à lui-même. C'est-à-dire que la sous-matrice $W_{(i,:)((i,:))}$ est égale à une matrice identité de dimension ℓ, il n'est donc pas nécessaire de stocker ces connexions. De plus, étant donné que le réseau est auto-associatif, W est symétrique ($w_{(i,j)(i',j')} = w_{(i',j')(i,j)}$). A partir de ces propriétés le coût total de stockage peut être réduit à :

$$C = \frac{c(c-1)\ell^2}{2}. \tag{2.15}$$

Si le coût minimal de stockage est donné par cette équation, ce mode de stockage est rarement utilisé. En effet, la matrice entière est le plus souvent stockée afin de paralléliser les accès mémoires.

2.3 Architectures matérielles

L'utilisation de poids et de connexions binaires ainsi que la simplicité de l'algorithme de décodage permettent des mises en œuvre matérielles particulièrement performantes. A l'instar du modèle de Willshaw [18], différentes architectures matérielles ont été proposées pour le modèle de réseau à clusters. Nous allons en présenter ici l'évolution.

2.3.1 Première architecture

La première architecture matérielle pour réseaux à clusters a été proposée dans [19]. Dans cet article, les auteurs présentent la mise en œuvre sur FPGA d'un réseau à clusters utilisant la règle Sum-of-Sum (SoS) comme règle dynamique et la règle Local-Winner-take-All (LWtA) comme règle d'activation. Cette mise en œuvre est entièrement parallèle, c'est-à-dire que l'ensemble des communications et des calculs nécessaires pour une itération de l'algorithme de décodage sont faits simultanément en un cycle d'horloge. Pour cela, la totalité de la matrice d'interconnexion W est stockée dans des registres (la symétrie de la matrice n'est pas prise en compte).

L'architecture suit la structure du réseau et attribue à chaque cluster un module. Ce module est composé de deux sous-modules (stockage et décodage) et d'un espace mémoire de taille $\ell^2(c-1)$ stockant les connexions du cluster avec les autres clusters du réseau. Le sous-module de stockage est responsable de la mise à jour des connexions lors de la phase d'apprentissage. Le sous-module de décodage met en œuvre les règles SoS et LWtA. La règle SoS étant mise en oeuvre à l'aide d'un arbre d'additionneurs et la règle LWtA à l'aide d'un opérateur max.

2.3.2 Simplification des calculs

Dans [20] les auteurs reprennent l'architecture proposée dans [19] et présentent une simplification du processus de décodage. Ils proposent une unique règle de décodage utilisant des opérateurs booléens pour déterminer l'état des neurones. Cette règle est la suivante :

$$e_{i,j}^{t+1} = \bigwedge_{i'=1}^{c} \left[\left(\bigvee_{j'=1}^{\ell} e_{i',j'}^{t} \wedge w_{(i,j)(i',j')} \right) \vee \overline{\bigvee_{j'=1}^{\ell} e_{(i',j')}^{t}} \right], \tag{2.16}$$

ici les symboles $\vee$ et $\wedge$ sont utilisés pour les opérateurs booléens OU et ET respectivement. En d'autres termes, pour être activé, un neurone doit être connecté avec au moins un neurone de chaque cluster contenant des neurones actifs. Nous avons montré dans [21] que cette règle est en fait similaire à l'utilisation de la règle dynamique Sum-of-Max (SoM) couplée avec l'utilisation de la règle d'activation Global-Winner-take-All (GWtA) et pouvait être également utilisée lorsque un réseau à clusters stocke des vecteurs creux. Comme pour l'utilisation de ces deux règles, seuls certains types de dégradation peuvent être pris en compte : ceux contenant uniquement des effacements ou uniquement des intrusions. Le processus de décodage est mis en échec par la présence d'effacement et d'intrusion dans un même cluster. L'utilisation de cette règle booléenne permet de grandement diminuer le nombre de ressources de calcul comparée à [19]. En effet l'arbre d'additionneurs est remplacé par un arbre de porte logique ET/OU et l'opérateur max est supprimé. La simplification des opérateurs entraîne également une réduction du chemin critique permettant l'utilisation de fréquences plus élevées. Une approche similaire a été proposée dans [22].

2.3.3 Exploiter la parcimonie

L'un des principaux défauts des architectures entièrement parallèles est qu'elles ne tiennent pas compte de la parcimonie des représentations manipulées dans les réseaux à clusters. En effet, très peu de neurones sont actifs à chaque itération t : le vecteur d'état contient un grand nombre de valeurs nulles. D'autre part, pour que le réseau soit capable de retrouver un vecteur stocké, il est nécessaire que le nombre de connexions à un soit faible.

Parcimonie du vecteur d'état

Dans [23] les auteurs présentent une architecture permettant d'exploiter la parcimonie du vecteur d'état. A chaque itération de l'algorithme de décodage, le calcul des scores est fait en plusieurs cycles d'horloge. A chacun de ces cycles, un neurone actif est sélectionné dans chacun des clusters. Les connexions des neurones sélectionnés sont ensuite chargées et les scores temporaires de la totalité des neurones sont mis à jour. Le nombre de cycles nécessaire pour une itération de l'algorithme de décodage est donc dépendant du nombre maximal de neurones actifs parmi tous les clusters du réseau.

Il n'est plus nécessaire avec cette approche d'accéder à la totalité de la matrice d'interconnexion en parallèle et un mode de stockage moins onéreux que des registres peut donc être utilisé. Les auteurs proposent dans leur mise en œuvre sur FPGA d'utiliser des blocs RAM, permettant ainsi de manipuler des matrices de connexions plus grandes que les précédentes mises en œuvre sur FPGA [19, 20, 22].

Parcimonie de la matrice de connexion

Une matrice parcimonieuse est une matrice contenant un grand nombre de valeurs nulles. Si le nombre de valeurs nulles est suffisamment important, il est possible d'utiliser des structures de données particulières permettant de réduire le coût de stockage. Il existe différentes méthodes permettant d'optimiser le stockage des matrices creuses. L'idée de base de ces méthodes est que seules les valeurs non-nulles nécessitent d'être stockées. Le plus souvent, chaque ligne de la matrice (ou chaque colonne) est représentée à l'aide d'une liste dont chaque élément contient la position d'une valeur non-nulle ainsi que la valeur elle-même.

Pour être capable de retrouver les vecteurs stockés, la matrice de connexion W d'un réseau à clusters doit contenir un faible nombre de connexions. Dans [24], les auteurs tirent parti de la parcimonie de W et proposent à la fois des structures de données spécifiques aux réseaux à clusters ainsi qu'une architecture permettant de manipuler ces structures de données. L'architecture proposée s'inspire de précédents travaux mettant en œuvre des multiplications d'une matrice parcimonieuse.

En plus du gain mémoire obtenu, l'architecture proposée dans [24] est générique. C'est-à-dire qu'elle permet de manipuler des réseaux de tailles variables, ce qui n'était pas le cas des précédentes architectures matérielles.

2.4 Étude des réseaux à clusters

Dans les deux précédentes parties, nous avons présenté les différentes évolutions des réseaux à clusters ainsi que des mises en œuvre de ces réseaux sur des architectures matérielles. Nous proposons maintenant une étude des performances de

ces derniers lorsqu'ils sont utilisés comme mémoire associative. Nous considérons dans cette étude à la fois des dégradations de type effacement et intrusion. Nous comparons dans un premier temps les réseaux à clusters avec les réseaux de Willshaw. Puis nous évaluons les différentes règles dynamiques et règles d'activation qui ont été proposées pour les réseaux à clusters. Pour chacune de ces règles, nous parlons également brièvement du coût qu'elles impliquent lorsqu'elles sont mises en œuvre sur de architectures matérielles. A notre connaissance, c'est la première fois qu'une telle étude est réalisée. Cette étude a pour vocation d'aider les concepteurs d'architectures matérielles lorsqu'ils mettent en œuvre des réseaux à clusters.

2.4.1 Protocole expérimental

Nous utilisons le même protocole expérimental pour l'ensemble des expériences réalisées. Nous utilisons des vecteurs de dimension $d = 2048$ ayant $p = 8$ composantes à un, générées de manière indépendante et uniforme. Une fois les vecteurs stockés, l'un d'entre eux est tiré au sort, il subit une dégradation, puis il est présenté au réseau. L'algorithme de décodage est alors itéré quatre fois pour retrouver le vecteur d'origine. Dans le cas d'effacement, la dégradation consiste à effacer quatre composantes à un du vecteur d'origine. Dans le cas d'intrusion, la dégradation consiste à ajouter huit composantes à un dans le vecteur d'origine. Chaque fois que le réseau ne réussit pas à retrouver le vecteur d'origine est considérée comme un échec. L'opération est répétée un 2000 de fois pour un même nombre M de vecteurs et le taux d'erreur est calculé.

2.4.2 Comparaison avec les réseaux de Willshaw

Un réseau à clusters peut être considéré comme un réseau de Willshaw stockant des vecteurs ayant une structure particulière. Cette structure permet d'utiliser des règles dynamiques et d'activation plus robustes que les précédentes règles proposées pour les réseaux de Wilshaw. Avant d'étudier l'impact de ces règles, nous proposons ici de comparer les deux types de réseau. Pour réaliser cette comparaison, nous générons les vecteurs de trois manières différentes :

— De manière purement aléatoire, les $p = 8$ composantes à un dans un vecteur peuvent prendre n'importe quelle position.

— Sous la forme de vecteur plein. Chaque vecteur peut être vu comme la concaténation de $c = p = 8$ sous-vecteurs, chacun contenant une composante à un.

— Sous la forme de vecteur creux. Chaque vecteur peut être vu comme la concaténation de $c = 2p = 16$ sous-vecteurs, $p = 8$ d'entre eux contiennent une composante à un.

Dans le premier cas, le réseau utilisé pour stocker les vecteurs est un réseau de Willshaw tandis que, dans les deux autres cas, on utilise un réseau à clusters. Pour les deux réseaux nous utilisons comme règle dynamique la règle SoS (voir équation 2.12 et comme règle d'activation la règle k-GWtA (voir paragraphe 2.2.2) avec $k = p = 8$. Ces deux règles sont couramment utilisées pour les réseaux de Willshaw [13] et pour les réseaux à clusters lorsqu'ils sont utilisés pour stocker des vecteurs creux [17].

Nous comparons les réseaux pour des dégradations de type effacement et de type intrusion. Les figures 2.5 et 2.6 rendent compte des performances des trois réseaux pour les deux types de dégradation. Bien que le processus de décodage soit

identique pour chacun des réseaux, on observe que le réseau à clusters stockant des vecteurs pleins offre de meilleures performances. C'est-à-dire que pour une même valeur de M, le taux d'erreur est inférieur à celui des deux autres réseaux. De la même manière, le réseau à clusters stockant des vecteurs creux est légèrement plus efficace qu'un réseau de Willshaw.

Ce premier résultat est très intéressant car la seule différence entre ces réseaux est que pour certains d'entre eux, (les réseaux à clusters) des connexions sont interdites : les connexions entre les neurones appartenant à un même cluster. L'ajout de cette simple contrainte permet alors d'améliorer l'aptitude du réseau à retrouver un vecteur. Une explication possible à ce phénomène est que l'ajout de cette contrainte permet de diminuer l'espace de recherche. En effet, le nombre de vecteurs différents possibles dans le cas du réseau de Willshaw est égal à $\binom{p}{d} = \binom{8}{2048} = 7.57 \times 10^{21}$ tandis que pour le réseau à clusters, le nombre de vecteurs possibles est de $\ell^c = 1.84 \times 10^{19}$. L'autre intérêt des réseaux à clusters est que les règles dynamiques et d'activation peuvent prendre en compte la structure du réseau. Nous allons maintenant étudier ces différentes règles.

2.4.3 Règles dynamiques

Nous comparons maintenant les deux règles dynamiques SoS et SoM utilisées pour les réseaux à clusters. Le rôle d'une règle dynamique est de calculer un score pour chaque neurone du réseau. La règle SoM (voir équation 2.13) considère que la contribution des neurones au score d'un neurone situé dans un autre cluster ne peut dépasser un. Cette règle est particulièrement adaptée à des dégradations de type intrusion. De plus, cette règle est moins coûteuse en termes de ressources matérielles. En effet, elle permet d'utiliser un arbre de portes logiques OU à

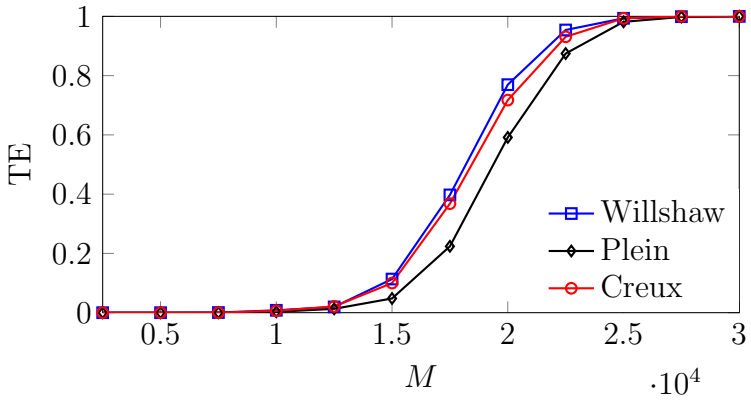


FIGURE 2.5 – Effacement : taux d'erreur (TE) en fonction du nombre M de vecteurs pour un réseau de Willshaw et deux réseaux à clusters, l'un stockant des vecteurs pleins et l'autre stockant des vecteurs creux.

la place d'un arbre d'additionneurs nécessaire pour la règle SoS [20] [22]. Nous comparons ces deux règles sur des vecteurs pleins subissant des dégradations de type effacement et de type intrusion. Les courbes 2.7 et 2.8 rendent compte des résultats obtenus.

On observe sur ces deux courbes que l'utilisation de la règle SoM améliore significativement les performances des réseaux à clusters. Comme attendu, cette amélioration est d'autant plus importante lorsque la dégradation est de type intrusion. Lorsque la dégradation est de type effacement la règle SoM n'apporte pas d'amélioration lors de la première itération de l'algorithme de décodage. En effet, aucun cluster ne possède deux neurones actifs. Cependant, suite à cette première itération, des neurones n'appartenant pas au vecteur recherché peuvent s'activer, causant ainsi des intrusions. La règle SoM est donc également utile lorsque les dégradations sont de type effacement.

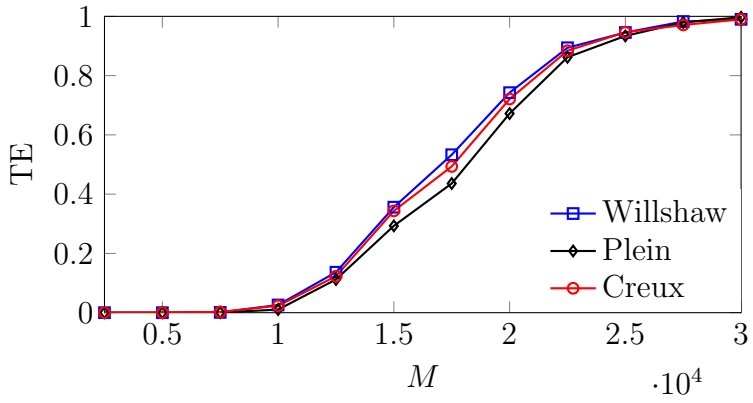


FIGURE 2.6 – Intrusion : taux d'erreur (TE) en fonction du nombre M de vecteurs pour un réseau de Willshaw et deux réseaux à clusters, l'un stockant des vecteurs pleins et l'autre stockant des vecteurs creux.

2.4.4 Règles d'activation

Afin de terminer cette étude, nous évaluons maintenant plusieurs règles d'activation utilisées pour les réseaux à clusters. Ces règles sont employées pour déterminer l'état des neurones du réseau, à partir du score calculé à l'aide d'une règle dynamique. Nous comparons trois règles différentes :

1. La règle k-GWtA que nous avons utilisée dans les précédentes expériences. Lorsque elle est mise en œuvre sur une architecture matérielle, cette règle implique d'utiliser un opérateur de tri ayant $c \times \ell = d$ entrées.

2. La règle LWtA (voir équation 2.14) la règle initialement proposée pour les réseaux à clusters. Cette règle implique d'utiliser c opérateurs MAX ayant ℓ entrées, elle est la moins couteuse en termes de ressources matérielles.

3. La règle GLkO (voir paragraphe 2.2.2), qui consiste à désactiver les neurones actifs ayant le plus petit score. Cette règle implique d'utiliser un opérateur MAX ayant $c \times \ell = d$ entrées. Si le nombre de clusters c est faible, cette règle est légèrement plus coûteuse que la règle LWtA. L'écart

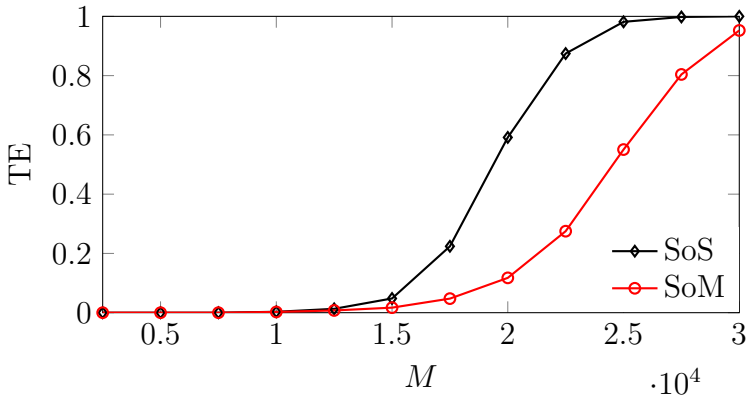


FIGURE 2.7 – Effacement : taux d'erreur (TE) en fonction du nombre M de vecteurs, les règles dynamiques SoM et SoS sont simulées.

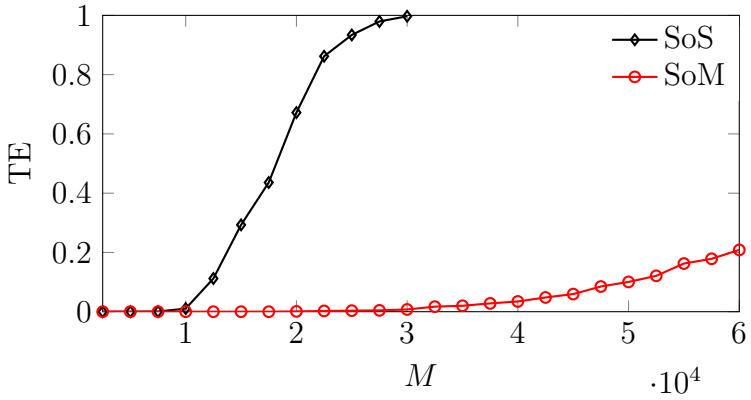


FIGURE 2.8 – Intrusion : taux d'erreur (TE) en fonction du nombre M de vecteurs, les règles dynamiques SoM et SoS sont simulées.

augmente avec c.

Lorsque la règle GLkO est utilisée pour des dégradations de type effacement, il est nécessaire d'utiliser lors de la première itération de l'algorithme de décodage une règle permettant d'activer les neurones manquants. Nous utilisons pour cela la règle k-GWtA. Pour les règles k-GWtA et LWtA nous utilisons comme règle dynamique la règle SoM. Pour la règle GLkO nous réalisons deux simulations, une avec la règle SoS et une autre avec la règle SoM. Les figures 2.9 et 2.10 rendent compte des performances pour les quatre configurations simulées.

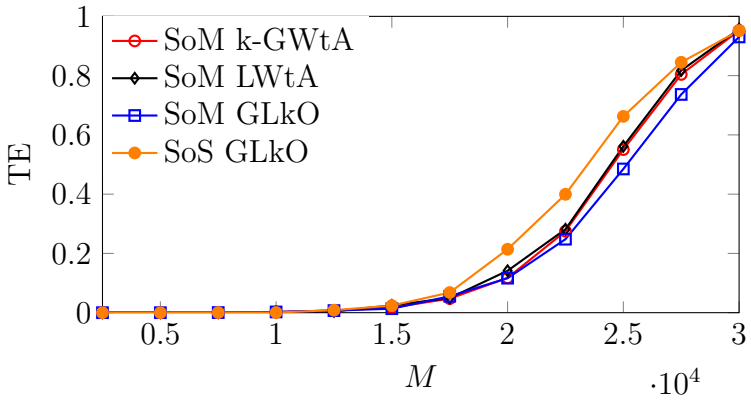


FIGURE 2.9 – Effacement : taux d'erreur (TE) en fonction du nombre M de vecteurs, les règles d'activation GWtA LWtA et GLkO sont simulées.

Lorsque la dégradation est de type effacement les règles k-GWtA et LWtA offrent des performances similaires. La règles GLkO apporte une légère amélioration lorsqu'elle est utilisée avec la règle SoM et une légère détérioration lorsqu'elle est utilisée avec la règle SoS. Pour une dégradation de type intrusion, les trois règles offrent des performances similaires, il faut cependant que la règle GLkO soit couplée avec la règle SoM. Comme pour des dégradations de type effacement la règle GLkO utilisée avec la règle SoS entraîne une dégradation du taux d'erreur. Cependant, cette dégradation peut être considéré comme légère en comparaison de l'utilisation de la règle SoS avec la règle k-GWtA (voir figure 2.8).

2.4.5 Discussion

Cette étude a permis de mettre plusieurs éléments en lumière. Tout d'abord, les réseaux à clusters offrent, par la structure même des vecteurs qu'ils manipulent, une plus grande capacité de stockage qu'un réseau de Willshaw. De plus, cette structure permet de concevoir des règles dynamiques et d'activation plus

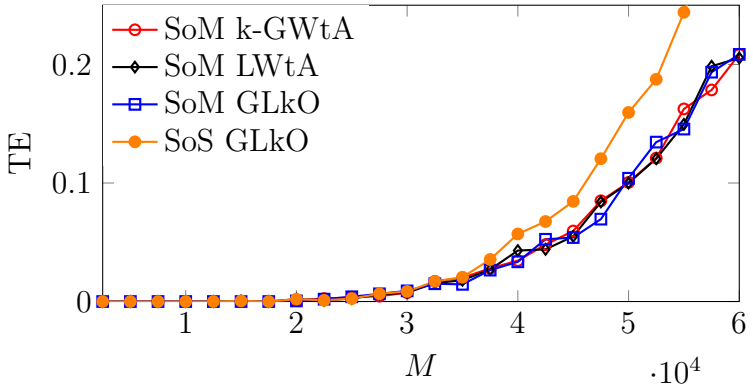


FIGURE 2.10 – Intrusion : taux d'erreur (TE) en fonction du nombre M de vecteurs, les règles d'activation GWtA LWtA et GLkO sont simulées.

robustes. Il est également intéressant d'observer que les règles les moins coûteuses en termes de ressources matérielles (SoM plus LWtA) offrent de très bonnes performances. Seule l'utilisation de la règle GLkO permet une amélioration.

2.5 Conclusion

Les modèles connexionnistes de mémoires associatives sont des modèles distribués permettant de stocker des vecteurs et de les restituer lorsqu'une version dégradée leur est présentée. Contrairement aux mémoires indexées, l'accès aux données ne se fait pas à l'aide d'une adresse mais directement par les données elles-mêmes. Dans ce chapitre, différents modèles de mémoires associatives ont été présentés. Parmi ces modèles, nous nous sommes focalisés sur les réseaux à clusters. Ce modèle, inspiré du modèle de Willshaw, offre une meilleure capacité de stockage tout en étant robuste au bruit. L'utilisation de connexions binaires ainsi que la simplicité de l'algorithme de décodage permet des mises en œuvre particulièrement efficaces sur des architectures matérielles dédiées. Cependant, ce type de réseau a été très peu utilisé pour des applications concrètes. La majo-

rité des travaux ont consisté à étudier ce réseau dans des conditions théoriques idéales d'utilisation. Dans la suite de cette thèse, nous nous intéresserons à son utilisation dans le cadre d'applications concrètes telles que la recherche du plus proche voisin.

Chapitre 3

Recherche du plus proche voisin

L'un des objectifs de cette thèse est d'utiliser les réseaux à clusters pour des applications de vision par ordinateur. Jusqu'alors, les réseaux à clusters ont surtout été utilisés sur des données artificielles sans considérer d'application réelle, ces données artificielles étant générées de telle sorte que les contraintes imposées par le réseau soient satisfaites.

Parmi les différentes tâches que l'on retrouve dans les algorithmes de vision par ordinateur, la recherche du plus proche voisin est sans doute l'une des plus récurrentes. Cette tâche consiste à rechercher dans un ensemble de vecteurs $\mathcal{V} = \{\mathbf{v}^1, ..., \mathbf{v}^M\}$, le vecteur $\mathbf{v}^z$ le plus proche de $\mathbf{q}$, un vecteur requête. La notion de proximité entre les vecteurs se faisant à l'aide d'une métrique donnée telle que la métrique euclidienne ou de Hamming. On retrouve ce type de problème dans des algorithmes de recherche d'images par le contenu [25], de classification non-paramétrique [26], de suivi d'objets [27] ou de regroupement de caractéristiques en mots visuels [28]. Une recherche linéaire consiste à calculer pour chaque vecteur $\mathbf{v}^z \in \mathcal{V}$ la distance $d(\mathbf{q}, \mathbf{v}^z)$, puis à sélectionner le vecteur ayant la plus petite distance. En vision par ordinateur, le nombre important de vecteurs ainsi que

leur taille, en fait l'une des tâches les plus gourmandes en ressources (en mémoire et en calcul).

Les modèles connexionnistes de mémoires associatives tels que les réseaux à clusters sont des systèmes qui mettent en œuvre une recherche du plus proche voisin de manière distribuée. Plutôt que de stocker les vecteurs de manière séquentielle, comme dans une mémoire indexée, un réseau associatif mémorise les relations entre les différentes composantes des vecteurs. Ces relations sont stockées dans une matrice de poids W. Lors de la phase de recherche, le vecteur requête est alors vu comme une version dégradée d'un des vecteurs stockés dans le réseau, $\mathbf{q} = \tilde{\mathbf{v}}^z$. Cependant, pour être stockés et retrouvés, les vecteurs doivent satisfaire certaines contraintes. Pour les réseaux à clusters, les vecteurs doivent être binaires et avoir une structure particulière. De plus, la métrique interne des modèles connexionnistes de mémoires associatives convient rarement aux applications de vision par ordinateur.

Dans ce chapitre, nous étudions la possibilité de stocker des vecteurs denses dans un réseau à clusters dans le but de réaliser une recherche du Plus Proche Voisin (PPV) de manière distribuée. Pour cela, nous adaptons la métrique interne des réseaux à clusters pour permettre une recherche suivant une métrique de Hamming ou euclidienne.

3.1 Vecteurs denses

Les vecteurs denses sont des vecteurs dont le nombre de composantes à zéro est faible. En vision par ordinateur, ces vecteurs peuvent être, par exemple, des descripteurs visuels, la représentation synthétique d'une image... Nous considérons ici deux types de vecteurs denses : les vecteurs binaires ($\mathbf{v} \in \{0,1\}^D$) et les vec-

teurs réels ($\mathbf{v} \in \mathbb{R}^D$). Pour des vecteurs binaires, la distance entre deux vecteurs est estimée à l'aide de la distance de Hamming :

$$PPV(\mathbf{q}) = \underset{z=1}{\overset{M}{\operatorname{argmin}}}\, d_H(\mathbf{q}, \mathbf{v}^z) = \underset{z=1}{\overset{M}{\operatorname{argmin}}} \sum_{j=i}^{D} (q_i \neq v_i^z), \tag{3.1}$$

tandis que pour des vecteurs réels, la distance euclidienne est utilisée :

$$PPV(\mathbf{q}) = \underset{z=1}{\overset{M}{\operatorname{argmin}}}\, d_E(\mathbf{q}, \mathbf{v}^z) = \underset{z=1}{\overset{M}{\operatorname{argmin}}} \sum_{i=0}^{D} (q_i - v_i^z)^2. \tag{3.2}$$

Afin d'effectuer une recherche du plus proche voisin sur des vecteurs denses à l'aide d'un réseau à clusters, nous proposons différentes modifications. D'une part, ces modifications concernent la phase de stockage des vecteurs dans le réseau. En effet, les vecteurs considérés ne satisfont pas les contraintes imposées par le réseau. D'autre part, nous modifions le processus de décodage pour que celui-ci permette de retrouver les vecteurs stockés.

3.2 Stockage

La méthode que nous présentons ici est générique et convient aux deux types de vecteurs considérés. Tout d'abord, nous nous intéressons au stockage des vecteurs denses dans un réseau à clusters. Formellement, nous considérons un ensemble de M vecteurs $\mathcal{V} = \{\mathbf{v}^1, ..., \mathbf{v}^M\}$ denses de dimension D. Ces vecteurs ne peuvent pas être directement stockés dans un réseau à clusters. Afin de permettre leur stockage, nous proposons de les encoder séparément afin de former l'ensemble $\mathcal{X} = \{\mathbf{x}^1, ..., \mathbf{x}^M\}$ tel que pour tout $z \in [1, ..., M]$, chaque vecteur $\mathbf{x}^z = encode(\mathbf{v}^z)$:

1. soit un vecteur binaire de dimension $c\ell$,
2. soit divisible en c sous-vecteurs de taille ℓ ayant chacun une seule composante à un.

L'ensemble des vecteurs $\mathcal{X}$ peut ensuite être stocké dans un réseau à clusters constitué de c clusters de ℓ neurones. Pour encoder les vecteurs $\mathbf{v}^z \in \mathcal{V}$, nous nous inspirons d'une méthode de quantification vectorielle très populaire pour la recherche du plus proche voisin appelée *Produit de Quantification* [29].

3.2.1 Produit de quantification

Le produit de quantification est une méthode de quantification vectorielle généralement appliquée à des vecteurs réels. Elle permet à la fois d'accélérer une recherche linéaire et de réduire l'espace mémoire nécessaire pour stocker $\mathcal{V}$. La quantification vectorielle est une technique de compression avec perte qui permet de représenter un grand ensemble de vecteurs $\mathcal{V} = \{\mathbf{v}^1, ..., \mathbf{v}^n\}$ à l'aide d'un ensemble $\mathcal{U} = \{\mathbf{u}^1, ..., \mathbf{u}^k\}$ de vecteurs de référence. L'ensemble $\mathcal{U}$ ayant une taille plus petite que $\mathcal{V}$ ($k \ll n$). Formellement, un quantifieur est une fonction $q()$ telle que :

$$q(\mathbf{v}^z) = \underset{\mathbf{u}^i \in \mathcal{U}}{\operatorname{argmin}}\, d_E(\mathbf{v}^z, \mathbf{u}^i). \tag{3.3}$$

Les vecteurs de référence $\mathbf{u}^i$ sont appelés mots de codes et l'ensemble $\mathcal{U}$ dictionnaire. Pour des vecteurs réels, le dictionnaire est le plus souvent trouvé à l'aide de l'algorithme des *k-means* qui minimise l'erreur résiduelle E_r :

$$E_r = \sum_{\mathbf{v}^i \in \mathcal{V}} d_E\left(\mathbf{v}^z, q(\mathbf{v}^z)\right). \tag{3.4}$$

Ce problème étant NP-complet, l'algorithme des *k-means* ne conduit pas à une solution optimale mais le minimum local trouvé est jugé satisfaisant dans bon nombre d'applications.

L'idée du produit de quantification est de décomposer l'espace d'origine de dimension D en c sous-espaces de dimension $\frac{D}{c}$ (en considérant que D est divisible par c). Chacun de ces sous-espaces est ensuite quantifié séparément à l'aide

d'un sous-dictionnaire $\mathcal{U}_i$. Pratiquement, chaque vecteur $\mathbf{v}^z \in \mathcal{V}$ est vu comme la concaténation de c sous-vecteurs $[\mathbf{v}_1^z, ..., \mathbf{v}_c^z]$ et est représenté par $q(\mathbf{v}^z) = [q_1(\mathbf{v}_1^z), ..., q_c(\mathbf{v}_c^z)]$ avec $q_i(\mathbf{v}_i^z) \in \mathcal{U}_i = \{\mathbf{u}_i^1, ..., \mathbf{u}_i^k\}$. Lors d'une recherche du plus proche voisin, la distance euclidienne $d_E(\mathbf{q}, \mathbf{v}^z)$ entre un vecteur requête et un vecteur de $\mathcal{V}$ est approximée par la distance $d_E(\mathbf{q}, q(\mathbf{v}^z))$ entre $\mathbf{q}$ et la version quantifiée de $\mathbf{v}^z$. Cette distance se calcule de la manière suivante :

$$d_E(\mathbf{q}, \mathbf{v}^z) \simeq d_E(\mathbf{q}, q(\mathbf{v}^z)) = \sum_{i=1}^{c} d_E(\mathbf{q}_i, q_i(\mathbf{v}_i^z)). \tag{3.5}$$

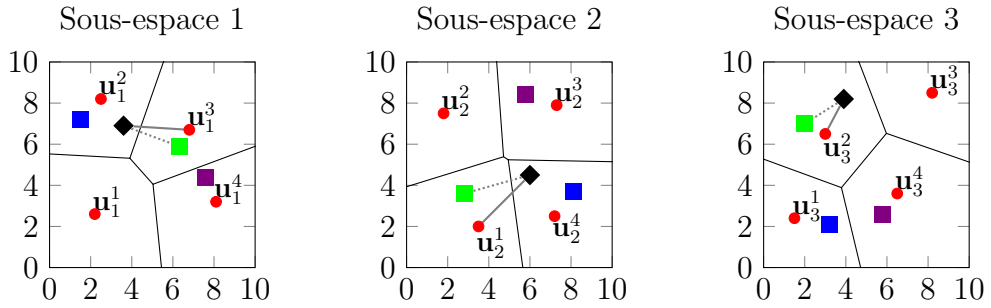
La figure 3.1 représente un exemple d'utilisation du produit de quantification. L'ensemble $\mathcal{V}$ est constitué de trois vecteurs $\mathbf{v}^z \in \mathbb{R}^6$. L'espace d'origine est divisé en $c = 3$ sous-espaces de dimension deux, chacun quantifié à l'aide d'un sous-dictionnaire $\mathcal{U}_i$ comprenant $k = 4$ mots de codes. Le plus proche voisin de $\mathbf{q}$ est le vecteur $\mathbf{v}^2$. La distance euclidienne entre ces deux vecteurs $d(\mathbf{q}, \mathbf{v}^2)$ est approximée par $d_E(\mathbf{q}, q(\mathbf{v}^2)) = d_E(\mathbf{q}_1, \mathbf{u}_1^3) + d_E(\mathbf{q}_2, \mathbf{u}_2^1) + d_E(\mathbf{q}_3, \mathbf{u}_3^2)$.

3.2.2 Encodage des vecteurs

Nous réutilisons la méthode du produit de quantification afin de former les vecteurs $\mathbf{x}^z = \mathit{encode}(\mathbf{v}^z)$ tels que $\mathbf{x}^z \in \{0, 1\}^{c\ell}$. Ces vecteurs pourront ensuite être stockés dans un réseau constitué de c clusters de ℓ neurones. Comme pour le produit de quantification, l'espace d'origine de dimension D est divisé en c sous-espaces de dimension $\frac{D}{c}$, chacun quantifié à l'aide d'un sous-dictionnaire $\mathcal{U}_i = \{\mathbf{u}_i^1, ..., \mathbf{u}_i^\ell\}$ de taille ℓ. Chaque sous-espace est ensuite associé à un cluster du réseau et chaque mot de code $\mathbf{u}_j^i \in \mathcal{U}_i$, utilisé pour représenter le sous-espace i, est associé au neurone (i, j). Lors de la phase d'encodage, les vecteurs $\mathbf{v}^z$ sont divisés en c sous-vecteurs $\mathbf{v}_i^z$. Pour chaque cluster i, le neurone alloué au mot de code ayant la plus petite distance avec le sous-vecteur $\mathbf{v}_i^z$ est sélectionné pour

$$\mathcal{V} = \left\{ \begin{array}{c} \mathbf{v}^1 \\ \mathbf{v}^2 \\ \mathbf{v}^3 \end{array} \right\} = \left\{ \begin{array}{c} [\mathbf{v}_1^1, \mathbf{v}_2^1, \mathbf{v}_3^1] \\ [\mathbf{v}_1^2, \mathbf{v}_2^2, \mathbf{v}_3^2] \\ [\mathbf{v}_1^3, \mathbf{v}_2^3, \mathbf{v}_3^3] \end{array} \right\} \overset{PQ}{\Longrightarrow} \left\{ \begin{array}{c} [\mathbf{u}_1^2, \mathbf{u}_2^4, \mathbf{u}_3^1] \\ [\mathbf{u}_1^3, \mathbf{u}_2^1, \mathbf{u}_3^2] \\ [\mathbf{u}_1^4, \mathbf{v}_2^3, \mathbf{v}_3^4] \end{array} \right\}$$

(a) Représentation matricielle.



(b) Représentation graphique du produit de quantification ; pour chaque sous-espace i les mots de codes $\mathbf{u}_i^j$ sont représentés par des points rouges, les sous-vecteurs $\mathbf{v}_i^z$ par des carrés et les sous-vecteurs requête $\mathbf{q}_i$ par des diamants noirs ; le vecteur le plus proche du vecteur requête est le vecteur $\mathbf{v}^2$; les distances $d_E(\mathbf{q}_i, \mathbf{v}_i^2)$ sont représentées par des lignes pointillées tandis que les distances $d_E(\mathbf{q}_i, q_i(\mathbf{v}_i^2))$ sont représentées par des lignes pleines.

FIGURE 3.1 – Produit de quantification.

stocker $\mathbf{v}^z$. La fonction suivante résume la procédure d'encodage des vecteurs $\mathbf{v}^z$:

$$\begin{aligned} \mathbf{v}^z = \quad & \underbrace{v_1^z, ..., v_{\frac{D}{c}}^z}_{\mathbf{v}_1^z}, \quad ..., \quad \underbrace{v_{(c-1)\frac{D}{c}+1}^z, ..., v_D^z}_{\mathbf{v}_c^z} \\ \rightarrow \quad \mathbf{x}^z = \quad & \mathbf{x}_1^z = \Phi(\mathbf{v}_1^z), \quad ..., \quad \mathbf{x}_c^z = \Phi(\mathbf{v}_c^z) \end{aligned} , \tag{3.6}$$

avec :

$$\Phi : x_{(i,j)} = \left\{ \begin{array}{l} 1 \text{ si } j = \underset{\mathbf{u}_i^k \in \mathcal{U}_i}{\operatorname{argmin}} d_X(\mathbf{v}_i^z, \mathbf{u}_i^k) \\ \\ 0 \text{ sinon} \end{array} \right. , \tag{3.7}$$

où $d_X()$ est la distance euclidienne pour des vecteurs réels et la distance de Hamming pour des vecteurs binaires. Dans l'exemple représenté par la figure 3.1,

l'encodage de $\mathcal{V}$ conduit à l'ensemble $\mathcal{X}$ suivant :

$$\mathcal{X} = \left\{ \begin{array}{c} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \mathbf{x}^3 \end{array} \right\} = \left\{ \begin{array}{c} [0010 \mid 1000 \mid 0001] \\ [0100 \mid 0001 \mid 0010] \\ [1000 \mid 0100 \mid 1000] \end{array} \right\}. \tag{3.8}$$

La figure 3.2 représente le stockage des vecteurs de la figure 3.1 dans un réseau à clusters.

Création des sous-dictionnaires $\mathcal{U}^i$

Pour des **vecteurs binaires**, chaque sous-espace associé à un cluster est un espace discret et fini, c'est-à-dire que tous les points peuvent êtres isolés et dénombrés. Ces propriétés permettent de construire pour chaque sous-espace i de dimension $\frac{D}{c}$, un sous-dictionnaire exhaustif $\mathcal{U}_i$ de taille $\ell = 2^{\frac{D}{c}}$ contenant tous les points de ce sous-espace. Ainsi, l'encodage des vecteurs $\mathbf{v}^z$ ne génère aucune perte d'information.

Pour des **vecteurs réels**, chaque sous-espace est un espace continu. Les sous-dictionnaires $\mathcal{U}^i$ ne peuvent donc offrir qu'une représentation partielle du sous-espace auquel ils sont associés. Afin de réduire l'erreur de quantification (voir équation 3.4) nous créons ces sous-dictionnaires à l'aide de l'algorithme des *k-means* comme proposé dans [29]. Leur taille est donc égale à $\ell = k$, avec k un hyperparamètre. Les sous-espaces n'étant représentés que de manière partielle, l'encodage des vecteurs $\mathbf{v}^z$ produit une perte d'information. L'importance de cette perte d'information diminue lorsque c et k augmentent.

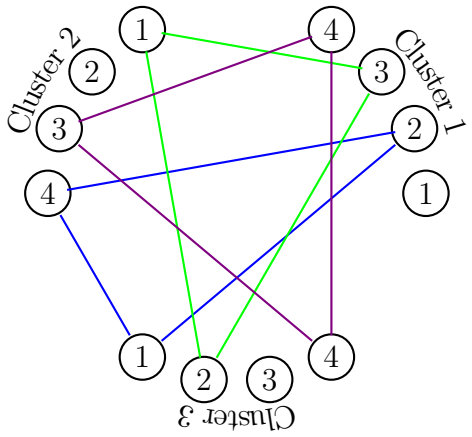


FIGURE 3.2 – Représentation graphique du stockage des vecteurs de la figure 3.1 à l'aide d'un réseau constitué de trois clusters de quatre neurones.

3.3 Recherche

Lors de la phase de recherche, le vecteur $\mathbf{q}$ est considéré comme une version dégradée d'un des vecteurs $\mathbf{v}^z \in \mathcal{V}$ tel que $\mathbf{v}^z = PPV(\mathbf{q})$. Le réseau est alors utilisé pour retrouver le vecteur $\mathbf{x}^z = encode(\mathbf{v}^z)$. La méthode d'encodage utilisée pour stocker $\mathcal{V}$ conduit à représenter dans chaque cluster un sous-espace de dimension $\frac{D}{c}$ de l'espace d'origine. Chaque neurone d'un cluster est associé à un vecteur représentatif de ce sous-espace. Le comportement binaire des neurones dans un réseau à clusters amène à considérer chacun de ces sous-vecteurs comme équidistants et ne peut permettre une recherche dans l'espace d'origine.

Dans l'exemple représenté par les figures 3.1 et 3.2, l'encodage du vecteur requête $\mathbf{q}$ à l'aide de la méthode d'encodage résumée par l'équation 3.6 conduit au vecteur binaire $encode(\mathbf{q}) = [0010 \mid 1000 \mid 0010]$. L'initialisation du réseau à l'aide de ce vecteur ($\mathbf{e} = encode(\mathbf{q})$) mène à l'activation des neurones $(1, 2)$, $(2, 1)$ et $(3, 3)$. Après l'utilisation de la règle dynamique SoM (voir le chapitre 2.2.2), le vecteur de score du réseau est égal à $\mathbf{s} = [0, 1, 2, 0 \mid 2, 0, 0, 1 \mid 0, 0, 2, 1]$. Après

l'application de la règle d'activation L-WtA (voire le chapitre 2.2.2), l'état du réseau est égal à $\mathbf{e} = [0010 \mid 1000 \mid 0001]$, ce qui correspond au vecteur $\mathbf{x}^1 = \mathit{encode}(\mathbf{v}^1)$. Cependant, comme annoncé précédemment, le plus proche voisin de $\mathbf{q}$ est le vecteur $\mathbf{v}^2$.

Afin d'effectuer la recherche dans l'espace d'origine, nous proposons d'ajouter à chaque neurone (i, j) une valeur rendant compte des distances entre le sous-vecteur requête $\mathbf{q}_i$ et le mot de code $\mathbf{u}_i^j$. Nous appelons cette valeur **potentiel d'action** d'un neurone. L'utilisation de potentiels d'action implique de modifier le processus de décodage.

3.3.1 Potentiel d'action

Plutôt que d'utiliser directement les distances $d_X(\mathbf{q}_i, \mathbf{u}_i^j)$, nous utilisons une mesure de similarité calculée à partir de ces distances. Si $\mathbf{q}$ est un vecteur requête de dimension D, les potentiels d'action de chaque neurone (i, j) sont calculés de la manière suivante :

$$p_{(i,j)} = T_i - d_X(\mathbf{q}_i, \mathbf{u}_i^j). \tag{3.9}$$

avec T_i une constante associée à un cluster. Cette constante peut être la même pour tous les clusters. Ainsi, dans chaque cluster i, le potentiel d'action le plus élevé est attribué au neurone associé au mot de code le plus proche du sous-vecteur $\mathbf{q}_i$. Le neurone associé au mot de code le plus éloigné se voit attribuer le potentiel d'action le plus faible. Ces potentiels d'actions sont regroupés dans un vecteur $\mathbf{p}$ et ajoutés au vecteur d'état $\mathbf{e} \in \{0, 1\}^{c\ell}$. L'activité d'un neurone (i, j) est alors égale à son état $e_{(i,j)}$, multiplié par son potentiel d'action $p_{(i,j)}$. La règle

dynamique SoS est alors modifiée de la manière suivante :

$$\forall i,j,s_{(i,j)} = \sum_{i'=1}^{c}\sum_{j'=1}^{\ell} e_{(i',j')}p_{(i',j')}w(i,j)(i',j'), \tag{3.10}$$

en d'autres termes, le score d'un neurone est égal à la somme des potentiels d'action des neurones actifs auxquels il est connecté.

Dans l'exemple représenté par les figures 3.1 et 3.2, si nous considérons tous les neurones comme actifs ($\mathbf{e} = \{1\}^{c\ell}$), l'application de la règle SoS conduit aux scores suivants :

$$\begin{aligned} s(\mathbf{v}^1) &= s_{(1,2)} = s_{(2,4)} = s_{(3,1)} = p_{(1,2)} + p_{(2,4)} + p_{(3,1)} \\ &= C - d_E(\mathbf{q}_1, \mathbf{u}_1^2) - d_E(\mathbf{q}_2, \mathbf{u}_2^4) - d_E(\mathbf{q}_3, \mathbf{u}_3^1) \\ &= C - d_E(\mathbf{q}, q(\mathbf{v}^1)) \qquad , \\ s(\mathbf{v}^2) &= C - d_E(\mathbf{q}, q(\mathbf{v}^2)) \\ s(\mathbf{v}^3) &= C - d_E(\mathbf{q}, q(\mathbf{v}^3)) \end{aligned} \tag{3.11}$$

avec

$$C = \sum_{i=1}^{c} T_i \tag{3.12}$$

L'application d'une simple règle d'activation telle que la règle LWtA permet alors de retrouver le plus proche voisin de $\mathbf{q}$. Dans cet exemple, la recherche est aisée car aucun vecteur stocké ne partage de connexion avec les autres vecteurs. Lorsque le nombre de vecteurs augmente, le partage de connexions entraîne des difficultés lors de la recherche du plus proche voisin et implique d'utiliser une règle dynamique et une règle d'activation plus robustes que les règles SoS et LWtA.

3.3.2 Processus de décodage

Le processus de décodage se décompose en une phase d'initialisation et en une phase itérative. Lors de la phase d'initialisation, les potentiels d'action sont calculés à partir du vecteur requête $\mathbf{q}$ et le vecteur d'état $\mathbf{e}$ est initialisé. Pour cela,

toute les valeurs du vecteur sont mises à un. Ce mode d'initialisation conduit à l'activation d'un grand nombre de neurones intrus, le processus de décodage aura alors en charge de les désactiver. Ce processus consiste à appliquer alternativement une règle dynamique et une règle d'activation. Cette phase s'arrête lorsqu'il ne reste qu'un seul neurone actif par cluster.

Règle dynamique

Dans le précédent chapitre, nous avons présenté les deux règles dynamiques (SoS et SoM) couramment utilisées pour les réseaux à clusters. Pour la règle SoS, tous les neurones actifs connectés au neurone (i, j) contribuent au score de ce dernier, tandis que la règle SoM tire partie de la structure du réseau en saturant les contributions provenant d'un même cluster à un.

Si ces deux règles offrent des performances équivalentes pour certaines applications [17], le choix de la règle dynamique est crucial dans le contexte de la recherche du plus proche voisin. En effet, de nombreuses intrusions sont présentes après la phase d'initialisation. Afin de minimiser le bruit provoqué par ces intrusions, nous utilisons ici la règle SoM que nous modifions afin de tenir compte des potentiels d'action. Pour chaque cluster, seul le neurone actif, connecté au neurone (i, j) et ayant le potentiel d'action le plus élevé, contribue au score de ce dernier. Cette règle peut s'écrire de la manière suivante :

$$\forall i, j, s_{(i,j)} = \sum_{i'=1}^{c} \max_{j'=1}^{\ell} \left(e_{(i',j')} p_{(i',j')} w_{(i,j)(i',j')} \right) \tag{3.13}$$

Règle d'activation

Une règle d'activation a pour objectif de déterminer l'état $e_{(i,j)}$ de chaque neurone à partir de son score $s_{(i,j)}$. Dans le cas qui nous concerne, la règle d'acti-

vation est utilisée pour éliminer les $c\ell - c$ neurones intrus. Il existe différentes règles d'activation dans la littérature des réseaux à clusters (voir chapitre précédent). Parmi elles, la règle GLkO est celle qui offre la plus forte probabilité de retrouver le vecteur d'origine lorsque les erreurs sont uniquement de type intrusion. Considérons $\mathcal{E} = \{(i_1, j_1), ..., (i_m, j_m)\}$ l'ensemble des m neurones actifs et $\mathcal{S} = \{s_{(i_1,j_1)}, ..., s_{(i_m,j_m)}\}$, leur score. La règle GLkO met à jour l'état des neurones de la manière suivante :

$$\forall (i,j) \in \mathcal{E}, e_{(i,j)} = \begin{cases} 1 \text{ si } s_{(i,j)} > \theta \\ 0 \text{ sinon} \end{cases}, \tag{3.14}$$

avec $\theta = \min(\mathcal{S})$. En d'autres termes, la règle GLkO met à zéro l'état des neurones actifs ayant le plus petit score. L'inconvénient de cette règle d'activation est que le nombre d'itérations dépend du nombre d'intrusions dans le réseau. Dans notre cas, le nombre d'intrusions est élevé, menant à un long temps de décodage. Afin de réduire ce nombre d'itérations, nous proposons deux variantes de la règle GLkO.

GLkO-saturée : La première de ces variantes revient à simplifier l'objectif du décodage. Pour cela, nous ajoutons une contrainte à la règle GLkO. Cette contrainte est la suivante : la distance entre le vecteur requête et son plus proche voisin doit être inférieure à une certaine distance δ. Le seuil θ utilisé dans l'équation 3.14 peut alors être calculé de la manière suivante :

$$\theta = \max\left(C - \delta, \min(\mathcal{S})\right) \tag{3.15}$$

avec C calculée à l'aide de l'équation 3.12. Nous appelons cette règle GLkO-saturée (GLkO-S). Son utilisation permet d'éviter que des valeurs trop petites de $\min(\mathcal{S})$ ne conduisent à la désactivation que d'un très faible nombre de neurones.

k-GLkO : La seconde de ces variantes s'inspire de la règle k-GWtA proposée dans [17]. Plutôt que choisir comme valeur pour le seuil θ le plus petit score de $\mathcal{S}$, on choisit le k^{ieme} plus petit score. Au début de la phase de décodage il est préférable de choisir une valeur de k élevée afin d'éliminer rapidement un grand nombre de neurones intrus. A la fin de la phase de décodage, une valeur faible de k permet une recherche plus fine. Lors de la première itération nous fixons $k = k^{init}$. Puis, à chaque nouvelle itération, nous faisons évoluer k à l'aide de l'équation suivante :

$$k = \max\left(1, \lfloor \alpha |\mathcal{S}| \rfloor\right), \tag{3.16}$$

avec $|\mathcal{S}|$ le cardinal de $\mathcal{S}$ et $\lfloor x \rfloor$ l'arrondi inférieur de x. Le paramètre $\alpha \in [0, 1]$ permet de régler la vitesse du décodage. Si ce paramètre est élevé, un grand nombre de neurones sera désactivé à chaque itération. En conséquence un faible nombre d'itérations sera nécessaire.

3.4 Évaluation

Nous allons maintenant évaluer les performances d'une recherche du plus proche voisin à l'aide d'un réseau à clusters. Cette évaluation sera faite à la fois sur des vecteurs générés aléatoirement ainsi que sur des vecteurs issus de données naturelles. Nous choisissons pour ces derniers des vecteurs SIFT [30] extraits de la base SIFT1M. Cette base introduite dans [29], est couramment utilisée pour évaluer les performances des techniques de recherche du plus proche voisin.

3.4.1 Vecteurs Binaires

Nous nous intéressons tout d'abord au stockage et à la recherche de vecteurs binaires.

Vecteurs aléatoires

Nous utilisons ici des vecteurs binaires de 64 bits distribués de manière uniforme et indépendante. Chaque vecteur est divisé en huit sous-vecteurs de huit bits et est stocké dans un réseau à clusters constitué de $c = 8$ clusters de $\ell = 2^8 = 256$ neurones. Les vecteurs requêtes sont générés en choisissant aléatoirement l'un des vecteurs de la base puis en permutant b bits. Tout d'abord, nous fixons $b = 10$ et faisons évoluer le nombre M de vecteurs stockés dans le réseau. La figure 3.3 rend compte du taux d'erreur ER et du nombre d'itérations NI en fonction de M pour les règles d'activation GLkO et GLkO-S. Ici δ est choisi comme étant égal à $b + 1 = 11$. Sur cette figure, on observe que les deux règles offrent les mêmes performances en termes de qualité de décodage. Le taux d'erreur évolue exactement de la même manière dans les deux cas. Cependant le nombre d'itérations de l'algorithme varie de manière différente. Pour la règle GLkO, le nombre d'itérations est élevé lorsque le nombre de vecteurs M est faible ($\sim$ 300 itérations pour 2500 vecteurs) et décroit lorsque M augmente. La règle GLkO-S offre le comportement inverse. Lorsque le taux d'erreur est égal à zéro le nombre d'itération est faible.

Nous proposons maintenant d'observer le comportement de la règle d'activation GLkO-S en fonction du nombre b de permutations. Pour cela nous fixons M et faisons évoluer b de quatre à dix. Nous conservons la même valeur pour δ que précédemment ($\delta = 11$). La figure 3.4 rend compte de l'évolution du taux d'erreur et du nombre d'itérations en fonction de b pour $M = \{10000, 15000\}$.

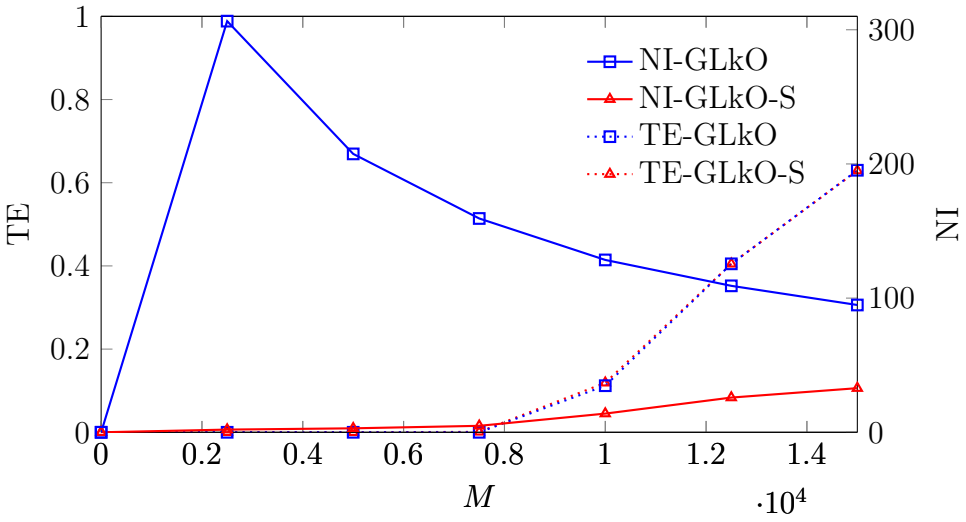


FIGURE 3.3 – Évolution du taux d'erreur (TE) et du nombre d'itérations (NI) pour des vecteurs générés aléatoirement et pour les deux règles d'activations (GLkO et GLkO-S).

Comme attendu, le taux d'erreur est maintenu à zéro jusqu'à un certain nombre de permutations (neuf pour $M = 10000$, sept pour $M = 15000$), puis il évolue de manière exponentielle. En ce qui concerne le nombre d'itérations, ce dernier reste constant (~ 14 pour $M = 10000$ et ~ 34 pour $M = 15000$), pour n'importe quel nombre de permutations.

Vecteurs issus de données naturelles

Les vecteurs SIFT [30] sont des vecteurs réels de dimension 128 qui représentent des caractéristiques visuelles extraites d'images. Ces caractéristiques sont couramment utilisées en vision par ordinateur dans des applications comme la classification d'objets ou la recherche d'images par le contenu. La base SIFT1M [30] regroupe un million de descripteurs SIFT. Afin de transformer les vecteurs SIFT en vecteurs binaires, nous utilisons une technique de compression nommée LSH (Locality Sensitive Hashing) [31], pour créer des vecteurs binaires de 64 bits. Pour

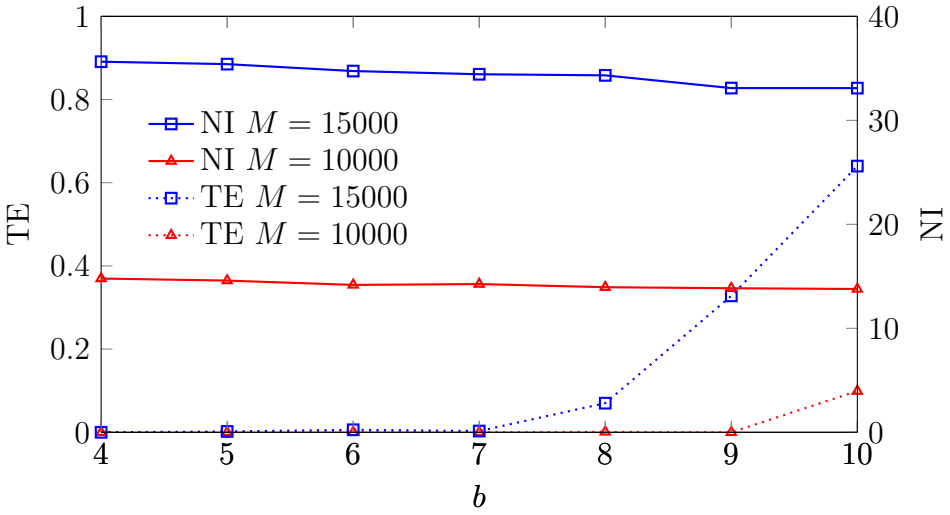


FIGURE 3.4 – Évolution du taux d'erreur et du nombre d'itérations en fonction de b le nombre de permutations. La règle d'activation GLkO-S est utilisée.

stocker ces vecteurs, nous utilisons le même découpage et le même réseau à clusters que précédemment ($c = 8, \ell = 2^8 = 256$). Nous reproduisons les deux mêmes expériences que pour des vecteurs aléatoires. Seule la règle d'activation GLkO-S est utilisée.

La figure 3.5 rend compte de l'évolution du taux d'erreur (TE) et du nombre d'itérations (NI) lorsque le nombre de vecteurs M varie. On observe sur cette figure une augmentation du taux d'erreur comparée à l'expérience précédente. Cette augmentation vient du fait que les réseaux à clusters offrent des performances maximales lorsque les vecteurs sont uniformément et indépendamment distribués. Les vecteurs obtenus à partir de données naturelles sont le plus souvent fortement corrélés, entraînant une dégradation du taux d'erreur.

3.4.2 Vecteurs réels

Nous allons maintenant étudier le stockage et la recherche de vecteurs réels. Dans cette étude, nous nous intéresserons uniquement à des vecteurs issus de

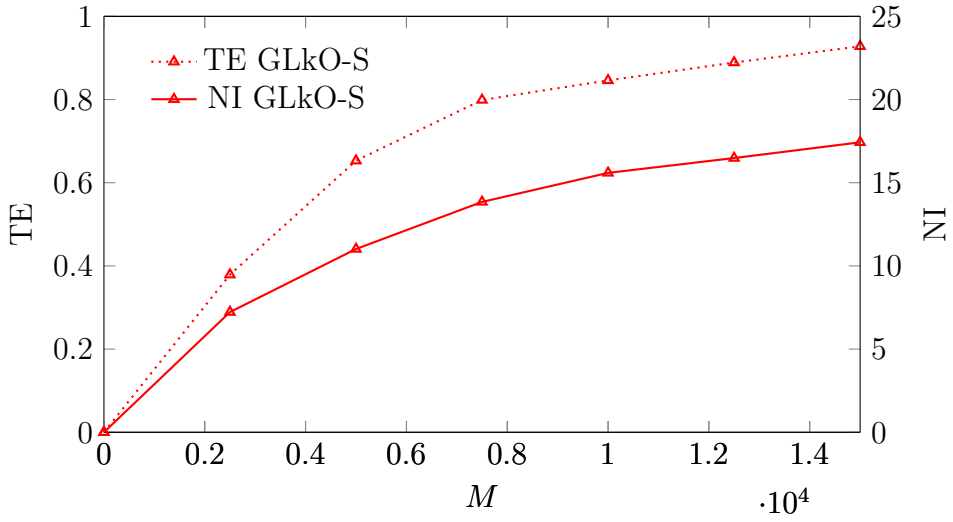


FIGURE 3.5 – Évolution du taux d'erreur (TE) et du nombre d'itérations (NI) pour des vecteurs issus de données naturelles pour la règle d'activation GLkO-S.

données naturelles. Les vecteurs SIFT sont des vecteurs réels de 128 dimensions. L'espace d'origine est divisé en huit sous-espaces de dimension 16. Chacun de ces sous-espaces est représenté à l'aide de 256 vecteurs de références, calculés à l'aide de l'algorithme des k-means. Après encodage, les vecteurs peuvent être stockés dans un réseau à clusters constitué de $c = 8$ clusters de $\ell = 256$ neurones. Les vecteurs requêtes sont eux-même des vecteurs SIFT tirés de la base SIFT1M. Nous considérons que le plus proche voisin est le vecteur minimisant la distance $d_E(\mathbf{q}, q(\mathbf{x}))$ entre le vecteur requête $\mathbf{q}$ et la version quantifiée $q(\mathbf{x})$. Nous utilisons comme règle d'activation la règle k-GLkO.

La figure 3.6 montre l'évolution du taux d'erreur et du nombre d'itérations en fonction de nombre M de vecteurs stockés pour différentes valeurs de α. Le paramètre α permet de régler le nombre de neurones désactivés à chaque itération. Une valeur faible de ce paramètre (proche de zéro) conduit à désactiver un faible nombre de neurones à chaque itération du processus de décodage. Comme attendu, on observe sur cette figure que de faibles valeurs pour α améliorent la

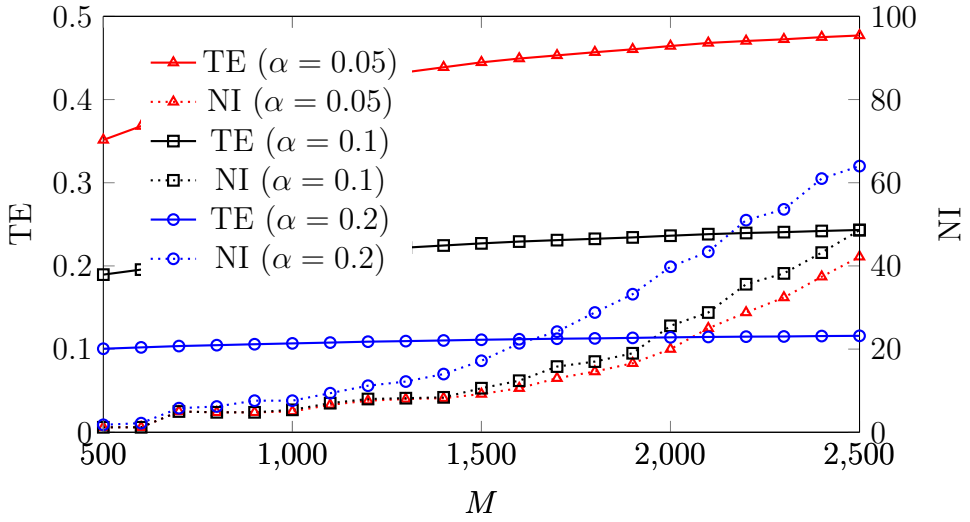


FIGURE 3.6 – Évolution du taux d'erreur (TE) et du nombre d'itérations (NI) pour des vecteurs issus de données naturelles, la règle d'activation k-GLkO est utilisée pour plusieurs valeurs de α.

précision. Cependant, cette amélioration se traduit par une nette augmentation du nombre d'itérations.

3.5 Conclusion

Nous nous sommes intéressés à l'utilisation d'un réseau à clusters pour effectuer une recherche du plus proche voisin. Cette tâche est massivement utilisée dans des applications de vision par ordinateur. Un réseau associatif peut être considéré comme un système effectuant une recherche du plus proche voisin de manière distribuée. Les vecteurs de la base de recherche sont stockés dans le réseau et le vecteur requête est considéré comme une version dégradée de son plus proche voisin. Cependant, les vecteurs pouvant être stockés dans un réseau associatif sont peu courants en vision par ordinateur. Dans ce chapitre, nous avons fait différentes propositions permettant d'utiliser un réseau à clusters pour

stocker et retrouver des vecteurs denses (à valeurs réelles ou binaires).

Pour stocker des vecteurs denses dans un réseau à cluster, nous avons proposé une méthode d'encodage exploitant la technique du produit de quantification. Cette technique couramment utilisée pour effectuer des recherches du plus proche voisin approchée permet de représenter un espace de grande dimension à l'aide d'un ensemble de dictionnaires, chacun utilisé pour représenter un sous-espace de l'espace d'origine. Pour retrouver le plus proche voisin d'un vecteur requête, nous avons introduit les potentiels d'action. Ces potentiels d'action permettent de rendre compte des distances dans l'espace d'origine à l'intérieur du réseau. Enfin, nous avons adapté le processus de décodage des réseaux à clusters afin qu'il prenne en compte ces potentiels d'action. Ces modifications ne sont pas sans conséquence lorsqu'elles sont mises en œuvre sur une architecture matérielle. Dans [32], nous avons étudié l'impact de ces modifications sur une mise en œuvre sur FPGA.

L'approche proposée a été évaluée sur des vecteurs générés aléatoirement ainsi que sur des vecteurs issus de données naturelles. Si l'approche proposée permet de retrouver efficacement des vecteurs qui sont indépendamment et uniformément distribués, dans le cas contraire, les performances se dégradent. Nous abordons dans le chapitre suivant cette problématique en proposant un modèle de réseau de neurones inspiré des réseaux à clusters. Ce nouveau réseau doit permettre d'améliorer le stockage de vecteurs qui ne sont pas indépendamment distribués.

Chapitre 4

Réseaux à Clusters Restreints

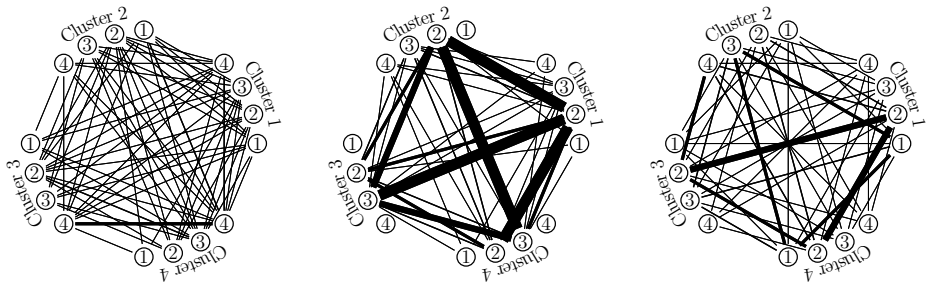
4.1 Contextes

4.1.1 Vecteurs corrélés

Les performances des réseaux à clusters sont maximales lorsque les vecteurs $\mathbf{x}^z$ sont orthogonaux deux à deux. Des vecteurs quasi orthogonaux peuvent être obtenus s'ils sont générés de manière uniforme et de manière indépendante, c'est-à-dire que :

1. toutes les composantes d'un sous-vecteur $\mathbf{x}_i^z$ ont la même probabilité d'être activées, $\forall i, j, \mathbb{P}(I_i = j) = \frac{1}{\ell}$,
2. la probabilité de coactivation de deux composantes est la même pour toutes les composantes des vecteurs $\mathbf{x}^z$, $\forall i, j, i', j', \mathbb{P}(I_i = j, I_{i'} = j') = \mathbb{P}(I_i = j) \times \mathbb{P}(I_{i'} = j')$,

avec I_i, l'index de la composante active dans les sous-vecteurs $\mathbf{x}_i^z$. Si la première de ces conditions n'est pas respectée, certains neurones auront tendance à être surutilisés par rapport à d'autres. Un cas extrême serait que dans un cluster



(a) Cas un (b) Cas deux (c) Cas trois

FIGURE 4.1 – Réseaux à clusters constitués de $c = 4$ clusters de $\ell = 4$ neurones stockant chacun huit vecteurs. L'épaisseur des arêtes est proportionnelle au taux d'utilisation des connexions.

un seul neurone soit utilisé. Ce neurone n'apporte alors plus aucune information sur l'état des autres neurones du réseau. Lorsque la deuxième condition n'est pas respectée, ce sont certaines connexions qui se retrouvent surutilisées. Dans les deux cas, l'effort de stockage n'est pas réparti sur l'ensemble du réseau et le processus de décodage peine à différencier les vecteurs stockés.

La figure 4.1 montre le stockage dans un réseau à clusters de vecteurs générés de trois manières différentes. Dans le cas un (figure 4.1a), les vecteurs sont générés de manière uniforme et indépendante. On observe que chaque neurone possède un nombre équivalent de connexions et que ces connexions sont utilisées un faible nombre de fois. Dans le cas deux, (figure 4.1b) les composantes à un sont générées à partir d'une distribution gaussienne tronquée. En conséquence, les neurones trois et deux dans chacun des clusters se retrouvent plus utilisés que les neurones un et quatre. Les connexions entre ces neurones sont également davantage utilisées. Dans le cas trois, les composantes non nulles sont générées de manière non-indépendante : soit les indices I_1,I_2,I_3 et I_4 sont pairs, soit ils sont impairs. Les neurones sont utilisés un même nombre de fois mais les connexions ont ten-

dance à être plus utilisées que dans le cas un.

La majorité des travaux portant sur les réseaux à clusters [6, 16, 15, 17] se placent dans le cas idéal (cas un). Cependant des vecteurs issus de données naturelles (image, son, texte,...) satisfont rarement les contraintes d'indépendance et d'uniformité. Le plus souvent les vecteurs sont fortement corrélés. Leur stockage dans un réseau à clusters conduit alors à des performances médiocres si elles sont comparées au cas idéal. Afin de stocker des vecteurs corrélés dans des réseaux à clusters, certains travaux [33, 34, 35] se sont intéressés à résoudre le problème de distributions non-uniformes (cas deux). Dans ce chapitre nous proposons une solution permettant de répondre au problème de non-indépendance (cas trois).

4.1.2 Distribution non-uniforme

Différentes solutions ont été proposées pour améliorer les performances des réseaux à clusters dans le contexte de distributions non-uniformes. Nous allons maintenant exposer ces différentes solutions. Dans [35], les auteurs proposent d'utiliser plusieurs réseaux. Durant la phase d'apprentissage, on sélectionne pour chaque vecteur $\mathbf{x}^z$ l'un des réseaux pour son stockage. Lors de la phase de décodage, le vecteur dégradé est présenté à chacun des réseaux qui tentent alors de retrouver le vecteur d'origine. Une fois le décodage terminé, chaque réseau propose donc une solution. Le vecteur d'origine est ensuite déterminé en comparant ces solutions avec le vecteur dégradé. L'utilisation de plusieurs réseaux est la solution la plus triviale et la plus simple à mettre en œuvre pour augmenter la capacité de stockage d'un réseau à clusters. Lorsque les vecteurs ne sont pas corrélés, elle est également la plus efficace. Lorsque les vecteurs ne sont pas uniformément distribués, elle offre une capacité de stockage plus importante que d'autres solutions

souvent plus compliquées à mettre en œuvre.

Dans [33], différentes stratégies sont étudiées. La première stratégie consiste à ajouter des clusters au réseau. Pour être stockés, les vecteurs $\mathbf{x}^z$ sont concaténés avec des vecteurs $\mathbf{y}^z$ uniformément distribués. Le réseau a alors pour objectif de retrouver $[\mathbf{x}^z, \mathbf{y}^z]$. La seconde stratégie consiste à allouer plusieurs neurones à une même composante de $\mathbf{x}$. Lors de l'apprentissage, l'un des neurones est choisi aléatoirement tandis que lors du décodage, l'ensemble des neurones alloués est activé. Cette stratégie permet de diminuer l'utilisation de neurones surchargés (voir figure 4.1b). Cependant, comme le même nombre de neurones est alloué à toutes les composantes de $\mathbf{x}^z$, des neurones sont également ajoutés pour les composantes s'activant rarement, conduisant ainsi à un surcoût inutile. Ces deux premières stratégies offrent des capacités de stockage moins importantes que l'utilisation de plusieurs réseaux [35]. La dernière stratégie proposée dans [33] consiste à recoder l'ensemble des vecteurs $\mathbf{x}^z$ à l'aide du codage d'Huffman pour créer les vecteurs $\mathbf{m}^z$. Les vecteurs $\mathbf{m}^z$ sont ensuite stockés dans un réseau à clusters. Cette dernière stratégie offre une capacité de stockage plus importante que l'utilisation de plusieurs réseaux. Toutefois, les dégradations ne peuvent impacter les vecteurs $\mathbf{x}^z$ originaux ce qui limite les cas d'utilisation.

Dans [34], les auteurs proposent d'améliorer la stratégie d'augmentation du nombre de neurones en allouant les neurones dynamiquement lors de la phase d'apprentissage. L'allocation est faite à partir du taux d'activation d'une composante ce qui conduit à allouer plus de neurones aux composantes fréquemment actives. Cette solution est sans doute la plus efficace pour répondre au problème de vecteurs non-uniformément distribués.

4.2 Hétéro-association

Nous proposons ici une nouvelle approche permettant d'améliorer les performances des réseaux à clusters lorsque les vecteurs ne sont pas distribués de manière indépendante. Cette approche repose sur une version hétéro-associative des réseaux à clusters que nous appelons réseaux à clusters restreints.

4.2.1 Architecture et apprentissage

Un réseau à clusters restreint est un modèle de mémoire hétéro-associative dérivé du réseau à clusters. Il se compose de deux couches de neurones. Une couche d'entrée, qui est associée aux vecteurs $\mathbf{x}^z$ à stocker et une couche cachée. La couche d'entrée est organisée en c clusters de taille ℓ et la couche cachée en c' clusters de taille ℓ'. Alors que les paramètres c et ℓ dépendent de la structure des vecteurs $\mathbf{x}^z$, les paramètres c' et ℓ' sont choisis lors de la conception du réseau. Ces deux couches sont reliées par une matrice de connexions binaires $W \in \{0,1\}^{c\ell \times c'\ell'}$. Chaque connexion $w_{(i,j)(i',j')}$ relie le j^{eme} neurone du cluster i de la couche d'entrée avec le j'^{eme} neurone du cluster i' de la couche cachée.

Afin d'être stocké dans le réseau, chaque vecteur $\mathbf{x}^z$ est associé à un vecteur $\mathbf{y}^z$ lors de la phase d'apprentissage. Les vecteurs $\mathbf{y}^z$ sont générés aléatoirement, de manière uniforme et indépendante. La matrice de connexions W est ensuite mise à jour à l'aide de la règle de Hebb saturée :

$$w^z_{(i,j)(i',j')} = w^{(z-1)}_{(i,j)(i',j')} \vee x^z_{(i,j)} y^z_{(i',j')}, \tag{4.1}$$

avec $w^z_{(i,j)(i',j')}$ l'état de la connexion entre les neurones (i,j) et (i',j') après l'apprentissage de la paire de vecteurs $(\mathbf{x}^z, \mathbf{y}^z)$. La figure 4.2 représente un réseau à clusters restreint constitué de quatre clusters de quatre neurones dans la couche

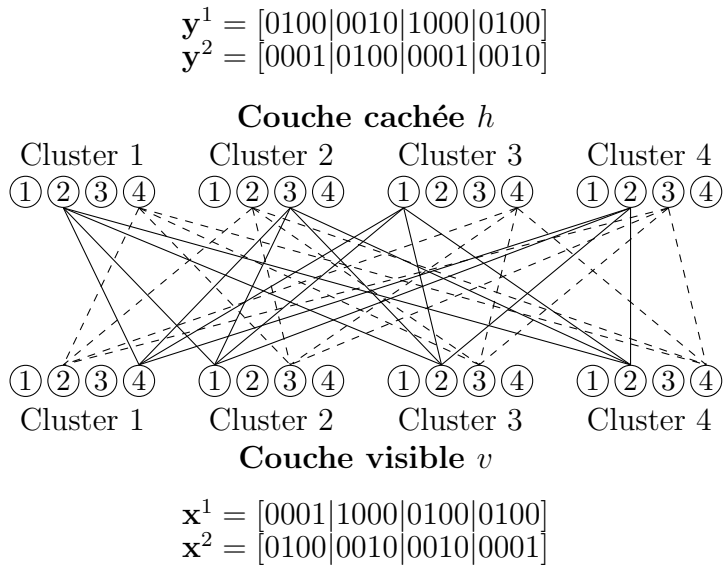


FIGURE 4.2 – Réseau à clusters restreint.

d'entrée et dans la couche cachée. Ce réseau est utilisé pour stocker deux vecteurs $\mathbf{x}^1$ et $\mathbf{x}^2$, qui sont chacun associés à un vecteur ($\mathbf{y}^1$ et $\mathbf{y}^2$) généré aléatoirement.

4.2.2 Décodage

L'objectif de la phase de décodage est de retrouver le vecteur $\mathbf{x}^z$ à partir de $\tilde{\mathbf{x}}^z$, une version dégradée de ce vecteur. Pour ce faire, l'algorithme de décodage s'aide de la représentation cachée $\mathbf{y}^z$. Tout d'abord, l'état de la couche d'entrée est initialisé à l'aide du vecteur dégradé. Puis, un algorithme itératif est utilisé pour retrouver $\mathbf{x}^z$. Chaque itération se décompose en deux phases, la propagation avant, de la couche d'entrée vers la couche cachée, et la propagation arrière, de la couche cachée vers la couche d'entrée. Chacune de ces propagations consiste à appliquer une règle dynamique $dyn()$ puis une règle d'activation $act()$ afin de déterminer l'état de la couche cible à partir de l'état de la couche de départ et de la matrice de connexion W. Lors de la propagation arrière, la transposée de W est utilisée. La figure 4.3 résume le processus de décodage.

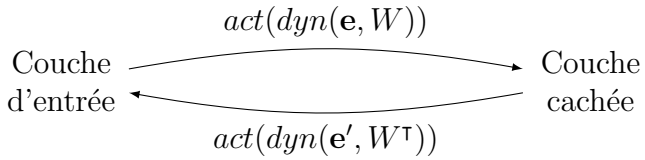


FIGURE 4.3 – Processus de décodage d'un réseau de clusters restreint : $\mathbf{e}$ et $\mathbf{e}'$ sont les vecteurs d'état de la couche d'entrée et et de la couche cachée.

Le choix des règles dynamiques et d'activation peut être fait parmi les différentes règles utilisées pour les réseaux à clusters. Elles peuvent être différentes pour la propagation avant et la propagation arrière. C'est d'ailleurs le cas lorsqu'un réseau est utilisé pour stocker des vecteurs denses. Nous allons présenter plus en détails le processus de décodage dans ce contexte particulier.

4.3 Évaluation des réseaux à clusters restreints

Nous allons maintenant évaluer les performances d'un réseau à clusters restreint. Cette évaluation est faite sur la tâche la plus communément utilisée pour évaluer les réseaux à clusters [6, 15, 16, 17, 33, 34, 35]. Cette tâche consiste à retrouver un vecteur $\mathbf{x}^z$ stocké dans le réseau lorsque ce dernier subit une dégradation de type effacement : c'est-à-dire qu'une partie des composantes à un du vecteur a été mise à zéro.

4.3.1 Évaluation des paramètres c' et ℓ'

Dans un premier temps, nous évaluons les performances d'un réseau à clusters restreint en fonction des paramètres c' et ℓ'. Ces paramètres permettent de déterminer le nombre de clusters de la couche cachée et leur taille. Cette évaluation est faite dans un cas optimal d'utilisation, c'est-à-dire lorsque les vec-

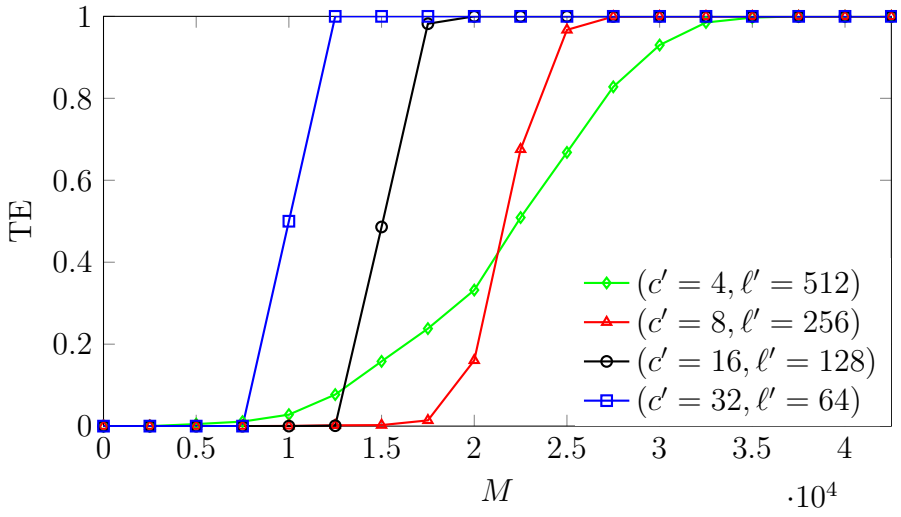


FIGURE 4.4 – Évolution du taux d'erreur (TE) en fonction du nombre de vecteurs M stockés pour différentes valeurs de c' et ℓ'. Pour chaque réseau, la couche d'entrée est constituée de $c = 8$ clusters de $\ell = 256$ et le nombre de neurones dans la couche cachée est égal à $d' = c'\ell' = 2048$.

teurs $\mathbf{x}^z$ stockés dans le réseau sont générés de manière uniforme et indépendante.

Nous utilisons ici des vecteurs de dimension $d = 2048$ divisible en huit sous-vecteurs de dimension 256. Chacun de ces sous-vecteurs ayant une composante égale à un. La structure des vecteurs $\mathbf{x}^z$ détermine l'organisation de la couche d'entrée d'un réseau à clusters restreint. Cette couche comporte donc $c = 8$ clusters de $\ell = 256$ neurones. Nous faisons évoluer les paramètres c' et ℓ' de manière à conserver le même nombre de neurones $d' = c'\ell' = 2048$ dans la couche cachée. Chaque configuration testée utilise donc une matrice de connexions $W \in \{0,1\}^{2048\times2048}$ et nécessite la même quantité mémoire.

Lors de la phase de test, les vecteurs présentés au réseau sont obtenus en effaçant quatre de leurs composantes. L'algorithme de décodage est ensuite itéré deux fois pour retrouver le vecteur d'origine. Chaque itération étant composée d'une propagation avant et d'une propagation arrière, nous utilisons pour les

deux propagations la règle Sum-of-Max (SoM) comme règle dynamique et la règle Local-Winner-take-All (L-WtA) comme règle d'activation.

La figure 4.4 rend compte de l'évolution du taux d'erreur en fonction du nombre M de vecteurs stockés pour quatre réseaux. On peut observer que lorsque c' est élevé, le taux d'erreur reste proche de zéro pour de faibles valeurs de M puis évolue rapidement pour atteindre un. A l'opposé, de faibles valeurs pour c' conduisent à une évolution du taux d'erreur en fonction de M beaucoup plus progressive. Le réseau permettant de stocker le plus grand nombre de vecteurs tout en maintenant un faible taux d'erreur (< 0.01) est le réseau le plus équilibré ($c' = c$ et $\ell' = \ell$). Dans la suite des expériences, nous utiliserons uniquement des réseaux équilibrés.

4.3.2 Comparaison avec un réseau à clusters

Nous comparons maintenant les performances d'un réseau à clusters restreint avec celles d'un réseau à clusters classique. La structure des vecteurs $\mathbf{x}^z$ est la même que précédemment, ils sont de dimension $d = 2048$ et sont divisés en huit sous-vecteurs de dimension 256. Pour stocker ces vecteurs, nous utilisons un réseau à clusters (RC) constitué de $c = 8$ clusters de $\ell = 256$ neurones et un réseau à clusters restreint (RCR) constitué de $c = c' = 8$ clusters de $\ell = \ell' = 256$ neurones. Lors de la phase de décodage, nous utilisons les règles Sum-of-Max et Local-Winner-take-All pour les deux réseaux. L'algorithme de décodage est itéré quatre fois pour le réseau à clusters et deux fois pour le réseau à clusters restreint (chaque itération comprend une propagation avant et une propagation arrière). Ces deux réseaux n'utilisent pas le même nombre de bits pour stocker la matrice de connexions W. Le nombre de bits B utilisé par les deux réseaux peut être

calculé à l'aide de ces deux équations :

$$C_{RC} = \frac{c(c-1)\ell^2}{2} \text{ et } C_{RCR} = cc'\ell\ell'. \tag{4.2}$$

Afin de comparer les performances de ces deux réseaux nous introduisons une nouvelle métrique que nous appelons *efficacité mémoire*. L'efficacité mémoire η est égale au ratio entre C, la quantité de bits stockés dans le réseau et B le nombre de bits utilisés par le réseau. Pour calculer C nous considérons que chaque vecteur $\mathbf{x}^z$ peut être stocké sous la forme c index codé sur $\log_2(\ell)$, B est alors égal à :

$$B = Mc\log_2(\ell), \tag{4.3}$$

avec M le nombre de vecteurs stockés dans le réseau.

Nous comparons les deux réseaux sur deux jeux différents de données. Le premier jeu de données contient des vecteurs générés aléatoirement tandis que le second est constitué de vecteurs générés à partir de données naturelles.

Vecteurs générés aléatoirement

Les réseaux à clusters restreints sont une extension des réseaux à clusters qui permettent d'améliorer le stockage de vecteurs non-indépendamment distribués. Afin de comparer les deux types de réseaux dans ce contexte, nous générons les vecteurs $\mathbf{x}^z$ de la manière suivante :

$$\forall i, i' \in [1, ..., c], i \neq i' \text{ et } j, j' \in [1, ..., \ell],$$
$$\mathbb{P}(I_i = j, I_{i'} = j') = \begin{cases} \left(\frac{k}{\ell}\right)^2 \text{ si } (j \bmod k) = (j' \bmod k), \\ 0 \text{ sinon} \end{cases},$$

avec I_i l'index de la composante à un dans le sous-vecteur $\mathbf{x}_i^z$ et k un paramètre permettant de régler la dépendance entre les composantes de $\mathbf{x}^z$. Si $k = 1$, les

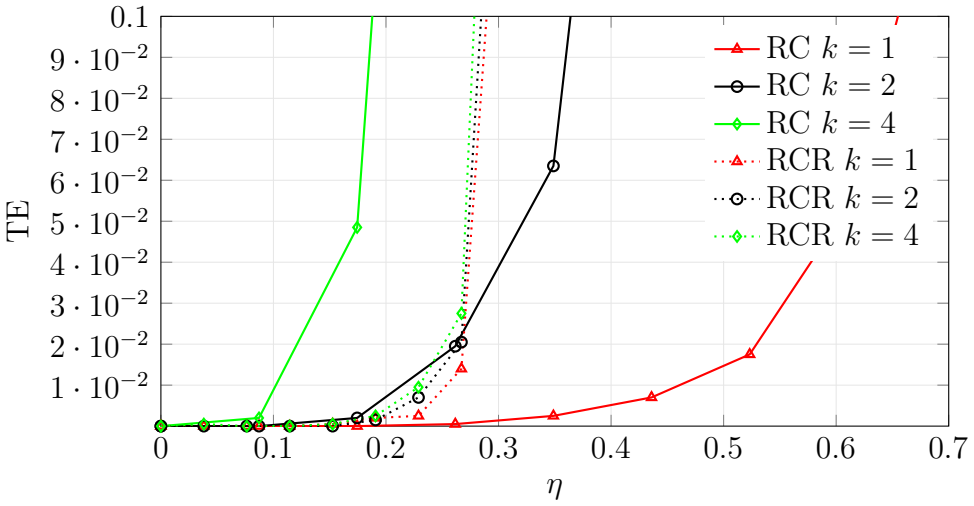


FIGURE 4.5 – Évolution du taux d'erreur (TE) en fonction de l'efficacité mémoire η pour différentes valeurs de k. Les vecteurs sont uniformément distribués.

vecteurs sont indépendamment distribués. Si $k = 2$, les index I_i d'un même vecteur sont soit pairs soit impairs. Nous évaluons les deux réseaux pour des vecteurs uniformément et non-uniformément distribués. Pour les vecteurs non-uniformément distribués, les composantes à un dans les sous-vecteurs $\mathbf{x}_i^z$ sont générées à partir d'un distribution gaussienne tronquée.

Les figures 4.5 et 4.6 rendent compte de l'évolution du taux d'erreur TE en fonction de l'efficacité mémoire η. Lors de la phase de test, les vecteurs présentés au réseau sont obtenus en effaçant quatre de leurs composantes. Pour le réseau à clusters, on observe que l'augmentation du paramètre k dégrade fortement la capacité du réseau à retrouver un vecteur. Lorsque les vecteurs sont uniformément distribués (figure 4.5), le taux d'erreur pour $\eta = 0.26$ et $k = 1$ est de 5.10^{-4}. Pour $k = 2$ le taux d'erreur est de 0.019. Pour $k = 4$ le taux d'erreur est de 0.38. Même si l'augmentation de k affecte les performances du réseau à clusters restreint, la dégradation est beaucoup plus légère. Pour une efficacité mémoire de 0.27, le taux d'erreur pour $k = 1$ est de 0.014, pour $k = 2$ le taux d'erreur est de 0.02

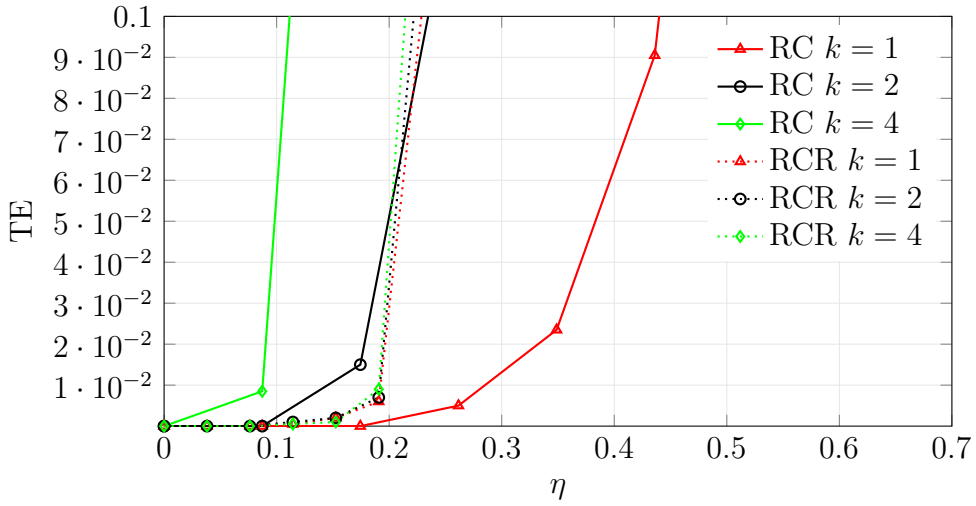


FIGURE 4.6 – Évolution du taux d'erreur (TE) en fonction de l'efficacité mémoire η pour différentes valeurs de k. Les vecteurs sont non-uniformément distribués.

et de 0.027 pour $k = 4$. On peut faire la même observation pour des vecteurs non-uniformément distribués (figure 4.6).

Vecteurs générés à partir de données naturelles

Nous testons maintenant les deux réseaux sur des vecteurs générés à partir de données naturelles. Nous utilisons pour cela des vecteurs SIFT [30] extraits de la base SIFT1M [29]. Afin d'être stockés, les vecteurs SIFT $\mathbf{v}^z \in \mathbb{R}^{128}$ sont encodés de la même manière que dans le paragraphe 3.4.2 à l'aide de la méthode d'encodage présentée dans le paragraphe 3.2.2. Cette méthode, basée sur le produit de quantification [29], permet de créer les vecteurs $\mathbf{x}^z \in \{0,1\}^{c\ell}$ tels que chaque sous-vecteur $\mathbf{x}_i^z \in \{0,1\}^{\ell}$ ait une seule composante à un. Nous utilisons ici $c = 8$ et $\ell = 256$. Une fois les vecteurs $\mathbf{x}^z$ générés, nous reproduisons la même expérience que dans le paragraphe précédent. Les vecteurs $\mathbf{x}^z$ sont d'abord stockés dans le réseau. Lors de la phase de test, un des vecteurs stockés est tiré au hasard et la moitié de ses composantes est effacée. Il est ensuite présenté au réseau qui itère

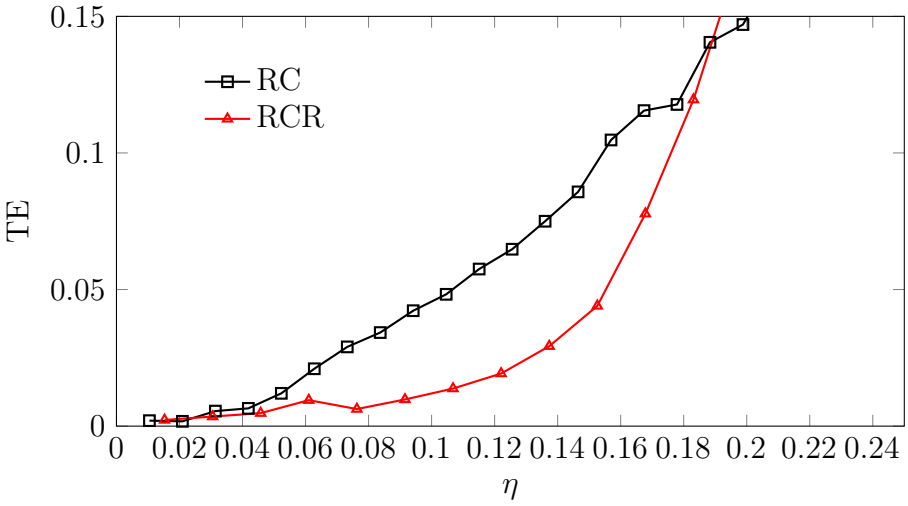


FIGURE 4.7 – Évolution du taux d'erreur (TE) en fonction de l'efficacité mémoire η pour différentes valeurs de k. Les vecteurs sont générés à partir de vecteurs SIFT.

son algorithme de décodage afin de retrouver le vecteur d'origine.

La figure 4.7 rend compte pour chacun des réseaux du taux d'erreur en fonction de l'efficacité mémoire. On observe que le réseau à clusters restreint retrouve plus facilement les vecteurs pour une même efficacité mémoire que le réseau à clusters.

Recherche de plus proche voisin

Nous allons maintenant comparer les deux réseaux lorsqu'ils sont utilisés pour réaliser une recherche du plus proche voisin sur des vecteurs denses. Nous utilisons dans cette expérience des vecteurs SIFT extraits de la base SIFT1M [29]. Comme dans l'expérience précédente, ces vecteurs sont encodés afin de produire des vecteurs $\mathbf{x}^z$ avec $c = 8$ et $\ell = 256$.

Lors de la phase de test, plusieurs vecteurs requêtes sont présentés au réseau à clusters restreint. A chaque nouveau vecteur, le réseau est tout d'abord initialisé. Cette initialisation consiste à calculer des potentiels d'action pour chacun des

neurones de la couche d'entrée (voir paragraphe 3.3.1) et à activer la totalité des neurones du réseau (couche d'entrée et couche de sortie). Puis le processus de décodage est itéré jusqu'à ce que le plus proche voisin soit retrouvé. Chaque itération du processus de décodage se décompose en une propagation avant et une propagation arrière (voir figure 4.3). Lors de la propagation avant, nous utilisons la règle décrite par l'équation 3.13 comme règle dynamique et la règle k-GLkO (voir paragraphe 3.3.2) comme règle d'activation. Pour la règle k-GLkO, nous utilisons les paramètres $k^{init} = 1000$ et $\alpha = 0.2$. Lors de la propagation arrière, les neurones de la couche cachée n'ayant pas de potentiel d'action, nous utilisons une règle Sum-of-Max classique comme règle dynamique et la règle Global-Winner-take-All comme règle d'activation.

La figure 4.8 rend compte de l'évolution du taux d'erreur et du nombre d'itérations en fonction de l'efficacité mémoire η. Sur cette courbe, sont également reportés les résultats obtenus avec un réseau à clusters, pour les mêmes valeurs de α et de k^{init}. Pour le réseau à clusters restreint, le taux d'erreur est très proche de zéro tant que η est inférieur à 0.023. Au-delà de cette valeur, le taux d'erreur se met à croître de manière exponentielle. Pour le réseau à clusters, le taux d'erreur est toujours supérieur à zéro et croît de manière quasi linéaire pour $\eta \in [0, 0.04]$. Pour les deux réseaux, le nombre d'itérations reste constant.

Discussion

Pour chacune des comparaisons, nous considérons que le nombre de bits nécessaire aux deux réseaux est calculé à l'aide des équations 4.2. Pour un réseau à clusters classique cette équation prend en compte le fait que la matrice de connexion W est symétrique et qu'il ne peut pas y avoir de connexion intra-

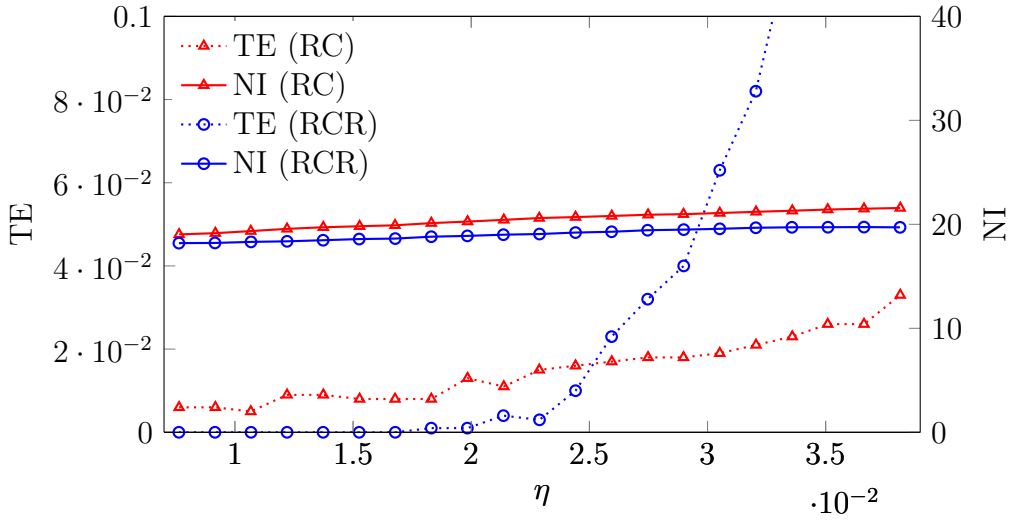


FIGURE 4.8 – Recherche du plus proche voisin sur des vecteurs SIFT. Évolution du taux d'erreur (TE) et du nombre d'itérations (NI) en fonction de l'efficacité mémoire η pour un réseau à clusters restreint (RCR) et un réseau à clusters (RC).

cluster. Cependant la majorité des architectures matérielles mettant en œuvre des réseaux à clusters ne tiennent pas compte de la symétrie de la matrice de connexion [19, 20, 22, 21, 23]. En effet, exploiter la symétrie de la matrice nécessite d'ajouter des structures de contrôle supplémentaires qui complexifient les architectures matérielles. Si la symétrie de la matrice W n'est pas prise en compte, le nombre de bits nécessaire pour la stocker est alors égal à :

$$C_{RC} = c(c-1)\ell^2. \tag{4.4}$$

Ce mode de calcul impacte fortement l'efficacité mémoire d'un réseau à clusters. Malgré un mode de calcul défavorable le réseau à clusters restreint offre de meilleurs performances en ce qui concerne le taux d'erreur dans la totalité des expériences que nous avons réalisées.

4.4 Conclusion

Dans ce chapitre, nous avons abordé le problème du stockage de vecteurs corrélés dans un réseau à clusters. Si ce type de réseau offre une grande capacité de stockage lorsque les vecteurs sont uniformément et indépendamment distribués, les performances chutent dans le cas contraire. Les maigres performances des réseaux à clusters lorsque les vecteurs sont corrélés limitent leur utilisation dans des applications réelles, notamment en vision par ordinateur. En effet, les données extraites de notre environnement, sont le plus souvent hautement corrélées.

Les précédents travaux portant sur le stockage de vecteurs corrélés se sont tous concentrés sur le problème de non-uniformité. Les travaux présentés ici sont les premiers à s'intéresser au stockage de vecteurs non-indépendamment distribués dans un réseau à clusters. Lorsque les vecteurs ne sont pas indépendamment distribués, certaines composantes des vecteurs ont tendance à s'activer ensemble provoquant une surutilisation de certaines connexions du réseau.

Afin de répondre au problème du stockage de vecteurs non-indépendamment distribués, nous avons proposé d'utiliser le principe de l'hétéro-association. Les vecteurs $\mathbf{x}^z$ que l'on cherche à stocker sont associés à des vecteurs $\mathbf{y}^z$ qui sont uniformément et indépendamment distribués, générés aléatoirement. Afin de stocker ces paires de vecteurs $(\mathbf{x}^z, \mathbf{y}^z)$, nous avons proposé une version hétéro-associative des réseaux à clusters appelée réseau à clusters restreint [36]. Ce réseau utilise un algorithme de décodage similaire aux réseaux à clusters.

Nous avons montré à travers différentes expériences la robustesse des réseaux à clusters restreints lorsque les vecteurs ne sont pas indépendamment distribués. Ces expériences ont été réalisées à la fois sur des vecteurs générés aléatoirement et sur des vecteurs générés à partir de données naturelles.

Chapitre 5

Quantification de réseaux convolutifs

5.1 Introduction

Dans les précédents chapitres nous avons présenté différents travaux portant sur les réseaux à clusters ou des extensions de ceux-ci. L'objectif était d'utiliser ce type de réseau pour des applications de vision par ordinateur. Dans un premier temps, nous avons proposé différentes modifications permettant de l'utiliser pour faire une recherche du plus proche voisin sur des vecteurs denses. Dans un second temps, nous avons proposé une version hétéro-associative des réseaux à clusters qui permet d'améliorer les performances de ce type de réseau lorsque les vecteurs sont corrélés.

Dans ce chapitre nous nous intéressons à une autre famille de réseaux de neurones, les réseaux convolutifs. Ce type de réseau est actuellement très utilisé pour des applications de vision par ordinateur notamment en classification d'images. Il détient depuis 2012 les meilleurs performances en matière de taux

de classification. Si les réseaux convolutifs offrent de bons taux de classification, ils consomment une grande quantité de ressources (mémoire et calcul) ce qui limite leur utilisation sur des systèmes embarqués. Afin d'étendre toujours plus le champ applicatif des réseaux convolutifs, beaucoup de recherches se concentrent sur leur simplification.

Après avoir présenté le fonctionnement des réseaux convolutifs, nous présenterons différents travaux qui visent à les simplifier afin de les utiliser sur des systèmes embarqués. Nous proposerons ensuite une approche qui utilise la quantification vectorielle et notamment le produit de quantification pour à la fois compresser et accélérer un réseau convolutif. Enfin, nous enrichirons cette approche en présentant comment un réseau quantifié peut être ré-entraîné.

5.2 Réseaux convolutifs

5.2.1 Classification automatique d'images

La classification automatique d'images est une application de vision par ordinateur qui consiste à attribuer une classe à une image à l'aide d'un système de classification. Un système de classification d'images peut être divisé en deux sous-systèmes. Le premier sous-système, appelé *extracteur de caractéristiques*, est responsable de la transformation des données brutes en une représentation synthétique. Le second sous-système, appelé *classifieur*, est chargé de classer ces représentations.

Un extracteur de caractéristiques est un système complexe. Il se compose le plus souvent de différents niveaux de représentation. Les premiers niveaux correspondant à des caractéristiques locales, c'est-à-dire des portions de l'image

d'entrée (pixels, patchs...) ; tandis que les derniers niveaux correspondent à des caractéristiques globales obtenues par l'agrégation de caractéristiques des niveaux précédents. À chaque niveau de représentation, différents choix doivent être faits : quel type de caractéristiques utiliser, comment agréger les caractéristiques du niveau précédent... Tous ces choix nécessitent un haut niveau d'expertise.

Un classifieur est un système qui utilise une fonction de décision pour classer des images. Cette fonction est construite à l'aide d'une base d'entraînement utilisée comme référence. On peut distinguer deux familles de classifieur. La première est celle des classifieurs non-paramétriques, qui ne nécessitent pas de phase d'apprentissage. Les exemples de l'ensemble d'entraînement sont directement utilisés dans la fonction de décision. La plus connue de ces méthodes est l'algorithme des k-plus proches voisins. A l'opposé, la famille des classifieurs paramétriques, utilise l'ensemble d'entraînement lors d'une phase d'apprentissage, pour calculer les paramètres de la fonction de décision. On peut nommer dans cette famille, les classifieurs SVM (*support vector machine*, machine à vecteurs de support), qui sont une généralisation des classifieurs linéaires [37].

La principale idée des réseaux convolutifs consiste à entraîner un système hiérarchique de classification d'images, extracteur de caractéristiques et classifieur, à l'aide d'un unique algorithme d'apprentissage. Ce système hiérarchique, inspiré du système de vision biologique, permet alors d'utiliser un nombre important de niveaux de représentation de l'image fournie en entrée.

5.2.2 Architecture

Un réseau convolutif se compose d'un ensemble de couches de neurones, chacune offrant une nouvelle représentation des données fournies en entrée. Chaque

couche est connectée à la suivante à l'aide d'un ensemble de paramètres w appelés poids. L'activité a_i^ℓ du neurone i de la couche ℓ, connecté aux neurones $\{j_1, ..., j_n\}$ de la couche $\ell - 1$ est calculée de la manière suivante :

$$a_i^\ell = f\left(b_i^\ell + \sum_{x=1}^{n} w_{i,j_x}^\ell a_{j_x}^{\ell-1}\right), \tag{5.1}$$

avec $w_{i,j}^\ell$ le poids de la connexion entre le neurone i de la couche ℓ et le neurone j de la couche $\ell - 1$, b_i^ℓ le biais associé au neurone i de la couche ℓ et $f()$ une fonction d'activation non-linéaire. La fonction d'activation la plus couramment utilisée est le rectifieur linéaire (ReLU) :

$$\mathrm{ReLU}(x) = \max(0, x). \tag{5.2}$$

Les architectures utilisées en vision par ordinateur sont inspirées par le système visuel biologique. La figure 5.1 représente de manière graphique un réseau convolutif. Pour simplifier la figure l'entrée du réseau est unidimensionnelle. Ces réseaux utilisent deux types de couches de neurones, des couches dites convolutives et des couches dites denses.

Couches convolutives

Les couches convolutives sont inspirées par les premières aires du cortex visuel. Dans ces aires, les neurones sont sensibles, d'une part à des motifs visuels locaux, d'autre part, à un même motif à différentes positions dans le champ visuel. Afin de reproduire ce comportement, les neurones des couches convolutives mettent en œuvre deux idées majeures. Premièrement, la localité spatiale de la zone visuelle associée à un neurone. Deuxièmement, le partage des poids entre neurones.

Les couches convolutives sont utilisées dans les premiers niveaux du réseau et conservent les contraintes spatiales de l'image fournie en entrée. Chaque neurone

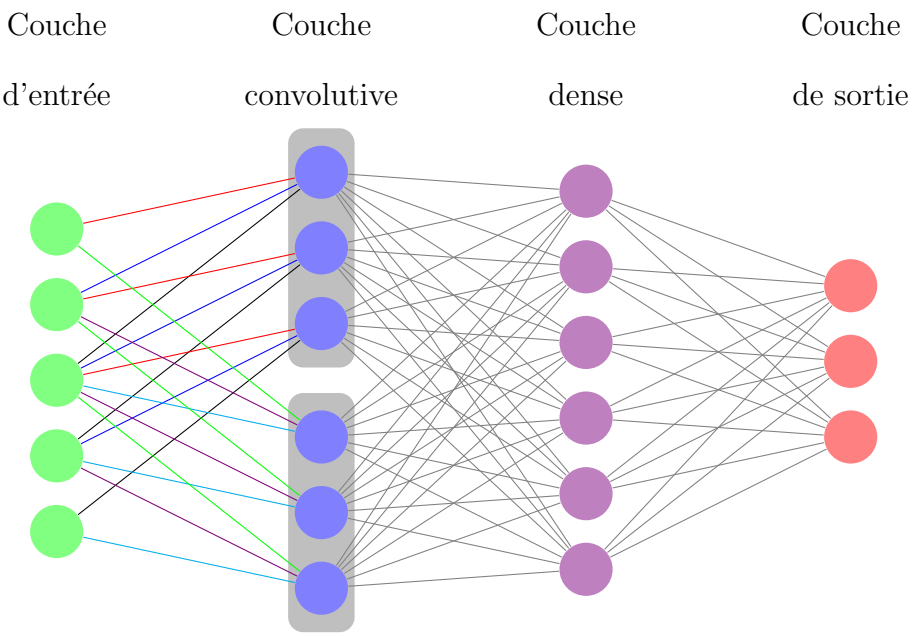


FIGURE 5.1 – Représentation graphique d'un réseau convolutif.

est connecté à un groupe de neurones de la couche précédente appartenant à un même voisinage spatial. Cela permet de les rendre sensibles à des motifs visuels locaux tels que des arêtes orientées ou des angles. Sur la figure 5.1, chaque neurone de la couche convolutive est connecté à trois neurones de la couche d'entrée. En outre, les neurones sont organisés en cartes de caractéristiques (2 sur la figure 5.1). Les neurones d'une même carte partagent les mêmes poids (symbolisés par des arêtes d'une même couleur sur la figure 5.1) et sont donc sensibles à un même motif visuel. Une couche convolutive est alors définie à l'aide de trois hyper-paramètres :

— K la taille de la zone associée à chaque neurone (pour des images la zone est alors de $K \times K$) ;

— F le nombre de cartes de caractéristiques ;

— S le saut entre deux neurones appartenant à la même carte de caractéristiques.

L'opération mathématique associée aux couches convolutives est la convolution

discrète (2D pour des images, 1D sur la figure 5.1). Ces couches jouent le rôle d'extracteur de caractéristiques locales.

Couches denses

Les couches denses, à l'opposé des couches convolutives, mettent en œuvre un modèle de connectivité globale. Tous les neurones d'une couche ℓ sont connectés aux neurones de la couche $\ell - 1$ (voir couche dense de la figure 5.1). Une couche est alors définie par le nombre N^ℓ de neurones qu'elle contient. Les couches denses jouent le rôle d'extracteur de caractéristiques globales et l'opération mathématique utilisée pour calculer l'activité des neurones est alors le produit matriciel.

La dernière couche d'un réseau convolutif utilisée pour la classification est une couche dense particulière qui joue le rôle de classifieur. Le nombre de neurones qui la constitue est égale à C, le nombre de classes utilisées pour trier les images. Le réseau représenté sur la figure 5.1 peut être utilisé pour classer des données appartenant à trois catégories différentes. La fonction d'activation utilisée diffère des précédentes couches. Couramment, la fonction *softmax* est utilisée et rend compte de la probabilité que l'image d'entrée appartienne à telle ou telle classe. Si $\mathbf{x}$ est le vecteur de sortie de la dernière couche du réseau (sans application d'une fonction d'activation), la sortie prédite $\hat{\mathbf{y}}$ par le réseau est alors égale à :

$$\forall i \in [1, ..., C], \hat{y}_i = softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}. \tag{5.3}$$

5.2.3 Entraînement des réseaux convolutifs

L'une des forces des réseaux convolutifs est d'utiliser un même algorithme d'apprentissage pour calculer l'ensemble des paramètres du réseau. Un algorithme d'apprentissage est un algorithme qui vise à minimiser une fonction objectif

$\mathcal{L}(\theta, \mathcal{D})$, avec $\theta = \{W, b\}$ les paramètres du réseaux et $\mathcal{D}$ un ensemble d'entraînement. Pour une tâche de classification d'images, l'ensemble est constitué d'images labellisées et peut être noté $\mathcal{D} = \{d_1 = (I^1, \mathbf{y}^1), ..., d_N = (I^n, \mathbf{y}^n)\}$ où I^i est une image et $\mathbf{y}^i \in \{0, 1\}^C$ le vecteur attendu en sortie. Ce vecteur est construit de la manière suivante :

$$\forall j \in [1, ..., C], y^i_j = \begin{cases} 1 \text{ si } I^i \text{ appartient à la classe } j \\ 0 \text{ sinon} \end{cases} \tag{5.4}$$

La fonction objectif a pour but de comparer la réponse apportée par le réseau $\hat{y}^i = net(I^i)$ avec la réponse attendue y^i. En classification multi-classes, la fonction d'entropie croisée est utilisée. Cette fonction permet de comparer deux distributions entre elles :

$$\mathcal{L}(\theta, \mathcal{D}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y^i_j \log(\hat{y^i_j}). \tag{5.5}$$

L'algorithme utilisé pour entraîner les réseaux convolutifs est l'algorithme de descente de gradient. Cet algorithme se décompose en trois phases : la phase de **propagation avant** ; la phase de **rétropropagation** ; la phase de **mise à jour**. Lors de la phase de propagation avant, la sortie du réseau est calculée pour chaque image de la base d'entraînement $\mathcal{D}$. Pendant la phase de rétro-propagation, il s'agit de calculer la fonction objectif $\mathcal{L}$, puis les gradients des paramètres du réseau (les poids w et les biais b) :

$$\Delta w = \frac{\partial \mathcal{L}}{\partial w}, \Delta b = \frac{\partial \mathcal{L}}{\partial b}. \tag{5.6}$$

Le calcul des gradients est possible car $\mathcal{L}$ est calculée par une composition de fonctions différentiables. Ces gradients sont ensuite utilisés pour mettre à jour les paramètres du réseau :

$$w^{t+1} = w^t - \alpha \Delta w, b^{t+1} = b^t - \alpha \Delta b, \tag{5.7}$$

avec α le taux d'apprentissage permettant de fixer la vitesse d'apprentissage. Après avoir itéré cette procédure suffisamment de fois, le réseau peut être réutilisé sur de nouvelles données.

En règle générale, l'ensemble d'entraînement $\mathcal{D}$ étant composé de plusieurs milliers d'images (1 million pour ImageNet [38]), on utilise une variante de l'algorithme de descente de gradient appelée descente de gradient stochastique. Dans cette variante, un sous-ensemble de $\mathcal{D}$, appelé batch, est utilisé à chaque itération de l'algorithme.

Rétropropagation du gradient

Nous allons maintenant présenter plus en détails la phase de rétropropagation du gradient dans un réseau de neurones. Pour cela, nous considérons un réseau formé de deux couches denses constituées de n^1 et n^2 neurones, qui prend en entrée un vecteur $\mathbf{x} \in \mathbb{R}^{n^0}$. Nous considérons également que chacune des deux couches ne possède pas de biais afin de simplifier la présentation. L'activation des neurones dans la première première couche est égale à :

$$\forall i \in [1, ..., n^1], y_i^1 = f(e_i^1) = f(\sum_{j=1}^{n^0} x_j w_{j,i}^1), \tag{5.8}$$

dans la seconde couche, l'activation est égale à :

$$\forall k \in [1, ..., n^2], y_k^2 = g(e_k^2) = g(\sum_{i=1}^{n^1} y_i^1 w_{k,j}^2), \tag{5.9}$$

et la fonction objectif est égale à :

$$\mathcal{L} = \sum_{k=1}^{n^2} h(y_k^2, t_k). \tag{5.10}$$

Une fois la propagation avant terminée, la phase de rétropropagation commence par le calcul de l'erreur pour les neurones de la couche deux. Pour un

neurone k, l'erreur correspond à la dérivée de $\mathcal{L}$ par rapport à e_k^2 :

$$\frac{\partial \mathcal{L}}{\partial e_k^2} = \frac{\partial \mathcal{L}}{\partial y_k^2} \frac{\partial y_k^2}{\partial e_k^2} = h'(y_k^2) g'(e_k^2). \tag{5.11}$$

Une fois l'erreur calculée, il est ensuite possible de calculer les gradients $w_{k,j}^2$ des poids de la couche deux ainsi que l'erreur dans la couche un :

$$\frac{\partial \mathcal{L}}{\partial w_{k,j}^2} = \frac{\partial \mathcal{L}}{\partial e_k^2} \frac{\partial e_k^2}{\partial w_{k,j}^2} = \frac{\partial \mathcal{L}}{\partial e_k^2} y_i^1, \tag{5.12}$$

$$\frac{\partial \mathcal{L}}{\partial e_j^1} = \sum_{k=1}^{n^2} \frac{\partial \mathcal{L}}{\partial e_k^2} \frac{\partial e_k^2}{\partial y_j^1} \frac{\partial y_j^1}{\partial e_j^1} = \sum_{k=1}^{n^2} \frac{\partial \mathcal{L}}{\partial e_k^2} w_{k,j}^2 f'(e_k^1). \tag{5.13}$$

Cette opération est répétée pour calculer les gradients $w_{j,i}^1$ des poids de la couche deux.

5.2.4 Réseaux convolutifs pour les systèmes embarqués

Au cours des dernières années, les performances en matière de taux de classification des réseaux convolutifs n'ont cessé de croitre. Cette augmentation s'est faite de paire avec une augmentation du coût de leurs mises en œuvre. En effet, les performances des réseaux convolutifs dépendent grandement du nombre de couches et du nombre de neurones contenus dans chaque couche. Afin d'obtenir des temps de traitement acceptables, ils sont le plus souvent entraînés et exécutés sur des architectures massivement parallèles telles que des cartes graphiques.

Le coût de mise en œuvre des réseaux convolutifs peut s'analyser selon deux critères : la consommation en ressources mémoire d'une part et la consommation en ressources de calcul d'autre part. Le tableau 5.1 rend comptes de la consommation en ressources (mémoire/calcul) d'un réseau convolutif célèbre, AlexNet [2]. Ce tableau met en évidence ce qui est vrai pour la majorité des réseaux convolutifs. Les couches convolutives sont les plus gourmandes en matière de ressources

Couche	**Méga Octet**	**Méga FLOP**
conv1	0.14 (0.06%)	105 (15%)
conv2	1.23 (0.5%)	224 (31%)
conv3	3.54 (1.4%)	150 (21%)
conv4	2.65 (1.1%)	112 (15%)
conv5	1.77 (0.7%)	74 (10%)
dense1	152 (62%)	37 (5%)
dense2	68 (28%)	17 (2%)
dense3	16 (6.5%)	4 (0.5%)
Total	**245**	**724**

TABLE 5.1 – Consommation en ressources (mémoire/calcul) de chacune des couches du réseau convolutif AlexNet.

de calcul ($\simeq$ 92%) tandis que les couches denses sont les plus gourmandes en terme de ressources mémoires ($\simeq$ 96%).

Dans l'objectif d'utiliser ces réseaux dans des applications mobiles, il y a dans les travaux récents un intérêt croissant sur la conception de réseaux moins gourmands en ressources. Afin d'y parvenir, deux approches sont explorées. La première de ces approches consiste à entraîner directement des réseaux *économes*. La structure des couches peut être simplifiée, c'est le cas par exemple des travaux présentés dans [39], les convolutions discrètes sur trois dimensions ($K \times K \times N^{\ell-1}$) sont remplacées par deux opérations de convolutions, une convolution sur deux dimensions ($K \times K$) suivie d'une convolution sur une seule dimension ($N^{\ell-1}$). Dans [40] les auteurs utilisent des goulots d'étranglement pour concevoir de petits réseaux. Il est également possible d'entraîner des réseaux dont les paramètres et

les variables possèdent une faible résolution. Dans [41, 42], les auteurs entraînent par exemple des réseaux utilisant des poids et des neurones binaires.

Une seconde approche consiste à considérer un réseau déjà entraîné et de lui appliquer des techniques de compression de données pour réduire sa taille. Plusieurs travaux utilisent la factorisation de matrices pour accélérer à la fois les couches convolutives [43, 44, 45] et compresser les couches denses [46, 47]. Dans [48] et [49], différentes méthodes de quantification vectorielle (dont le produit de quantification) sont testées pour compresser et accélérer les réseaux convolutifs. Cette seconde approche peut être utilisée conjointement avec la précédente.

Nous nous intéresserons dans la suite de ce chapitre plus particulièrement à la seconde approche. La méthode que nous proposons pour réduire le coût des réseaux convolutifs utilise le produit de quantification pour compresser un réseau déjà entraîné. Puis le réseau compressé est ré-entraîné à l'aide de l'algorithme de descente de gradient.

5.3 Entraînement de réseaux compressés

La compression d'un réseau revient à trouver une nouvelle architecture à partir d'une architecture d'origine. Cette nouvelle architecture est moins coûteuse en terme de ressources mais offre de moins bons taux de classification. Pour réduire la dégradation entraînée par l'étape de compression, nous proposons de ré-entraîner le réseau compressé. Ce ré-entraînement est fait en modifiant l'algorithme de descente de gradient pour que celui-ci prenne en compte la nouvelle architecture du réseau. Nous appliquons cette approche sur des réseaux compressés à l'aide du produit de quantification comme présenté dans [49]. Les expériences que nous réalisons montrent que le ré-entraînement d'un réseau compressé permet

d'améliorer significativement ses performances en matière de taux de classification.

5.3.1 Quantification d'un réseau convolutif à l'aide du produit de quantification

Le produit de quantification est une technique de quantification vectorielle couramment utilisée pour accélérer la recherche de plus proches voisins [29]. Elle permet d'estimer rapidement la distance euclidienne entre un vecteur requête et un vecteur de la base de recherche. Cette technique peut être utilisée à la fois pour compresser et accélérer un réseau convolutif.

Le produit vectoriel est l'opération qui domine l'exécution d'un réseau convolutif aussi bien dans les couches denses que convolutives. Formellement, un vecteur d'entrée $\mathbf{x} \in \mathbb{R}^D$ est multiplié avec un ensemble de vecteurs de poids $\mathcal{W} = \{\mathbf{w}^1, ..., \mathbf{w}^n\}, \mathbf{w}^z \in \mathbb{R}^D$. Comme présenté dans [49], le produit de quantification peut être utilisé pour réduire l'espace mémoire nécessaire pour stocker $\mathcal{W}$. Nous montrons qu'il permet également d'accélérer les produits vectoriels entre $\mathbf{x}$ et $\mathbf{w}^z \in \mathcal{W}$.

L'application du produit de quantification aux poids $\mathcal{W}$ revient à diviser l'espace vectoriel à D dimensions en m sous-espaces de dimension $c = \frac{D}{m}$. Chaque vecteur de poids $\mathbf{w}^z \in \mathcal{W}$ est alors vu comme la concaténation de m sous-vecteurs $\mathbf{w}_i^z \in \mathbb{R}^c$. Les sous-vecteurs $\mathbf{w}_i^z$ appartenant au même sous-espace vectoriel i sont ensuite quantifiés à l'aide de l'algorithme des k-means. Cet algorithme permet de calculer un dictionnaire $\mathcal{U}^i = \{\mathbf{u}_i^1, ..., \mathbf{u}_i^k\}$ constitué de k mots de code $\mathbf{u}_i^j$ en minimisant l'erreur de reconstruction. Une fois les dictionnaires calculés, chaque vecteur de poids $\mathbf{w}^z$ peut être représenté comme la concaténation de m mots de

code :

$$\mathbf{w}^z \simeq q(\mathbf{w}^z) = [q_1(\mathbf{w}_1^z), ..., q_m(\mathbf{w}_m^z)], q_i(\mathbf{w}_i^z) = \underset{\mathbf{u}_i^j \in \mathcal{U}^i}{\operatorname{argmin}} d(\mathbf{u}_i^j, \mathbf{w}_i^z), \tag{5.14}$$

avec $d(\mathbf{x}, \mathbf{y})$ la distance euclidienne entre deux vecteurs.

Seuls les dictionnaires $\mathcal{U}^i$ et les index des mots de code sont stockés en mémoire. Le nombre de bits nécessaire pour stocker $\mathcal{W}$ est alors égal à :

$$kD32 + nm \log_2 k, \tag{5.15}$$

en considérant une représentation flottante sur 32 bits. Les produits vectoriels entre $\mathbf{x}$ et les vecteurs $\mathbf{w}^z \in \mathcal{W}$ sont approximés de la manière suivante :

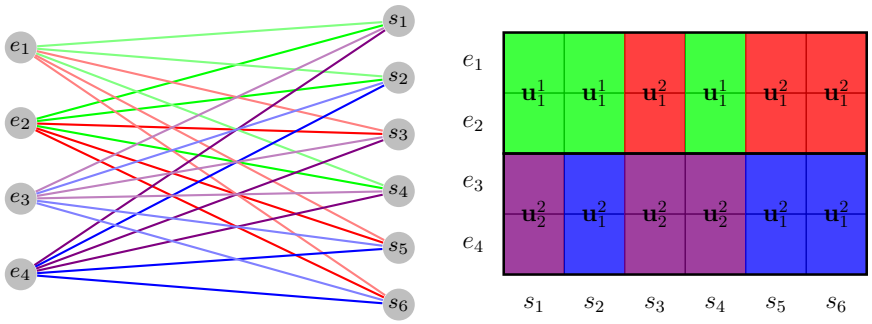
$$\mathbf{x} \cdot \mathbf{w}^z \simeq \mathbf{x} \cdot q(\mathbf{w}^z) = \sum_{i=1}^{m} \mathbf{x}_i \cdot q_i(\mathbf{w}_i^z). \tag{5.16}$$

Afin d'accélérer cette opération, les sous produits vectoriels $\mathbf{x}_i \cdot \mathbf{u}_i^j$ sont calculés en avance. Le nombre total de FLOP est alors égal à :

$$kD + mn \tag{5.17}$$

Afin d'appliquer cette méthode aux réseaux convolutifs, il convient de trouver la bonne manière de découper l'ensemble de poids. Calculer la sortie d'une couche dense revient à calculer le produit matriciel entre un vecteur d'entrée $\mathbf{x} \in \mathbb{R}^{N^{in}}$ et une matrice de poids $W \in \mathbb{R}^{N^{in} \times N^{out}}$. Le découpage se fait alors le long de la première dimension de W. La figure 5.2 représente une couche dense constituée de quatre neurones d'entrée et six neurones de sortie, quantifiée à l'aide des paramètres ($c = 2$ et $k = 2$).

Pour une couche convolutive, la sortie est calculée à l'aide d'une convolution discrète entre une image $I \in \mathbb{R}^{d^{in} \times d^{in} \times N^{in}}$ et un ensemble de filtres $W \in \mathbb{R}^{d^F \times d^F \times N^{in} \times N^{out}}$ (d^{in} et d^F étant la dimension spatiale de I et des filtres respectivement). Nous découpons l'ensemble de poids W le long de N^{in}, et nous



(a) Représentation graphique. (b) Représentation matricielle.

FIGURE 5.2 – Quantification d'une couche dense constituée de $N^{in} = 4$ neurones d'entrée et $N^{out} = 6$ neurones de sortie. Les paramètres de quantification sont $(c = 2, k = 2)$.

utilisons le même dictionnaire à chaque position du filtre. La figure 5.3 représente un exemple de couche convolutive quantifiée.

5.3.2 Ré-entraînement d'un réseau quantifié

Il s'agit désormais de s'interroger sur la manière dont une couche quantifiée à l'aide du produit de quantification peut être ré-entraînée à l'aide de l'algorithme de descente de gradient. Considérons une couche dense constituée de N^{in} neurones en entrée et N^{out} neurones en sortie, quantifiée avec les paramètres (c, k). La quantification de cette couche peut-être considérée comme une modification de son architecture. L'activation des neurones de sortie en fonction de l'état $\mathbf{e}$ des neurones de la couche d'entrée peut être écrite de la manière suivante :

$$f(\mathbf{s}) = f\left(\sum_{i=1}^{m} \mathbf{e}_i U^i B^i\right), \tag{5.18}$$

avec $\mathbf{e}_i \in \mathbb{R}^c$ un sous-vecteur de $\mathbf{e}$, $U^i \in \mathbb{R}^{c \times k}$ une matrice contenant le dictionnaire $\mathcal{U}^i$ et $B^i \in \{0,1\}^{k \times N^{out}}$ une matrice d'indexation. Chaque colonne de cette dernière matrice contient un unique un.

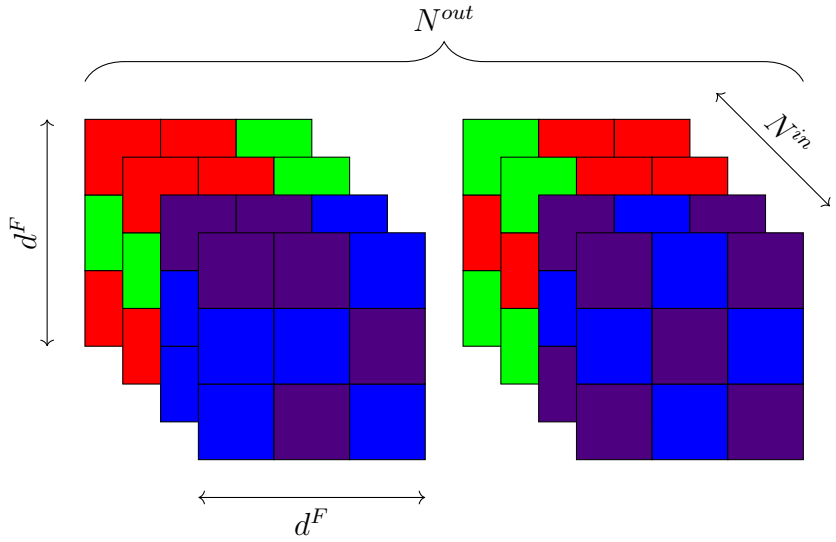


FIGURE 5.3 – Quantification d'une couche convolutive $N^{in} = 4$, $N^{out} = 2$, $d^F = 3$. Les paramètres de quantification sont $(c = 2, k = 2)$.

La figure 5.4 représente de manière graphique et matricielle la nouvelle architecture de la couche représentée sur la figure 5.2. Elle se décompose en deux sous-couches. La première de ces sous-couches est associée aux dictionnaires $\mathcal{U}^i$ tandis que la seconde l'est aux index. Des neurones intermédiaires sont notés h_{ij}, avec $i \in [1, ..., m]$ et $j \in [1, ..., k]$.

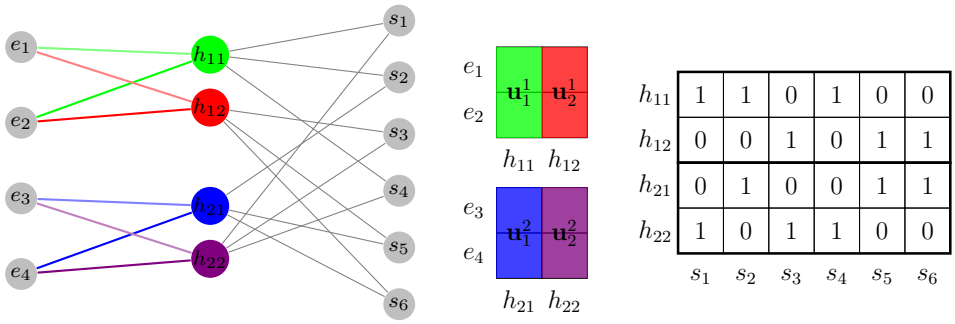
Lors de la phase de ré-entraînement, seule la sous-couche associée aux dictionnaires est mise à jour. En appliquant l'algorithme de descente de gradient, il est possible de calculer les gradients des mots de code $\mathbf{u}_i^j$. Si l'on considère $\frac{\partial \mathcal{L}}{\partial \mathbf{s}}$ l'erreur dans la couche de sortie, alors :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_i^j} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial \mathbf{u}_i^j}, \tag{5.19}$$

avec :

$$\frac{\partial \mathbf{s}}{\partial h_{ij}} = B^i_{j,:} \text{ et } \frac{\partial h_{ij}}{\partial \mathbf{u}_i^j} = \mathbf{e}_i, \tag{5.20}$$

ici $B^i_{j,:}$ est la j^{ieme} ligne de la matrice d'indexation B^i. L'équation 5.19 peut alors



	s_1	s_2	s_3	s_4	s_5	s_6
h_{11}	1	1	0	1	0	0
h_{12}	0	0	1	0	1	1
h_{21}	0	1	0	0	1	1
h_{22}	1	0	1	1	0	0

(a) Représentation graphique. (b) Représentation matricielle.

FIGURE 5.4 – Nouvelle architecture d'une couche dense constituée de $N^{in} = 4$ neurones d'entrée et $N^{out} = 6$ neurones de sortie. Les paramètres de quantification sont $(c = 2, k = 2)$.

être réécrite comme suit :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_i^j} = \mathbf{e}_i \sum_{\ell \in [1,\dots,N^{out}]}^{B_{j,\ell}^i \neq 0} \frac{\partial \mathcal{L}}{\partial s_\ell}. \tag{5.21}$$

Il est possible de quantifier la totalité des couches d'un réseau et de les ré-entraîner par la suite. Il est également possible de se concentrer sur certaines couches du réseau. Lorsque seule une partie des couches est quantifiée, les couches qui leur sont supérieures sont également ré-entraînées en appliquant une descente de gradient classique.

5.4 Expériences

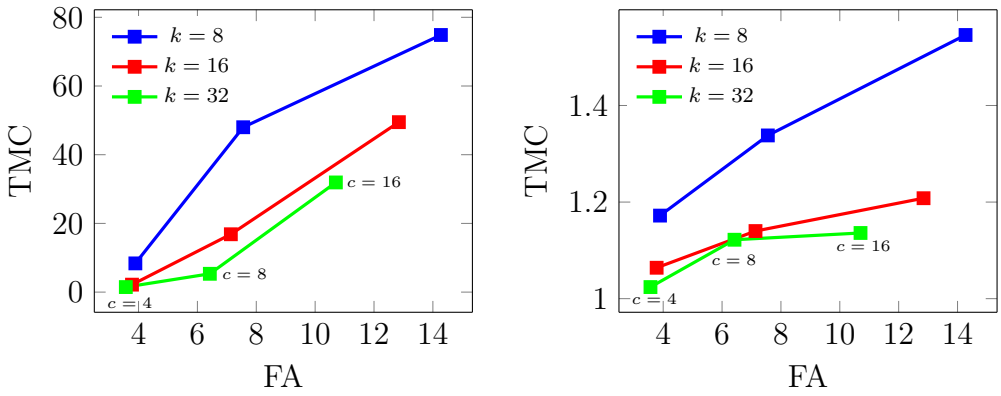
Nous présentons maintenant les différentes expériences que nous avons réalisées sur l'entraînement de réseaux convolutifs quantifiés. Dans ces expériences, nous testons notre approche sur deux jeux de données différents : MNIST et ImageNet. Pour ces deux jeux de données, nous entraînons tout d'abord un réseau de neurones, que nous quantifions ensuite à l'aide du produit de quantification pour différentes valeurs des paramètre c et k. Puis le réseau quantifié est ré-entraîné.

5.4.1 MNIST

MNIST est un jeu de données regroupant des images de chiffres manuscrits de zéro à neuf. Chaque image est de taille 28×28 pixels en niveau de gris. Le jeux de données se décompose en deux ensembles. Un ensemble d'entraînement constitué de 60000 images et un ensemble de test constitué de 10000 images. Comme il est souvent d'usage, nous utilisons les 10000 première images de l'ensemble d'entraînement comme ensemble de validation. Cet ensemble est utilisé pour sélectionner le meilleur modèle.

Pour ce jeu de données, nous n'utilisons pas un réseau convolutif mais un perceptron multicouche. Ce type de réseau est uniquement composé de couches denses. Le réseau que nous utilisons est constitué d'une couche d'entrée comportant 784 neurones (28×28), de trois couches denses de 1024 neurones chacune et d'une couche de sortie constituée de dix neurones. Pour chacune des couches nous utilisons la technique de normalisation par batch [50]. Cette technique permet d'accélérer la vitesse d'apprentissage du réseau. Pour chacune des couches nous utilisons des rectifieurs linéaires comme fonction d'activation.

Nous entraînons tout d'abord un réseau non quantifié comme modèle de référence. Cette première phase d'entraînement comporte 1000 epochs au cours desquelles le taux d'apprentissage est diminué de manière exponentiel entre 3×10^{-3} et 3×10^{-7}. Chaque batch comporte 100 images. Le taux de mauvaise classification du modèle de référence est égal à 0.97%. Puis nous quantifions ce modèle pour différentes valeurs de c et k. Les mêmes paramètres sont utilisés pour l'ensemble du réseau excepté la dernière couche qui n'est pas quantifiée. Pour chaque combinaison de paramètres, nous calculons le taux de mauvaise classification avant et après la seconde phase d'apprentissage. Les figures 5.5 et



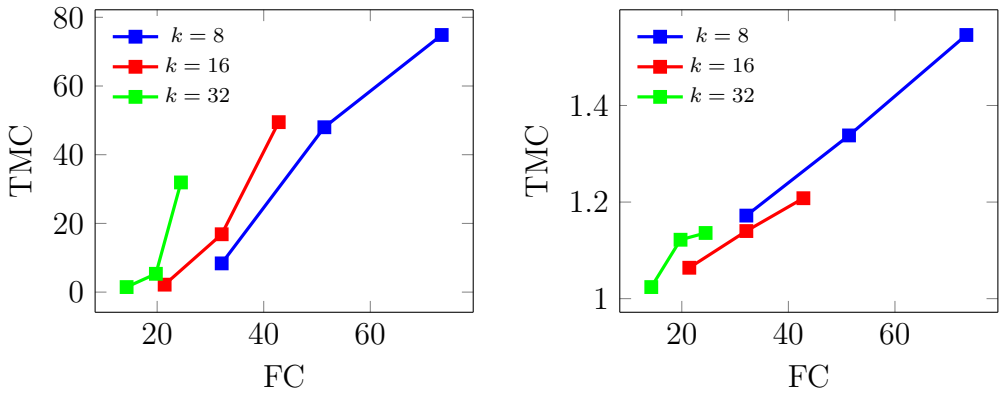
(a) Sans ré-entraînement.

(b) Avec ré-entraînement.

FIGURE 5.5 – Taux de mauvaise classification (TMC) en fonction du facteur d'accélération (FA) pour différentes valeurs de c et k.

5.6 rendent compte des taux de mauvaise classification obtenus en fonction des facteurs d'accélération et de compression. Chaque point est obtenu en prenant la moyenne de cinq simulations successives.

On peut d'abord observer que l'augmentation de la taille c des sous-espaces de quantification ainsi que la diminution de la taille k des dictionnaires conduisent à une augmentation des facteurs d'accélération et de compression mais augmentent également le taux de mauvaise classification. Cela est vrai pour un réseau ré-entraîné ou non. Cependant, le ré-entraînement d'un réseau quantifié conduit à une diminution importante du taux de mauvaise classification. Cette diminution est d'autant plus importante que les facteurs d'accélération et de compression sont élevés. En effet, un réseau quantifié à l'aide des paramètres $c = 16$ et $k = 8$ ($FA = 14.3$ et $FC = 73.4$) conduit à un taux de mauvaise classification de 74.85% sans ré-entraînement. Ce taux est ramené à 1.54% suite à la phase de ré-entraînement.



(a) Sans ré-entraînement.

(b) Avec ré-entraînement.

FIGURE 5.6 – Taux de mauvaise classification (TMC) en fonction du facteur de compression (FC) pour différentes valeurs de c et k.

5.4.2 ImageNet

Le second jeu de données ImageNet contient des images couleurs réparties entre 1000 catégories. Le jeu de données est divisé en deux ensembles, un ensemble d'entraînement constitué de plus d'un million d'images et un ensemble de validation constitué de 50000 images. Les résultats rapportés ici sont obtenus sur cet ensemble. Le réseau que nous testons est un réseau convolutif appelé AlexNet [2]. Ce réseau a fortement contribué à populariser les réseaux convolutifs en gagnant la compétition ilsvrc2012. Ce réseau est constitué de cinq couches convolutives, de deux couches denses et d'une couche de sortie. Nous entraînons une première fois le réseau en suivant la méthodologie décrite dans [2] et nous obtenons un taux de mauvaise classification de 43%.

Couches convolutives

Nous commençons par étudier l'effet de la quantification sur les couches convolutives. Les mêmes valeurs pour les paramètres c et k, sont utilisées pour l'en-

semble des couches quantifiées, excepté la première couche. En effet, le nombre de cartes de caractéristiques dépend du codage des images fournies en entrée, ici trois (pour un codage RGB). La taille des sous-espaces de quantification ne peut donc dépasser cette valeur. Le tableau 5.2 rend compte des résultats obtenus en matière de compression, d'accélération et de précision, pour différentes valeurs des paramètres c et ℓ. Ici toutes les couches convolutives sont quantifiées en même temps. Lors de la phase de ré-entraînement, les couches quantifiées ainsi que les couches qui leurs sont supérieures (les couches denses) sont mises à jour à l'aide de l'algorithme de rétropropagation du gradient. Étant donné que les couches convolutives sont les couches les plus gourmandes en ressources de calcul, nous nous attarderons plus particulièrement sur la capacité de la méthode à accélérer le réseau.

Dans un premier temps, on observe que pour chacune des configurations testées, le ré-entraînement du réseau quantifié améliore de manière significative le taux de mauvaise classification (TMC). Par exemple, lorsque $c/\ell = 8/128$ le TMC passe de 75.7% à 44.6%. Sans ré-entraînement, le réseau quantifié serait considéré comme de piètre qualité. On s'aperçoit dans un second temps que le découpage présenté dans la figure 5.3 ne permet pas de bien compresser la première couche du réseau. En effet, le facteur d'accélération de cette couche avoisine deux pour chacune des configurations testées. Par conséquent, si dans le réseau d'origine la première couche utilise 16% des ressources de calcul, elle en utilise 30 à 40% dans les réseaux quantifiés.

c/k	Original		8/128		16/128		8/64	
Couches	**Mflop**	**Moct**	**Mflop**	**Moct**	**Mflop**	**Moct**	**Mflop**	**Moct**
conv1	105	0.14	55×1.9	0.012×11.9	55×1.9	0.01×11.9	45×2.3	0.09×14.7
conv2	224	1.2	37×6.1	0.08×14.8	23×9.8	0.07×18.6	32×6.9	0.05×23.0
conv3	150	3.5	24×6.2	0.23×15.5	15×10.0	0.18×19.7	21×7.0	0.15×23.8
conv4	112	2.6	22×5.0	0.27×9.9	15×7.3	0.23×11.4	18×6.2	0.16×16.5
conv5	74	1.8	18×4.2	0.24×7.2	13×5.8	0.22×8.0	13×5.5	0.14×12.7
Total	**666**	**9.3**	**156**×4.3	**0.84**×11.2	**121**×5.5	**0.71**×13.1	**130**×5.1	**0.51**×18.2
TMC Q	43%		75.7%		94.3%		86.2%	
TMC QR			44.6%		46.2%		45.3%	

TABLE 5.2 – Quantification des couches convolutives pour différents paramètres c et k.

Couches denses

Nous étudions maintenant l'effet de la quantification sur les couches denses du réseau. Pour les deux premières couches (fc6 et fc7) nous utilisons les mêmes valeurs pour les paramètres c et k. Pour la dernière couche (fc8) nous utilisons $c = 1$ et $k = 16$. Le tableau 5.3 rend compte des résultats obtenus en terme de compression, d'accélération et de précision. Comme pour les couches convolutives, toutes les couches denses sont compressées et ré-entraînées en même temps. Nous nous intéresserons plus particulièrement à la capacité de la méthode à compresser le réseau.

En comparaison avec les résultats obtenus sur les couches convolutives, on observe que les couches denses se prêtent mieux à la quantification. Malgré des taux de compression et d'accélération plus importants, le TMC (avant et après ré-entraînement) est moins dégradé par le processus de quantification. Par exemple, pour $c/k = 16/32$, le taux de compression est de 38.2 et la dégradation du TMC est seulement de 0.8%. Comme pour les couches convolutives le ré-entraînement

c/k	Original		16/32		16/64	
Couches	**Mflop**	**Moct**	**Mflop**	**Moct**	**Mflop**	**Moct**
fc6	37.7	151	2.6×14.2	2.65×56.9	2.95×12.8	4.13×36.6
fc7	16.8	67.1	1.2×14.2	1.18×56.9	1.31×12.8	1.83×36.6
fc8	4.1	16.4	4.2×0.98	2.31×7.09	4.16×0.98	2.31×7.09
Total	**58.6**	**234**	**8.0**×7.3	**6.14**×38.2	**8.4**×7.0	**8.3**×28.3
TMC Q	43.0%		57.6%		49.9%	
TMC QR			43.8%		43.3%	

TABLE 5.3 – Quantification des deux premières couches denses (fc6, fc7) pour différentes valeurs des paramètres c et k. La dernière couche (fc8) est quantifiée avec $c = 1$ et $k = 16$.

d'un réseau quantifié améliore le TMC.

La totalité du réseau

Nous présentons maintenant des résultats lorsque la totalité des couches du réseau sont quantifiées. Le tableau 5.4 présente les résultats que nous obtenons lorsque la totalité du réseau est quantifiée pour $c = 8$ et $\ell = 128$ (excepté la première couche convolutive pour laquelle $c = 3$). Le réseau quantifié est quinze fois plus petit que le réseau d'origine et requiert 4.4 fois moins d'opérations flottantes. Sans ré-entraînement le TMC est égal à 75.4%, une fois ré-entraîné il redescend à 45.6%.

5.5 Conclusion

Au cours de la dernière décennie, les réseaux convolutifs ont permis des progrès importants dans le domaine de la vision par ordinateur et tout particulièrement en classification d'images. S'ils sont devenus l'approche privilégiée dans bon nombre

c/k	Original		8/128	
Couches	**Mflop**	**Moct**	**Mflop**	**Moct**
convolutives	666	9.3	156$\times$4.3	0.84$\times$11.2
denses	58.6	234	15.3$\times$3.8	9.5$\times$16
Total	**724**	**244**	**166**$\times$4.4	**16.2**$\times$15
TMC Q	43.0%		75.4%	
TMC QR			45.6%	

TABLE 5.4 – Quantification de la totalité du réseau.

d'applications, leur mise en œuvre sur des systèmes embarqués nécessite de réduire leur consommation en ressources de calcul et de mémoire. Les travaux présentés dans ce chapitre étudient l'utilisation d'une technique de quantification vectorielle pour à la fois compresser et accélérer un réseau convolutif déjà entraîné.

La quantification vectorielle consiste à représenter un grand ensemble de vecteurs à l'aide d'un ensemble plus petits de vecteurs de références appelé dictionnaire. Chaque vecteur de l'ensemble d'origine est alors représenté à l'aide d'un index. Appliqué à un réseau convolutif, elle consiste à modifier l'architecture de ce dernier. Une couche quantifiée peut alors être vue comme la composition de deux sous-couches : une sous-couche associée aux dictionnaires et une sous-couche associée aux index. Considérant cette nouvelle architecture, nous avons proposé de ré-entraîner à l'aide de l'algorithme de rétropropagation du gradient la sous-couche associée aux dictionnaires. Cette étape permet de réduire la dégradation de la précision engendrée par le processus de quantification.

Nous avons réalisé différentes expériences sur deux jeux de données couramment utilisés pour évaluer des algorithmes de classification d'images. Pour valider notre approche nous utilisons, la technique du produit de quantification [29] pour quantifier un réseau. Les résultats que nous avons obtenu montrent que le ré-

entraînement d'un réseau quantifié permet d'améliorer nettement la précision de celui-ci.

Afin de poursuivre les travaux débutés ici, différents axes nous semblent pertinents. Premièrement, il nous semble important d'étendre ces travaux à d'autres techniques de quantification vectorielle et notamment de quantification structurée telles que la quantification composite [51], la quantification résiduelle [52] ou la quantification additive [53]. Deuxièmement, les réseaux que nous testons dans nos expériences ne sont pas conçus pour être mis-en-œuvre sur des systèmes embarqués. Afin de concevoir des réseaux toujours moins gourmands, il paraît intéressant d'appliquer notre approche sur des réseaux dit économes tel que *MobileNets* [39] et *SqueezeNet* [40]. Ces réseaux offrent des taux de classification équivalents ou supérieurs à AlexNet tout en utilisant 46 fois moins de mémoire et en allant jusqu'à dix fois plus vite. Enfin, la méthode que nous proposons ne permet pas de ré-entraîner les sous-couches associées aux index. L'assignation des index étant faite de manière discrète, la faire à l'aide de l'algorithme de rétropropagation du gradient paraît compliqué mais serait une extension prometteuse aux travaux présentés ici.

Chapitre 6

Conclusion et perspectives

6.1 Conclusion

Cette thèse a été dédiée à l'étude de modèles connexionnistes utilisés dans le contexte de la vision par ordinateur embarquée. Nous nous sommes intéressés à deux modèles de réseaux de neurones différents, les réseaux à clusters et les réseaux convolutifs. Pour le premier, l'objectif a été d'utiliser ce modèle dans le contexte de la vision par ordinateur. Pour le second, l'objectif a été d'utiliser ce modèle pour des applications embarquées.

6.1.1 Réseaux à clusters

Un réseau à clusters est un modèle connexionniste permettant de réaliser des mémoires associatives. Ce type de réseau a connu un vif intérêt suite à sa découverte et différents projets de recherche autour de ce modèle ont vu le jour. Il a notamment été sollicité pour comprendre les mécanismes de la mémoire à long terme du cerveau biologique. On peut notamment citer le projet ERC NEUCOD [54] qui a eu pour objectif d'aboutir sur une nouvelle théorie de l'information

mentale exploitant les réseaux à clusters. D'autre part, l'utilisation de neurones et de connexions binaires couplée à un algorithme simple de décodage permet des mises en œuvre simples et efficaces de ce réseau sur des architectures matérielles. Le projet SENSE [55] a eu pour but de concevoir un système de vision par ordinateur embarquée utilisant les réseaux à clusters, de l'acquisition de l'information visuelle jusqu'à des traitements de haut niveau. Les travaux présentés dans cette thèse ont été faits dans le cadre de ce second projet.

Afin d'utiliser les réseaux à clusters dans le contexte de la vision embarquée, nous nous sommes intéressés à son utilisation pour effectuer une recherche du plus proche voisin. La recherche du plus proche voisin est une tâche récurrente en vision par ordinateur. Elle consiste à retrouver dans un ensemble de vecteurs, le vecteur le plus proche d'un certain vecteur requête. Les vecteurs considérés sont le plus souvent denses, c'est-à-dire qu'ils comportent un faible nombre de valeurs nulles. Étant extraits de données naturelles, des images, ces vecteurs sont également fortement corrélés.

Pour stocker des vecteurs denses à l'aide d'un réseau à clusters, nous proposons une méthode d'encodage basée sur la technique du produit de quantification. Chaque cluster du réseau est associé à un sous-espace de l'espace d'origine et chaque neurone de ce cluster est associé à un vecteur représentatif de ce sous-espace appelé mot de code. Pour être stocké dans le réseau, un vecteur est lui même vu comme la concaténation de plusieurs sous-vecteurs. Dans chaque cluster, le neurone associé au mot de code le plus proche du sous-vecteur associé est sélectionné pour mémoriser l'information. Afin de retrouver le vecteur le plus proche en présence d'un certain vecteur requête, nous avons proposé différentes modifications de l'algorithme de décodage d'origine. Les résultats que nous obte-

nons sont de piètre qualité lorsque les vecteurs sont générés à partir de données naturelles. De plus, les modifications que nous apportons au processus de décodage complexifient les architectures matérielles le mettant en œuvre.

Dans le but d'améliorer la capacité de restitution lorsque les vecteurs sont corrélés, nous avons proposé une extension des réseaux à clusters. Cette extension exploite l'hétéro-association pour supprimer les effets indésirables entraînés par la co-activation répétée de deux composantes. Le modèle que nous proposons, appelé réseau à clusters restreint, est un modèle à deux couches, une couche d'entrée et une couche cachée. La couche d'entrée de ce réseau est associée aux vecteurs à stocker tandis que la couche cachée est associée à des vecteurs générés aléatoirement. Les expériences que nous avons menées montrent que ce réseau est plus robuste qu'un réseau à clusters classique face à des vecteurs qui ne sont pas indépendamment distribués.

6.1.2 Réseaux convolutifs

Depuis la compétition ILSVRC2012, les réseaux convolutifs sont devenus la référence pour les applications de classification d'images et plus largement en vision par ordinateur. Ces réseaux, entraînés à l'aide de l'algorithme de rétropropagation du gradient, permettent de construire des représentations hiérarchiques complexes à partir de données brutes telles que des images. Afin de les populariser sur des systèmes embarqués, un grand nombre de recherches visent à en réduire le coût, tant en termes de ressources mémoires que de calculs.

Les travaux présentés dans cette thèse utilisent des techniques de quantification vectorielle pour accélérer et compresser des réseaux convolutifs déjà entraînés. Le processus de quantification conduit à une dégradation des perfor-

mances en termes de taux de classification, comparé au réseau d'origine. Cette dégradation est d'autant plus importante lorsque ce sont les couches convolutives qui sont quantifiées. Afin d'y remédier, nous avons montré comment l'algorithme de rétropropagation du gradient peut être modifié pour qu'il prenne en compte la nouvelle architecture du réseau. La phase de ré-entraînement du réseau permet alors de contenir la dégradation inférieure à 3% pour un réseau quinze fois plus petit et allant quatre fois plus vite que le réseau d'origine.

6.2 Limites et perspectives

Malgré les modifications que nous avons apportées aux réseaux à clusters, nous n'avons pas réussi à correctement les utiliser pour réaliser une recherche du plus proche voisin. Même s'il existe ses similitudes entre cette tâche et la fonction d'une mémoire associative, il me parait sage de considérer que l'utilisation de réseaux à clusters n'est pas pertinente dans ce contexte. De manière générale, il semble que ces réseaux ne devraient pas êtres envisagés dans les premiers niveaux d'un système de vision par ordinateur. D'une part, les vecteurs pouvant être stockés dans un réseau à clusters (binaires et structurés en sous-vecteurs) correspondent peu aux types d'information traités dans ces niveaux (haute précision et dont la structure dépend de la structure des images). D'autre part, les systèmes actuels tels que les réseaux convolutifs paraissent bien plus pertinents pour ce type de tâches. Ces réseaux sont tout d'abord performants pour construire un système hiérarchique d'extraction de caractéristiques sur une tâche particulière comme la classification d'images. De plus, cet extracteur peut ensuite être réutilisé sur une tâche complètement différente comme la recherche d'images par le contenu [56, 57].

L'utilisation des réseaux à clusters pourrait cependant être pertinente dans les derniers niveaux des systèmes de vision par ordinateurs actuels. Si les réseaux convolutifs offrent d'excellentes performances dans de nombreuses tâches, ils souffrent également de plusieurs défauts qui limitent leurs utilisations sur des systèmes embarqués. Outre le coût en ressources de ces réseaux que nous avons abordé dans cette thèse, on peut citer :

1. La taille importante de l'ensemble d'entraînement et le nombre important d'itérations nécessaire à l'algorithme de rétropropagation pour obtenir des résultats satisfaisants. Pour ces raisons l'entraînement est le plus souvent réalisé *offline*.
2. L'oubli catastrophique [58], un système entraîné sur une première tâche oublie celle-ci lorsqu'il est entraîné sur une seconde tâche.

Pour pallier à ces défauts, l'utilisation de réseaux à clusters pourrait être une solution. Le mécanisme d'apprentissage de ces réseaux est bien plus simple et rapide. De plus, si le réseau dispose du matériel suffisant (le nombre de neurones), il n'oublie pas ce qu'il a mémorisé. Afin de combiner ces deux réseaux, il semble cependant important que les données générées à l'aide d'un réseau convolutif aient la structure des données manipulées par les réseaux à clusters. Cela peut être fait en transformant ces données après coup, par exemple en utilisant des techniques similaires à celle présentée dans le chapitre 2. Une approche plus intéressante serait d'entraîner un réseau convolutif à générer directement des données pouvant être manipulées par un réseau à clusters. On peut pour cela s'inspirer du réseau SuBiC (supervised structured binary code) [59]. Ce réseau convolutif, entraîné à l'aide de l'algorithme de rétropropagation du gradient, permet d'encoder des images sous la forme de vecteur ayant la même structure que les réseaux à clusters.

Bibliographie

[1] Alan M Turing. Computing machinery and intelligence. In Parsing the Turing Test, pages 23–65. Springer, 2009.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

[4] Frank Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. Psychological review, 65(6) :386, 1958.

[5] Cominlabs. `https://www.cominlabs.u-bretagneloire.fr`.

[6] Vincent Gripon and Claude Berrou. Sparse neural networks with large learning diversity. IEEE transactions on neural networks, 22(7) :1087–1096, 2011.

[7] David J Willshaw, O Peter Buneman, and Hugh Christopher Longuet-Higgins. Non-holographic associative memory. Nature, 1969.

[8] Teuvo Kohonen. Correlation matrix memories. IEEE transactions on computers, 100(4) :353–359, 1972.

[9] Donald O Hebb. The organization of behavior, 1949.

[10] Teuvo Kohonen and Matti Ruohonen. Representation of associated data by matrix operators. IEEE Transactions on Computers, 100(7) :701–702, 1973.

[11] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8) :2554–2558, 1982.

[12] Bart Kosko. Bidirectional associative memories. IEEE Transactions on Systems, Man, and Cybernetics, 18(1) :49–60, 1988.

[13] Jim Austin and T John Stonham. Distributed associative memory for use in scene analysis. Image and Vision Computing, 5(4) :251–260, 1987.

[14] Gunther Palm, Friedhelm Schwenker, Friedrich T Sommer, and Alfred Strey. Neural associative memories. Associative processing and processors, pages 307–326, 1997.

[15] Behrooz Kamary Aliabadi, Claude Berrou, Vincent Gripon, and Xiaoran Jiang. Storing sparse messages in networks of neural cliques. IEEE Transactions on neural networks and learning systems, 25(5) :980–989, 2014.

[16] Vincent Gripon and Claude Berrou. Nearly-optimal associative memories based on distributed constant weight codes. In Information Theory and Applications Workshop (ITA), 2012, pages 269–273. IEEE, 2012.

[17] Ala Aboudib, Vincent Gripon, and Xiaoran Jiang. A study of retrieval algorithms of sparse messages in networks of neural cliques. In COGNITIVE 2014 : the 6th International Conference on Advanced Cognitive Technologies and Applications, pages 140–146, 2014.

[18] JV Kennedy and J Austin. A hardware implementation of a binary neural associative memory. In Microelectronics for Neural Networks and Fuzzy

Systems, 1994., Proceedings of the Fourth International Conference on, pages 178–185. IEEE, 1994.

[19] Hooman Jarollahi, Naoya Onizawa, Vincent Gripon, and Warren J. Gross. Architecture and implementation of an associative memory using sparse clustered networks. In Proceedings of IEEE International Symposium on Circuits and Systems, pages 2901–2904, May 2012.

[20] Philippe Coussy, Cyrille Chavet, Hugues Nono Wouafo, and Laura Conde-Canencia. Fully binary neural network model and optimized hardware architectures for associative memories. ACM Journal on Emerging Technologies in Computing Systems (JETC), 11(4) :35, 2015.

[21] Robin Danilo, Homman Jarollahi, Vincent Gripon, Philippe Coussy, Laura Conde-Canencia, and Warren J. Gross. Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks. In Proceedings of ISCAS Conference, pages 2501–2504, May 2015.

[22] Hooman Jarollahi, Naoya Onizawa, Vincent Gripon, and Warren J. Gross. Reduced-complexity binary-weight-coded associative memories. In Proceedings of International Conference on Acoustics, Speech, and Signal Processing, pages 2523–2527, May 2013.

[23] Hooman Jarollahi, Naoya Onizawa, and Warren J Gross. Selective decoding in associative memories based on sparse-clustered networks. In Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE, pages 1270–1273. IEEE, 2013.

[24] Hugues Gérald Nono Wouafo. Architectures matérielles numériques intégrées et réseaux de neurones à codage parcimonieux. PhD thesis, Université de

Bretagne Sud, 2016.

[25] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pages 3304–3311. IEEE, 2010.

[26] Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1–8. IEEE, 2008.

[27] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking : A survey. Acm computing surveys (CSUR), 38(4) :13, 2006.

[28] Josef Sivic and Andrew Zisserman. Video google : A text retrieval approach to object matching in videos. In null, page 1470. IEEE, 2003.

[29] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence, 33(1) :117–128, 2011.

[30] David G Lowe. Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2) :91–110, 2004.

[31] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pages 380–388. ACM, 2002.

[32] Robin Danilo, Hugues Nono Wouafo, Cyrille Chavet, Vincent Gripon, Laura Conde-Canencia, and Philippe Coussy. Associative memory based on clustered neural networks : improved model and architecture for oriented edge detection. In Design and Architectures for Signal and Image Processing (DASIP), 2016 Conference on, pages 51–58. IEEE, 2016.

[33] Bartosz Boguslawski, Vincent Gripon, Fabrice Seguin, and Frédéric Heitzmann. Huffman coding for storing non-uniformly distributed messages in networks of neural cliques. In proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, volume 1, pages 262–268, July 2014.

[34] Bartosz Boguslawski, Vincent Gripon, Fabrice Seguin, and Frédéric Heitzmann. Twin neurons for efficient real-world data distribution in networks of neural cliques : Applications in power management in electronic circuits. IEEE transactions on neural networks and learning systems, 27(2) :375–387, 2016.

[35] Hugues Wouafo, Cyrille Chavet, and Philippe Coussy. Improving storage of patterns in recurrent neural networks : Clone-based model and architecture. In Circuits and Systems (ISCAS), 2015 IEEE International Symposium on, pages 577–580. IEEE, 2015.

[36] Robin Danilo, Philippe Coussy, Laura Conde-Canencia, Vincent Gripon, and Warren J Gross. Restricted clustered neural network for storing real data. In Proceedings of the 25th edition on Great Lakes Symposium on VLSI, pages 205–210. ACM, 2015.

[37] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. IEEE Intelligent Systems and their applications, 13(4) :18–28, 1998.

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3) :211–252, 2015.

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

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

[41] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect : Training deep neural networks with binary weights during propagations. In Advances in neural information processing systems, pages 3123–3131, 2015.

[42] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In Advances in neural information processing systems, pages 4107–4115, 2016.

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

[44] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1984–1992, 2015.

[45] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. IEEE

transactions on pattern analysis and machine intelligence, 38(10) :1943–1955, 2016.

[46] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, pages 2148–2156, 2013.

[47] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, pages 1269–1277, 2014.

[48] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv :1412.6115, 2014.

[49] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4820–4828, 2016.

[50] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv :1502.03167, 2015.

[51] Ting Zhang, Chao Du, and Jingdong Wang. Composite quantization for approximate nearest neighbor search. In ICML, number 2, pages 838–846, 2014.

[52] Yongjian Chen, Tao Guan, and Cheng Wang. Approximate nearest neighbor search by residual vector quantization. Sensors, 10(12) :11259–11273, 2010.

[53] Artem Babenko and Victor Lempitsky. Additive quantization for extreme vector compression. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[54] Neucod. http://recherche.imt-atlantique.fr/neucod/.

[55] Sense. https://reliasic.cominlabs.u-bretagneloire.fr/web/sense.

[56] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In European conference on computer vision, pages 584–599. Springer, 2014.

[57] Artem Babenko and Victor Lempitsky. Aggregating local deep features for image retrieval. In Proceedings of the IEEE international conference on computer vision, pages 1269–1277, 2015.

[58] John R Riesenberg. Catastrophic Forgetting in Neural Networks. PhD thesis, University of Cincinnati, 2000.

[59] Himalaya Jain, Joaquin Zepeda, Patrick Pérez, and Rémi Gribonval. Subic : A supervised, structured binary code for image search. In Proc. Int. Conf. Computer Vision, volume 1, page 3, 2017.