UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
ÉTIENNE BEAULAC

RECONSTRUCTION D'OBJETS 2D FRAGMENTÉS À L'AIDE DE RÉSEAUX DE NEURONES CONVOLUTIFS SIAMOIS ÉQUIVARIANTS AUX ROTATIONS ET DE MATRICES D'ADJACENCE DE CONTOURS

MAI 2023

CE MÉMOIRE A ÉTÉ ÉVALUÉ
PAR UN JURY COMPOSÉ DE :

M. Fadel Touré, directeur de recherche
Département de mathématiques et d'informatique
Université du Québec à Trois-Rivières

M. François Meunier, membre du jury
Département de mathématiques et d'informatique
Université du Québec à Trois-Rivières

M. Alexandre Blondin Massé, membre du jury
Département d'informatique
Université du Québec à Montréal

Remerciements

Mes premiers remerciements vont à mes directeurs de recherche, les professeurs Fadel Touré et Alain Goupil, qui m'ont fait confiance en me proposant ce projet de recherche. Merci pour vos conseils, votre disponibilité, nos nombreuses discussions fort intéressantes ainsi que pour l'opportunité d'avoir voyagé en Turquie.

J'en profite d'ailleurs pour remercier toute l'équipe du projet archéologique Kerkenes, en particulier le professeur Scott Branting et Dominique Langis-Barsetti, pour leur chaleureux accueil et leur grande générosité.

Merci aux enseignant·e·s que j'ai côtoyés et aux étudiant·e·s à qui j'ai eu la chance d'enseigner. Vous continuez à forger la personne que je suis aujourd'hui et je vous en serai pour toujours très reconnaissant.

Je souhaite aussi remercier tous mes proches et amis, et tout spécialement ma mère Jacynthe et Émanuel, pour votre écoute, votre patience et votre support moral. Sans vous, rien de tout cela n'aurait été possible.

Finalement, je tiens à remercier le Conseil de recherches en sciences naturelles et en génie du Canada, le Fonds de recherche du Québec – Nature et technologie et la Fondation de l'Université du Québec à Trois-Rivières pour leur soutien financier tout au long de mes études.

Résumé

Lors de leurs recherches, les archéologues découvrent de nombreux artéfacts, souvent cassés en plusieurs morceaux. Au terme de leurs fouilles, ils se retrouvent donc, en quelque sorte, avec plusieurs casse-têtes dont les pièces ont été mélangées ensemble.

La littérature présente plusieurs approches qui reposent sur les réseaux de neurones profonds pour reconstruire des objets à partir de leurs fragments. Or, celles-ci restreignent soit la forme des fragments, soit leur orientation, soit leur nombre. Cela ne correspond pas à la réalité de plusieurs domaines d'application, notamment l'archéologie.

Pour remédier à ces limitations, ce mémoire propose une nouvelle méthodologie basée sur les réseaux de neurones convolutifs siamois et équivariants aux rotations. Celle-ci est en mesure de prédire l'adjacence de paires de fragments de forme et d'orientation quelconques. Nous proposons également un nouvel algorithme pour générer en grand nombre des fragments à utiliser en guise de données d'entraînement, un aspect important pour la création de modèles performants.

Étienne Beaulac

Fadel Touré et Alain Goupil

Abstract

During their research, archaeologists discover many artifacts, often broken into several pieces. At the end of their excavations, they find themselves with several puzzles whose pieces have been mixed together.

The literature presents several approaches using deep neural networks to reconstruct objects from their fragments. However, these approaches restrict either the shape of the fragments, their orientation or their number. This does not fit the reality of several application domains, notably archaeology.

To remedy these limitations, this memoir presents a new methodology based on rotation invariant convolutional Siamese neural networks. This architecture is able to predict the adjacency of pairs of fragments of arbitrary shape and orientation. We also introduce an algorithm that is able to generate a large number of fragments to be used as training data, an important aspect for the creation of efficient models.

Table des matières

Liste des figures

Chapitre 1

Introduction

1.1 Problématique

La reconstruction d'objets fragmentés, comme son nom l'indique, est le processus de reconstruction d'un ou de plusieurs objets à partir de leurs fragments. Certains de ces fragments peuvent être manquants ou abîmés. Il s'agit d'un problème où les méthodes de résolution diffèrent selon la nature des fragments : images bidimensionnelles (2D), surfaces tridimensionnelles (3D) ou volumes tridimensionnels.

Cette problématique peut être divisée en trois sous-problèmes [3, 9] :

1. comparer les fragments, deux par deux, afin d'identifier des paires potentiellement adjacentes ;
2. reconstruire globalement l'objet original ;
3. réparer les défauts causés par les fragments manquants ou abîmés.

Les approches présentées dans l'état de l'art au chapitre 2 s'inscrivent dans une ou plusieurs de ces sous-problématiques.

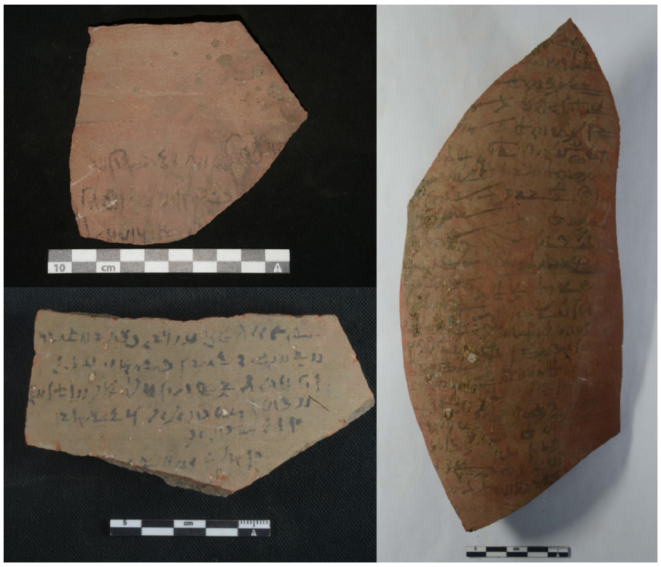


FIGURE 1.1 – Fragments d'ostraca égyptiens (tessons de poterie comportant des inscriptions), tiré de [1].

Ce sujet de recherche possède notamment des applications en archéologie. Lors de leurs recherches, les archéologues découvrent de nombreux artéfacts, souvent brisés en plusieurs morceaux (figures 1.1 et 1.3) et érodés par le temps. Au terme de leurs fouilles, ils se retrouvent donc, en quelque sorte, avec plusieurs casse-têtes dont les pièces ont été mélangées. C'est pourquoi le processus de réassemblage est long et ardu. De plus, la manipulation de fragments entraîne nécessairement certains bris : la reconstruction manuelle est donc considérée comme un type d'analyse destructive [10]. L'automatisation de ce processus permettrait, à terme, de mieux conserver les tessons en diminuant le nombre de manipulations nécessaires. Des applications existent aussi en sciences criminalistiques, par exemple pour la reconstruction de crânes humains (figure 1.2) ou de documents déchiquetés (figure 1.4).

Tel qu'abordé dans l'état de l'art au chapitre suivant, plusieurs nouvelles approches de reconstruction 2D utilisent l'apprentissage profond. Or, celles-ci restreignent le nombre, la forme ou l'orientation des fragments. Cela limite leur applicabilité à des cas réels et complique leur potentielle généralisation à des données 3D.

FIGURE 1.2 – Résultat d'une reconstruction automatisée d'un crâne humain à partir de fragments tridimensionnels, tiré de [2]. Les contours des différents fragments sont tracés en rouge.

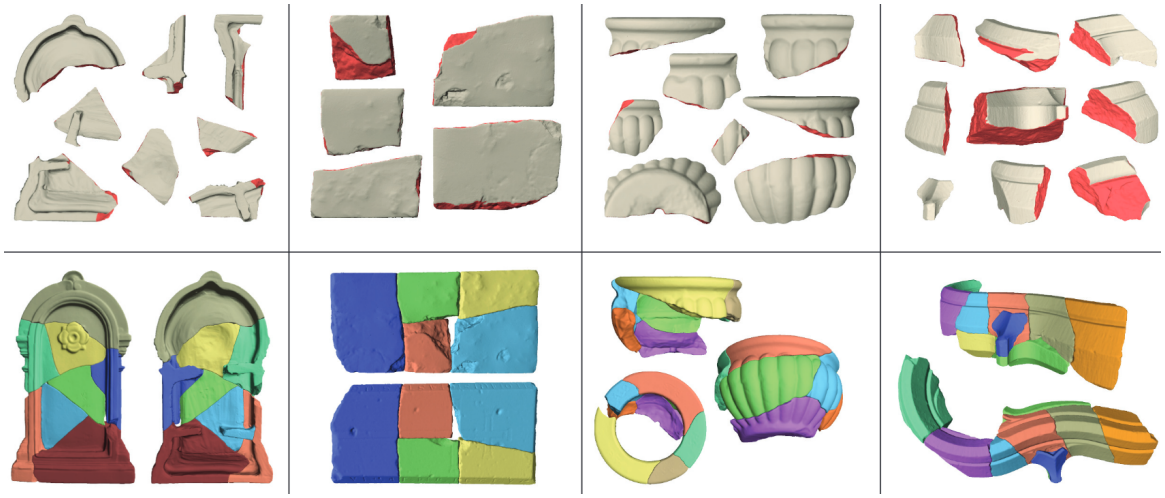


FIGURE 1.3 – Reconstruction de fragments archéologiques tridimensionnels, tiré de [3]. La rangée du haut montre les fragments originaux, tandis que la rangée du bas montre les reconstructions proposées par le système.

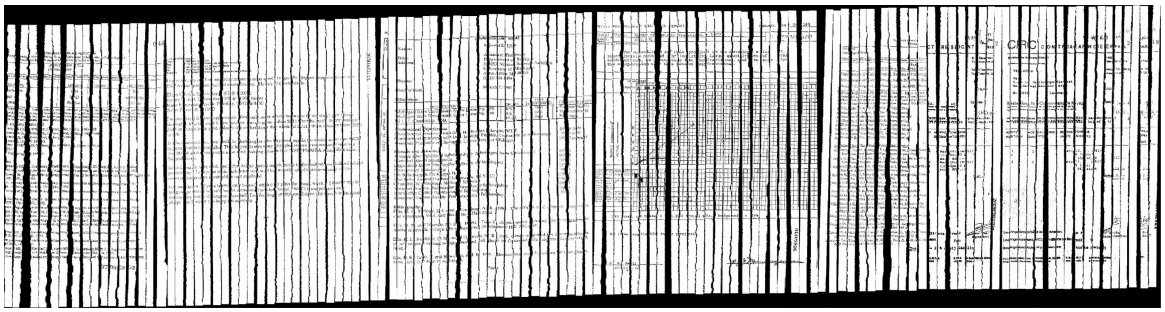


FIGURE 1.4 – Fragments réassemblés de cinq documents déchiquetés, tiré de [4].

1.2 Objectifs

Ce mémoire s'intéresse au premier des trois sous-problèmes présentés précédemment, soit l'identification de paires de fragments adjacents.

L'objectif principal consiste à développer une approche basée sur l'apprentissage profond permettant de prédire l'adjacence de paires de fragments, et ce, avec le moins de contraintes possible au niveau du nombre, de la taille, de la forme et de l'orientation des fragments. Cela a pour but d'augmenter l'applicabilité de l'approche proposée dans des situations réelles. Par exemple, on peut penser à la reconstruction de poteries en archéologie ou de documents en science forensique, où les fragments sont de forme et d'orientation quelconques.

L'objectif secondaire vise à concevoir un algorithme de génération de fragments artificiels. L'apprentissage profond nécessite un grand nombre de données d'entraînement et il n'existe pas, dans le domaine public, de jeu de données répondant aux besoins de ces travaux. Il devient donc nécessaire de créer des données de toutes pièces en grand nombre.

1.3 Contributions nouvelles

Ce mémoire présente deux nouveaux algorithmes qui, ensemble, permettent de fragmenter artificiellement des images rectangulaires. Comme l'apprentissage profond nécessite un grand nombre de données, il aurait été impossible de réaliser les entraînements présentés dans les chapitres subséquents sans ces algorithmes. Nous avons jugé qu'il était essentiel d'expliciter ces étapes étant donné leur importance dans la résolution de notre problématique.

Nous utilisons également les réseaux équivariants aux rotations, ce qui, à notre connaissance, n'avait jamais été tenté dans un contexte de reconstruction d'objets brisés. C'est grâce à eux que nous avons développé notre nouvelle approche de classification de matrices d'adjacence, présentée au chapitre 4, qui permet d'évaluer l'adjacence de fragments de forme quelconque.

1.4 Organisation du mémoire

Le chapitre 2 présente l'état de l'art de la reconstruction d'objets fragmentés et, plus particulièrement, celui de l'identification de paires de fragments adjacents. Nous y cernons les raisons motivant l'objectif de ce mémoire. Le chapitre 3 introduit les notions préalables à propos de l'apprentissage machine et des réseaux de neurones. Ces concepts sont utilisés au chapitre 4 ; celui-ci présente la méthodologie proposée, incluant des algorithmes pour générer artificiellement des fragments d'images et deux types d'architectures neuronales. Finalement, le chapitre 5 analyse les résultats obtenus à la suite des expérimentations.

Chapitre 2

État de l'art

Comme les travaux de ce mémoire se consacrent à l'identification de paires de fragments adjacents, nous présentons ci-après plusieurs approches qui s'inscrivent dans cette problématique. Elles sont classées en deux grandes catégories : les approches algorithmiques et les approches neuronales.

2.1 Approches algorithmiques

Avant la démocratisation de l'apprentissage profond, la reconstruction d'objets fragmentés nécessitait la conception d'algorithmes déterministes. Comme ces algorithmes ne peuvent pas apprendre d'un grand nombre de données comme les réseaux de neurones, il était nécessaire d'ajuster manuellement ces algorithmes pour les spécialiser à une situation particulière. Plus bas sont présentés des algorithmes de reconstruction 3D et 2D. Ceux-ci comportent plusieurs limitations ayant servi à l'élaboration des objectifs de ce mémoire : l'ajustement manuel des paramètres, un nombre restreint de données de test, et les couleurs de fragments adjacents doivent être similaires.

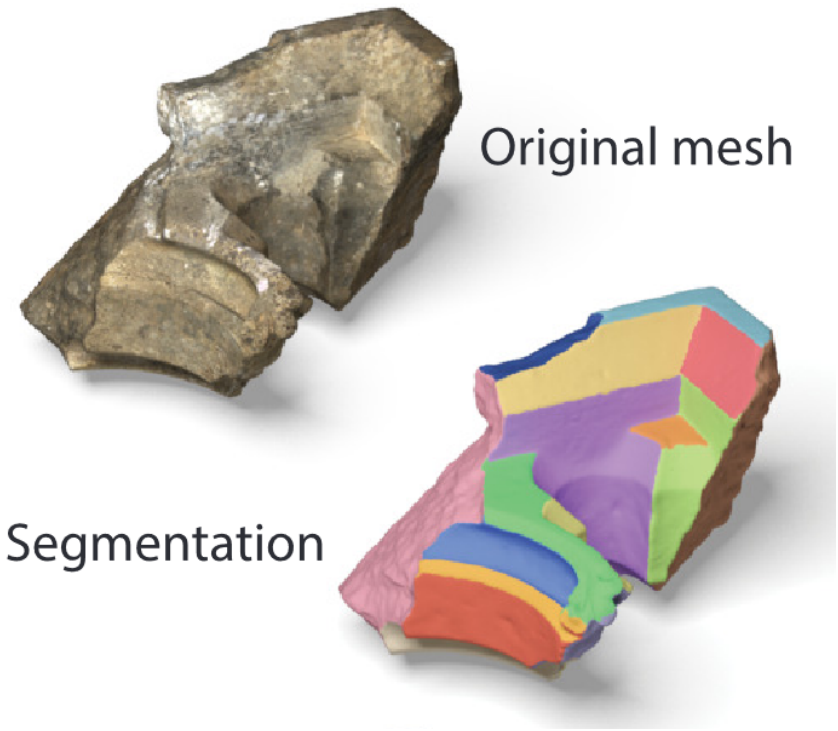


FIGURE 2.1 – La segmentation des surfaces d'un fragment, tiré de [3].

Papaioannou et coll. [3] proposent un algorithme permettant d'identifier des fragments adjacents à partir de leurs modèles 3D. La surface de chaque fragment est divisée en sous-surfaces par un algorithme de segmentation afin d'en extraire les différentes facettes (figure 2.1). Les facettes des fragments sont ensuite comparées entre elles avec, entre autres, les méthodes RANSAC (*random sample consensus*) [11], recuit simulé (*simulated annealing*) [12] et ICP (*iterative closest point*) [13]. L'objectif est de trouver une transformation qui minimise la distance entre les facettes. Certains seuils doivent être ajustés manuellement. La précision de l'algorithme de prédiction d'adjacence n'est pas présentée [14]. Sa performance a toutefois été évaluée en partie manuellement sur un jeu de 77 fragments, dont 64 proviennent d'un site archéologique. Les auteurs notent que l'algorithme performe bien sur des objets brisés sans érosion, mais le pouvoir de discrimination diminue pour des fragments érodés. Par exemple, sur un sous-ensemble de 17 fragments appartenant à 8 objets distincts, seulement 4 de ces objets ont été correctement réassemblés.

Justino et coll. [10] présentent un algorithme pour reconstruire des feuilles de papier déchirées à la main. Les fragments sont 2D et seule leur forme est prise en compte, c'est-à-dire que ce qui est dessiné, écrit ou imprimé sur les morceaux est ignoré. Un fragment est donc un contour polygonal, soit une suite de sommets et de segments. Dans un premier temps, l'algorithme de Ramer-Douglas-Peucker (RDP) [15, 16] simplifie les contours en supprimant les sommets qu'il juge superflus. Chaque sommet restant est ensuite caractérisé par un vecteur de 5 nombres :

— l'angle formé par les deux segments incidents au sommet ;
— la longueur du premier segment incident ;
— la longueur du second segment incident ;
— la coordonnée en x du sommet ;
— la coordonnée en y du sommet.

L'algorithme juge que deux sommets de deux fragments différents sont adjacents si la somme de leurs angles vaut environ 360° et que les longueurs de leurs segments incidents homologues sont semblables. Les seuils sont choisis à la suite d'un apprentissage sur des données d'entraînement. L'algorithme a été testé sur 100 documents déchirés à la main en ensembles comportant entre 3 et 16 fragments. Pour les feuilles séparées en 5 morceaux et moins, la précision de reconstruction totale du document est supérieure à 95 %. Toutefois, elle baisse lorsque le nombre de fragments augmente : elle diminue à environ 75 % pour 10 fragments, puis à environ 50 % pour 15 fragments.

L'article de Zhang et Li [17] s'attaque au même problème que le précédent : reconstruire un document papier 2D déchiré à la main. Or, ici, on s'intéresse plutôt aux photos, où la couleur des fragments est une information importante à considérer. C'est pourquoi les auteurs utilisent un algorithme de RDP adapté. Comme dans l'algorithme RDP original, les sommets superflus sont éliminés. De plus, on souhaite que chaque segment soit de couleur approximativement uniforme. Pour ce faire, des sommets sont ajoutés lorsque la variation de couleur le long d'un segment est supérieure à un seuil. Deux fragments sont adjacents s'ils partagent un ensemble de segments de longueur et de couleur similaires. Un score est ensuite calculé en fonction de la longueur de

la frontière commune, du nombre de segments partagés et de la ressemblance entre les couleurs des segments jugés adjacents. Les seuils et la formule de calcul du score ont été ajustés empiriquement. Une fois qu'une paire est considérée comme adjacente, l'algorithme ICP s'occupe de raffiner la position relative des frontières communes. Pour valider leurs travaux, les auteurs utilisent quatre photos rectangulaires déchirées à la main. L'erreur de reconstruction moyenne varie entre 0,10 et 0,26. Les auteurs définissent l'erreur de reconstruction comme la moyenne des distances entre chaque pixel du contour d'un fragment et son pixel correspondant sur le contour du fragment adjacent.

La méthode développée par Derech et coll. [18] se différencie des précédentes par l'extrapolation des fragments. Les auteurs notent que dans certaines situations, la géométrie des fragments 2D peut ne pas être exacte, par exemple, à cause de l'érosion en archéologie. C'est pourquoi l'algorithme proposé débute par l'étape d'extrapolation, soit la génération d'une bande de k pixels autour de la frontière de chaque fragment. Les fragments extrapolés sont ensuite comparés aux autres pour trouver des intersections où les différences de couleur et de gradient sont faibles. Ces différences sont calculées à l'intérieur d'un score de dissimilitude normalisé par rapport à la longueur de la frontière partagée entre les deux fragments comparés. L'algorithme a été évalué sur 42 images de fresques, certaines déjà brisées, d'autres fracturées par ordinateur. Chacune comporte, au plus, 40 fragments. La précision de la prédiction de l'adjacence (pourcentage du nombre de paires adjacentes valides par rapport à toutes les paires adjacentes prédites) est supérieure à 94 %. Cependant, les pièces appartenant à des fresques différentes n'ont pas été mélangées entre elles pendant la validation. Cela facilite l'évaluation en réduisant les possibilités de faux positifs. De plus, les auteurs notent qu'il faut 200 minutes pour reconstruire un artéfact de 36 morceaux, ce qui est un temps relativement élevé.

En 2019 s'est déroulée la compétition DAFNE (*Digital Anastylosis of Frescoes challeNgE*). Ce concours consistait à développer une méthode spécialisée pour l'anas-

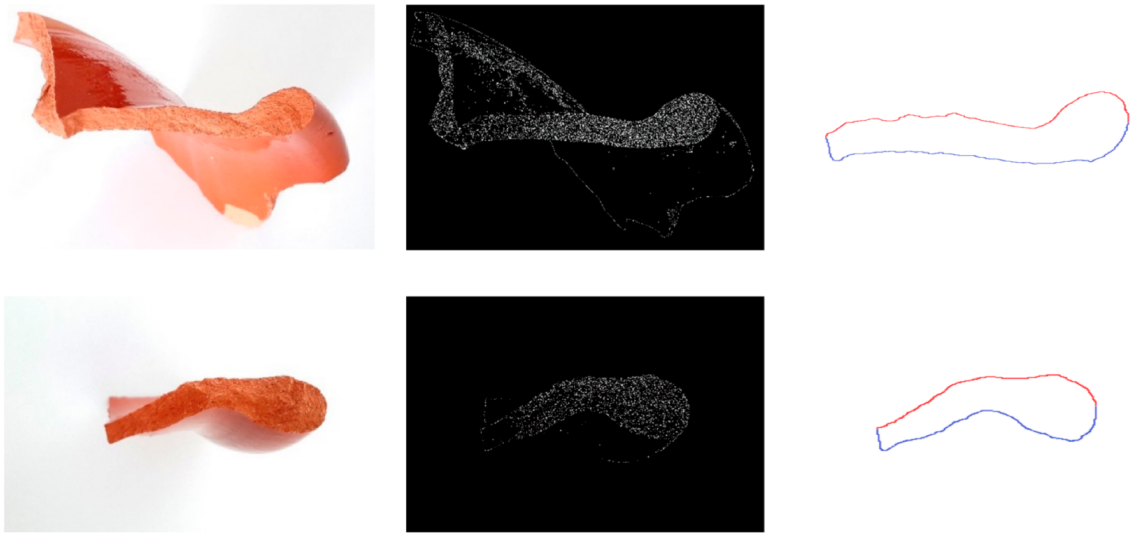


FIGURE 2.2 – Exemples d'extraction des profils de fracture à partir de photos prises de côté dans [5].

tylose de fresques [1]. Pour cette occasion, Dondi et coll. [19] ont rendu public un jeu de données composé de fragments 2D érodés et générés artificiellement à partir de 65 photos de fresques connues. Les participants ont utilisé plusieurs approches, dont les descripteurs SIFT [20], la corrélation croisée normalisée invariante aux rotations [21] et les descripteurs FAST et BRISK [22]. Ce dernier a gagné la compétition et l'exactitude des positions prédites varie, selon la fresque, entre 71 % et 100 % avec une moyenne de 91 %.

Dans un autre ordre d'idées, au lieu de se servir d'images de poteries vues du dessus, Eslami et coll. [5] ont plutôt recours à des photos prises de côté (figure 2.2). Les profils de fracture des tessons sont extraits avec le filtre de Canny [23], puis comparés avec la méthode des transformées en ondelettes [24, 25]. Une précision de 95,3 % a été mesurée à la suite d'une validation effectuée sur un pot moderne cassé en 19 tessons.

Les approches algorithmiques présentées dans cette section comportent des limi-

1. L'anastylose est un terme archéologique désignant la reconstruction de monuments, d'œuvres ou d'objets endommagés à partir de leurs fragments d'origine.

tations communes. Premièrement, la plupart utilisent peu de données de validation. Deuxièmement, tous les algorithmes nécessitent des seuils : certains pour identifier des ressemblances chromatiques, d'autres, pour des ressemblances géométriques. Dans les deux cas, l'utilisation de seuils fixes n'est pas souhaitable dans notre contexte de reconstruction archéologique de poteries : la couleur et la forme des tessons ont probablement été altérées par les intempéries. L'ajustement manuel de seuils est une procédure qui se généralise difficilement. Tel que résumé dans [5], « One of the major problems in all the methods proposed in the literature to recognize edges matching concerns the identification of threshold values. » C'est pour toutes ces raisons que, dans ce mémoire, nous nous intéressons aux méthodes issues de l'apprentissage profond, où l'ajustement de seuils est substitué à la collecte d'un grand nombre de données qui permet aux algorithmes d'apprendre ces paramètres.

2.2 Réseaux de neurones convolutifs et architectures siamoises

Les réseaux de neurones convolutifs (*convolutional neural networks*, ou CNN) [26] font partie de la famille de l'apprentissage profond. Il y a une dizaine d'années, les CNN ont révolutionné la vision par ordinateur. Grâce à leur grande capacité de généralisation, ils constituent actuellement la meilleure approche pour presque toute tâche de reconnaissance d'images [27]. Dans ce mémoire, l'approche que nous proposons utilise un CNN classificateur binaire et un réseau siamois équivariant aux rotations. Ces types de réseaux sont présentés ci-après.

2.2.1 Classification binaire

En 2019, Le et Li introduisent le réseau *JigsawNet* [9], une méthode hybride utilisant, dans un premier temps, l'algorithme de Zhang et Li présenté précédemment [17] pour générer des propositions de frontières adjacentes entre des paires de fragments. Un CNN évalue ensuite chaque proposition. Pour ce faire, on juxtapose les deux fragments comme la proposition le suggère, puis on extrait une image de leur frontière partagée. Le CNN utilise cette image pour déterminer s'il s'agit d'une bonne reconstruction : le réseau agit alors comme classificateur binaire. Celui-ci obtient une précision (définition 3.8) de 78 % à 85 % et un rappel (définition 3.9) de plus de 98 %. L'ensemble de validation est composé de plusieurs centaines de milliers de paires issues de plus de 100 photos artificiellement fragmentées en 36 ou 100 morceaux.

Paixão et coll. [4] utilisent une méthodologie similaire pour réassembler des documents papier déchiquetés. Les fragments sont tous de forme rectangulaire et étroite (figure 1.4). Pour vérifier si deux bandelettes sont adjacentes, elles sont mises côte à côte, puis des fenêtres de 32 par 32 pixels sont prises sur leur frontière commune. Ensuite, un CNN classe ces fenêtres en produisant un score entre 0 et 1 (valide si supérieur à 0,5, invalide sinon). Pour obtenir le score d'adjacence de deux bandelettes, on calcule la moyenne des scores de leurs fenêtres. Au terme de la validation sur près de 200 documents déchiquetés, le modèle atteint une précision de plus de 92 %.

Ces deux approches seraient difficilement applicables à la reconstruction de poterie. *JigsawNet* débute par un algorithme qui se base sur des ressemblances précises au niveau de la géométrie et de la couleur des fragments. C'est un a priori valable pour des fragments intacts, par exemple les photos déchirées qui sont utilisées dans l'article, mais pas pour des objets abîmés par les intempéries pendant des centaines, voire des milliers d'années. Quant à [4], l'utilisation d'une fenêtre mobile permet de s'adapter à toute longueur du document, mais cela requiert des fragments filiformes : les fragments de forme quelconque ne sont pas pris en charge.

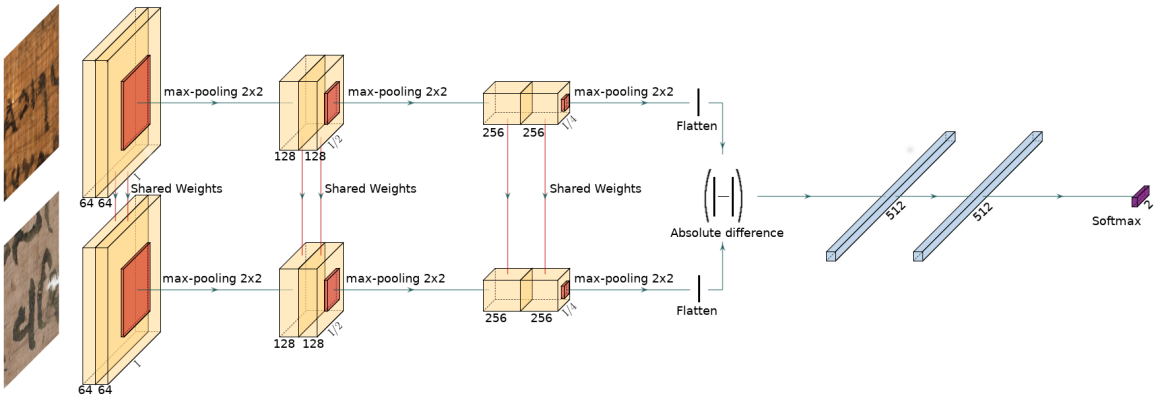


FIGURE 2.3 – Architecture siamoise proposée par [6]. Le réseau reçoit deux images à comparer. Les deux branches du réseau sont complètement identiques. Leurs sorties respectives sont jointes, puis entrent dans d'autres couches neuronales.

2.2.2 Architectures siamoises

Les architectures neuronales siamoises ont été introduites en 1993 par Bromley et coll. [28] pour valider des signatures. Pour recevoir deux images, un réseau siamois débute par deux branches jumelles, d'où son nom (figure 2.3). C'est pourquoi ce type de réseau peut comparer deux entrées du même type et prédire une information par rapport à la paire reçue.

Les auteurs présentés ci-après dans cette sous-section (Doersch et coll. [29], Paumard et coll. [7, 30], Noroozi et Favaro [8], Pirrone et coll. [6] et Ostertag et Beurton-Aimar [1, 31]) ont étudié la prédiction de la position relative de sous-images carrées. Cela est en quelque sorte semblable à la résolution d'un puzzle de type *jeu de taquin*. Or, le fait de limiter la forme des fragments est un des problèmes auquel souhaite remédier ce mémoire. Pour obtenir des fragments de forme carrée, la géométrie de leurs contours est ignorée. Une attention particulière est donc portée au contenu central des fragments. Cependant, comme mentionné précédemment, les couleurs ne constituent pas une information fiable pour plusieurs champs d'application, dont l'archéologie. Malgré tout, les réseaux siamois sont des architectures pertinentes pour les travaux de ce mémoire, dont l'objectif est de prédire l'adjacence de paires de frag-

ments. D'ailleurs, notre méthodologie présentée dans les chapitres suivants utilise, entre autres, un réseau siamois.

Doersch et coll. [29] utilisent un CNN siamois qui reçoit deux sous-images carrées, soit le fragment central de référence et celui dont la position relative doit être prédite. Dans une grille 3×3, il y a 8 positions adjacentes au carré central : en haut à gauche, en haut au centre, ..., en bas à droite. Ces 8 positions correspondent aux 8 classes de sortie du réseau. La jonction des deux branches du réseau siamois s'effectue par concaténation. À partir des jeux de données Pascal VOC 2007 et ImageNet [32], plus de 250 000 paires de carrés ont été extraites aléatoirement pour tester le modèle, qui a obtenu une exactitude avoisinant les 40 %. Dans le contexte de ce mémoire, cette approche possède une applicabilité limitée : elle suppose qu'un fragment possède 8 fragments adjacents, ce qui n'est pas toujours le cas dans la réalité.

Dans [7, 30], l'objectif pratique de Paumard et coll. est semblable à Doersch et coll. [29] : prédire la position relative entre un fragment central et un fragment voisin dans une grille 3×3. L'architecture du CNN siamois est toutefois plus moderne et la jonction des branches siamoises utilise le produit de Kronecker matriciel [2]. L'usage de ce dernier augmente la précision au prix d'un nombre beaucoup plus important de paramètres à entraîner. De plus, les auteurs ajoutent une neuvième classe en sortie du réseau représentant la non-adjacence. Pour simuler l'érosion, les positions d'où sont extraits les carrés dans l'image d'origine sont légèrement modifiées par des translations aléatoires (figure 2.4). Cela force la présence d'un espace entre les fragments. Cette méthodologie a été testée, entre autres, sur ImageNet [32], tel que fait dans [29] présenté au paragraphe précédent. L'exactitude obtenue atteint 65 %, et 57 % pour la version sans produit de Kronecker. Notons cependant que le réseau avec produit de Kronecker contient un nombre supérieur de paramètres à entraîner, donc la comparaison n'est pas tout à fait rigoureuse.

2. Cette opération entre deux matrices, dénotée par $\otimes$, s'effectue en insérant la seconde matrice devant chaque élément de la première, puis en effectuant la multiplication par un scalaire. Par exemple, $\begin{bmatrix}2 & 3\end{bmatrix} \otimes \begin{bmatrix}1 & 5 & 4\end{bmatrix} = \begin{bmatrix}2\begin{bmatrix}1 & 5 & 4\end{bmatrix} & 3\begin{bmatrix}1 & 5 & 4\end{bmatrix}\end{bmatrix} = \begin{bmatrix}2 & 10 & 8 & 3 & 15 & 12\end{bmatrix}$.

FIGURE 2.4 – Exemple de grille 3×3 érodée utilisée par [7].

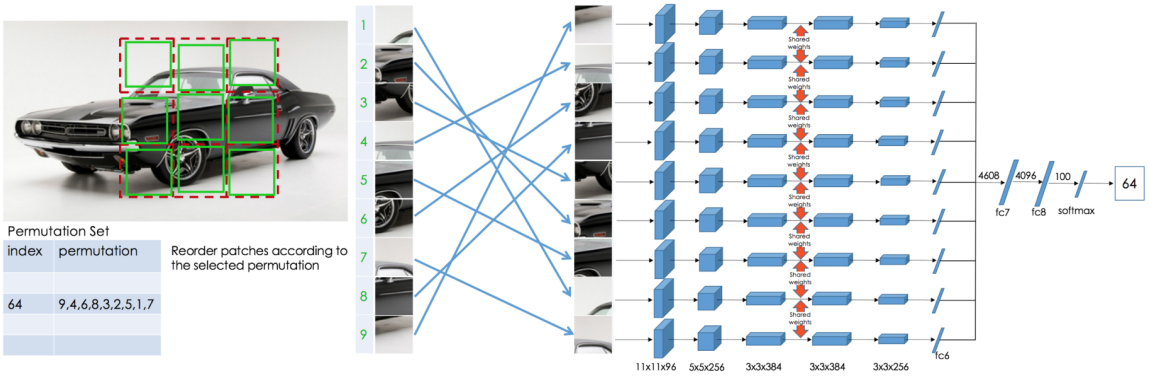


FIGURE 2.5 – Le réseau siamois ennéade de Noroozi et Favaro [8]. L'architecture reçoit en même temps 9 fragments provenant d'une grille 3×3 et doit prédire la permutation dans laquelle les fragments ont été donnés aux 9 branches.

Noroozi et Favaro [8] utilisent un CNN siamois ennéade[3] (figure 2.5). Il reçoit en entrée 9 pièces issues d'une grille 3×3 érodée aléatoirement et tente de prédire la permutation dans laquelle se trouvent les morceaux reçus. Il n'y a donc pas de comparaison deux par deux. De plus, on suppose que les 9 pièces sont adjacentes en formant une grille 3×3. Les 9 branches du réseau sont jointes par concaténation. Pour constituer les classes en sortie du réseau, un sous-ensemble de toutes les permutations possibles des 9 pièces de la grille est sélectionné au hasard. La performance du modèle varie grandement selon le nombre de permutations (le nombre de classes de sortie) : plus il y a de possibilités, plus il est difficile de faire le bon choix. L'exactitude est de 99 % pour 6 permutations possibles, mais diminue à 88 % pour 100 permutations, puis jusqu'à 54 % pour 1000 permutations.

Pirrone et coll. [6] ont développé un réseau CNN siamois pour déterminer si deux images carrées de papyrus font partie du même document. Le type de fusion utilisé est la différence absolue élément par élément[4], car ils jugeaient que la concaténation produisait des résultats asymétriques : la prédiction produite par le réseau en recevant le fragment 1 suivi du fragment 2 n'est pas toujours la même que lorsque le réseau reçoit le fragment 2 suivi du fragment 1. Validé sur plus de 500 fragments de papyrus

3. À 9 branches.

4. La différence absolue élément par élément de deux vecteurs est une opération symétrique : par exemple, pour $\begin{bmatrix}4 & 3\end{bmatrix}$ et $\begin{bmatrix}2 & 7\end{bmatrix}$, le résultat est $\begin{bmatrix}|4-2| & |3-7|\end{bmatrix} = \begin{bmatrix}2 & 4\end{bmatrix}$.

variés, le modèle atteint une exactitude de 96 % et un rappel de 82 %.

Ostertag et Beurton-Aimar [1, 31] utilisent une architecture convolutive siamoise pour déterminer la position relative entre des fragments d'ostraca égyptiens. Les images étant carrées, des caractéristiques sont directement extraites des quatre côtés des photos, puis jointes avec la différence absolue élément par élément. Leur jeu de données est composé de près de 1 000 images carrées d'ostraca égyptiens, dont 100 sont conservées pour la validation. L'exactitude de validation s'élève à 94 %.

Pour terminer, remarquons que dans toutes les approches présentées dans cette sous-section, on tient pour acquise l'orientation des images : dans le jeu de données, tous les fragments sont déjà à l'endroit. Cela ne correspond pas à toutes les réalités, dont celle des archéologues. Pendant la reconstruction, leurs tessons de poterie doivent nécessairement être pivotés pour trouver des paires compatibles. C'est pourquoi ces méthodes ne peuvent pas être utilisées pour résoudre la problématique choisie dans ce mémoire. L'approche proposée dans les chapitres subséquents résout ces limitations.

2.3 Apprentissage de métriques

L'apprentissage de métriques (*metric learning*) [33] peut être vu comme une sous-famille des réseaux siamois. Un réseau siamois classificateur binaire traditionnel se comporte comme une fonction, notée M_{classif}, qui reçoit deux images et qui produit un nombre réel compris entre 0 et 1. La prédiction booléenne est vraie si la sortie du réseau est supérieure à 0,5 :

$$M_{\text{classif}}(\text{image}_1, \text{image}_2) > 0{,}5. \tag{2.1}$$

Dans l'apprentissage de métriques, les deux vecteurs issus des deux branches ne sont pas concaténés et envoyés dans d'autres couches : ils sont comparés en calculant leur distance. Ici, le modèle n'effectue pas de comparaison, mais transforme chaque image en vecteur de caractéristiques (*feature vectors*). Ce modèle se comporte donc comme une fonction, notée M_{ML}, qui reçoit une seule image et qui produit un vecteur élément de $\mathbb{R}^N$. Ces vecteurs sont ensuite comparés à l'aide d'une métrique [5] d (habituellement, la métrique euclidienne). La prédiction booléenne est vraie si la distance est inférieure à un seuil prédéterminé :

$$d(M_{\text{ML}}(\text{image}_1), M_{\text{ML}}(\text{image}_2)) < \text{seuil}. \tag{2.2}$$

Ainsi, si les deux vecteurs de caractéristiques sont près l'un de l'autre dans l'espace $\mathbb{R}^N$, le réseau prédit que les deux vecteurs sont similaires. Dans notre contexte, la notion de similarité correspondrait à ce que deux fragments soient adjacents.

Il existe plusieurs techniques pour entraîner un réseau à produire de bons vecteurs de caractéristiques, notamment le *contrastive loss* [34, 35] et le *triplet loss* [36]. Ce dernier a initialement été employé pour la comparaison de visages (*FaceNet*). Le *metric learning* est principalement avantageux lorsqu'il y a un grand nombre de classes pour lesquelles on dispose de peu de données d'entraînement (*one-shot learning*). Un second avantage est sa vitesse : le vecteur de caractéristiques de chaque entrée n'a qu'à être extrait une fois. Ensuite, les comparaisons s'effectuent à l'aide de la distance euclidienne, ce qui se calcule rapidement.

Cependant, certains auteurs notent que les classificateurs siamois comportent des atouts non négligeables par rapport à l'apprentissage de métriques [37, 38]. Les comparaisons sont généralement plus exactes, bien qu'un peu plus lentes. De plus, l'entraînement est habituellement plus stable, tandis que les modèles avec *metric learning* peuvent être plus capricieux. C'est pour cette raison que dans ce mémoire, nous nous

5. Une métrique est une fonction qui calcule la distance, un nombre réel, entre deux éléments. Ici, ces éléments sont des vecteurs.

concentrons sur les architectures siamoises, et non l'apprentissage de métriques.

2.4 Réseaux antagonistes génératifs

Les réseaux antagonistes génératifs (*generative adversarial network*, ou GAN) sont utilisés, entre autres, pour générer ou transformer des images [39]. Un modèle GAN est en réalité composé de deux réseaux : le générateur et le discriminateur. Le générateur produit une sortie pour laquelle il est entraîné (par exemple : une image aléatoire d'un visage ou un modèle 3D d'une poterie reconstruite à partir de ses pièces). Le discriminateur est un classificateur binaire qui doit déterminer si ce qu'on lui envoie est une vraie donnée du jeu d'entraînement (par exemple : une photo d'un visage réel ou une poterie complète) ou un « faux » issu du générateur. Les deux réseaux agissent donc comme des antagonistes, des adversaires, qui s'entraînent dans une boucle. Le générateur s'améliore à partir des prédictions du discriminateur pour chercher à le tromper. En revanche, plus le générateur se perfectionne, plus le discriminateur devient critique, ce qui force encore plus le générateur à produire des sorties à l'allure authentique.

Rafique et coll. [40] ont utilisé un GAN pour résoudre de petits puzzles (9 pièces carrées mélangées dans une grille 3×3). Leurs résultats montrent que même pour des images simples, des détails importants sont perdus pendant la reconstruction. Cela s'explique par la nature d'un GAN : ce réseau n'a aucun incitatif à réutiliser les fragments d'images tels quels. Les auteurs jugent ainsi que cette architecture n'est pas un bon choix pour réassembler des objets dont les fragments sont connus.

Toutefois, Li et coll. [41] ont également entraîné un GAN pour résoudre cette problématique et ont obtenu de meilleurs résultats. Le réseau, au lieu de produire la solution du puzzle, génère la permutation qui remet les pièces dans l'ordre. Cette

méthode réussit avec une exactitude de 79 %. Uniquement des images carrées de maisons, de personnes, de guitares et d'éléphants ont été utilisées.

Cependant, tout comme la majorité des approches précédentes, le nombre de fragments, leur forme et leur orientation sont restreints. C'est pourquoi nous n'avons pas exploré davantage cette avenue.

2.5 Conclusion

L'état de l'art a résumé quelques approches algorithmiques et a mis en lumière certaines de leurs limitations. Nous avons ensuite présenté les réseaux de neurones convolutifs et les architectures siamoises, sur lesquels se basent les travaux de ce mémoire. Celles-ci sont formellement introduites avec plus de détails au chapitre suivant. Finalement, nous avons abordé d'autres approches issues de l'apprentissage profond, soit l'apprentissage de métriques et les réseaux antagonistes génératifs, et pourquoi nous ne les avons pas approfondies dans le cadre de ces travaux de recherche.

Chapitre 3

Entraînement de réseaux de neurones

Ce chapitre met en place le cadre théorique nécessaire à la méthodologie du prochain chapitre. Nous débutons par la présentation des concepts liés à l'entraînement d'un modèle d'apprentissage profond. Nous définissons ensuite quelques architectures neuronales, dont les réseaux siamois, introduits au chapitre précédent.

Les notations, définitions et concepts présentés dans ce chapitre sont tirés de *Deep Learning* par Goodfellow et coll. [42], de *Dive into Deep Learning* de Zhang et coll. [43] et du cours CS230 de l'université Stanford [44].

3.1 Notation

Un résumé de la notation utilisée dans ce chapitre est présenté au tableau 3.1.

a	Un scalaire
$\boldsymbol{a}$	Un vecteur
a_i	L'élément i du vecteur $\boldsymbol{a}$
$\boldsymbol{A}$	Une matrice
$A_{i,j}$	L'élément (i, j) de la matrice $\boldsymbol{A}$
A	Un tenseur d'ordre supérieur ou égal à 3
$\mathsf{A}_{i,j,k}$	L'élément (i, j, k) du 3-tenseur A
$\mathbb{R}$	L'ensemble des nombres réels
$\mathbb{R}^n$	L'ensemble des vecteurs réels de dimension n
$\mathbb{R}^{H \times W}$	L'ensemble des matrices réelles de dimensions $H \times W$
$\mathbb{R}^{H \times W \times C}$	L'ensemble des 3-tenseurs réels de dimensions $H \times W \times C$
$\mathbb{X}$	L'ensemble des données d'entraînement
m	Le nombre de données d'entraînement, soit card($\mathbb{X}$)
$\boldsymbol{x}^{(i)}$	L'élément i de l'ensemble $\mathbb{X}$
$y^{(i)}$	L'étiquette de $\boldsymbol{x}^{(i)}$
$\hat{y}^{(i)}$	Une prédiction de $y^{(i)}$
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$	Le gradient de la fonction f par rapport à $\boldsymbol{x}$
$\mathbf{1}_{\text{condition}}$	1 si la condition est vraie, 0 sinon (fonction caractéristique).

TABLE 3.1 – Notation utilisée dans ce chapitre.

Rappelons qu'un **tenseur** est une généralisation à n dimensions des matrices. À titre d'exemples, un vecteur est un tenseur d'ordre 1, ou 1-tenseur, et une matrice est un 2-tenseur. Comme une matrice réelle peut être vue comme un tableau de nombres, un 3-tenseur est analogue à un prisme (une superposition de matrices) ou bien une matrice dont les éléments sont des vecteurs.

3.2 Entraînement

L'entraînement est à la base du domaine de l'apprentissage automatique (*machine learning*) et, donc, de l'apprentissage profond (*deep learning*). En apprentissage supervisé, l'entraînement utilise un jeu de données : un ensemble $\mathbb{X}$ de taille m où chaque élément $\boldsymbol{x}^{(i)} \in \mathbb{X}$ possède une étiquette $y^{(i)}$. Dans le cadre d'une classification binaire, $y^{(i)} \in \{0, 1\}$. L'objectif est de trouver un ensemble de paramètres $\boldsymbol{\theta}$ tel que les prédictions d'un modèle M, $\hat{y}^{(i)} = M(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$, sont les plus proches possibles des vraies étiquettes $y^{(i)}$.

Définition 3.1. Un **modèle** est une fonction M paramétrée par $\boldsymbol{\theta}$ et qui reçoit un élément d'entraînement $\boldsymbol{x}^{(i)}$ afin de prédire son étiquette $y^{(i)}$. On note cette prédiction

$$\hat{y}^{(i)} = M(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}). \tag{3.1}$$

En pratique, les paramètres d'un modèle peuvent être des vecteurs, des matrices et des tenseurs. Sans perte de généralité, on considère que $\boldsymbol{\theta}$ est un vecteur réel.

3.2.1 Fonction de perte

Une fonction de perte définit numériquement l'objectif de l'apprentissage. Au fur et à mesure de l'entraînement, on souhaite que la perte diminue. Ainsi, lorsque le modèle M s'améliore et que les prédictions $\hat{y}^{(i)}$ sont plus près des vraies étiquettes $y^{(i)}$, la valeur retournée par la fonction de perte doit être de plus en plus petite.

Définition 3.2. Une **fonction de perte** (*loss function*), notée L, est une fonction dérivable par rapport à $\boldsymbol{\theta}$ et qui évalue l'erreur commise par un modèle paramétré par $\boldsymbol{\theta}$.

Toute fonction de perte doit être dérivable par rapport aux paramètres du modèle afin d'être en mesure d'utiliser la descente de gradient, un algorithme d'apprentissage présenté plus loin (définition 3.10).

Définition 3.3. L'**erreur quadratique moyenne** (*mean squared error*, ou MSE) est une fonction de perte d'équation

$$L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) = (y^{(i)} - \hat{y}^{(i)})^2, \tag{3.2}$$

où $\hat{y}^{(i)} = M(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \in \mathbb{R}$ et $y^{(i)} \in \mathbb{R}$.

La MSE est une fonction de perte qui peut être utilisée dans plusieurs contextes d'apprentissage, dont la régression. Cependant, il ne s'agit pas de la plus appropriée pour la classification binaire, où $y^{(i)} \in \{0, 1\}$ et $\hat{y}^{(i)} \in (0, 1)$.

Définition 3.4. L'**entropie croisée** (*cross-entropy*) est une fonction de perte d'équation

$$L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) = -y^{(i)} \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}), \tag{3.3}$$

où $\hat{y}^{(i)} = M(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \in (0, 1)$ et $y^{(i)} \in \{0, 1\}$.

L'entropie croisée est préférable à la MSE, car elle augmente de plus en plus rapidement lorsque le modèle s'éloigne de la bonne étiquette (voir figure 3.1). Ses fondements théoriques s'appuient sur l'estimation par maximum de vraisemblance.

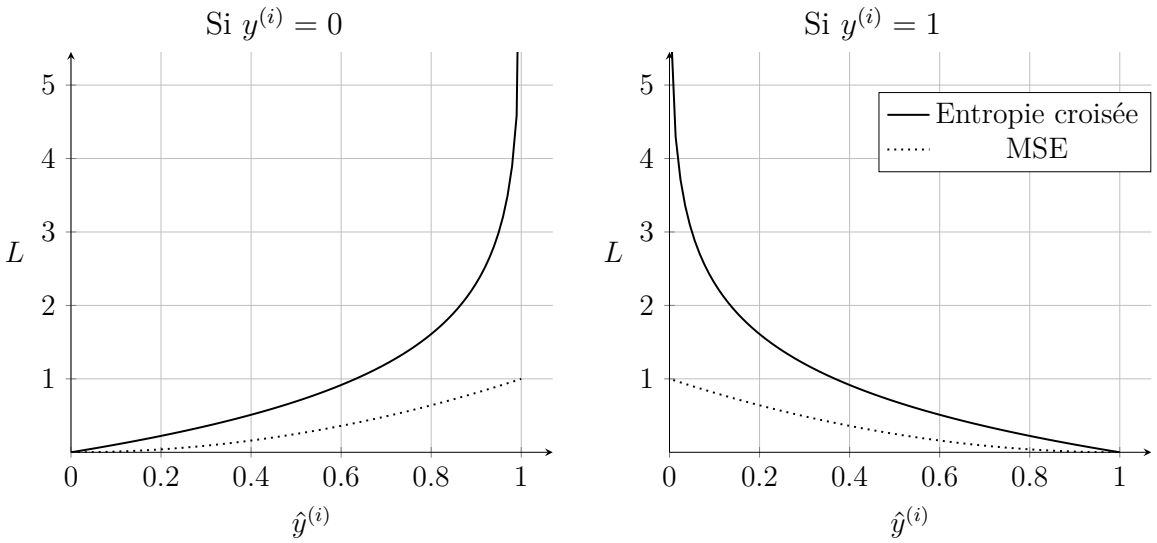


FIGURE 3.1 – Comparaison de la MSE et de l'entropie croisée. Lorsque la prédiction $\hat{y}^{(i)}$ s'éloigne de la vraie étiquette $y^{(i)}$ (0 ou 1), l'entropie croisée augmente plus rapidement et tend vers l'infini, tandis que la MSE vaut, au maximum, 1. Dans les deux cas, la perte est nulle lorsque la prédiction $\hat{y}^{(i)}$ égale $y^{(i)}$, la valeur désirée.

Pendant l'entraînement, on calcule la moyenne des valeurs de la fonction de perte pour chaque élément de l'ensemble d'entraînement $\mathbb{X}$. Puisque M, L, et les $\boldsymbol{x}^{(i)}$ et $y^{(i)}$ de $\mathbb{X}$ sont fixes, cette moyenne dépend uniquement de $\boldsymbol{\theta}$ et est définie par

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}). \tag{3.4}$$

Comme L est dérivable par rapport à $\boldsymbol{\theta}$, J l'est aussi.

3.2.2 Métriques

Une fonction de perte possède les propriétés nécessaires pour entraîner un modèle : elle est dérivable et diminue lorsque le modèle s'améliore. Cependant, il est souvent difficile d'analyser les variations de la valeur de perte. C'est pourquoi on calcule, en plus, des métriques. Il est important de noter qu'il n'est pas ici question de *metric learning* (section 2.3).

Définition 3.5. Une **métrique** est une fonction qui mesure la performance d'un modèle.

Contrairement à la définition de fonction de perte (3.2), la définition de métrique ne comporte aucune restriction et est volontairement floue pour permettre de s'adapter aux besoins de la tâche d'apprentissage. On évalue une métrique afin d'interpréter plus concrètement et intuitivement la performance d'un modèle sur un ensemble de données.

Définition 3.6. Pour évaluer la performance d'un classificateur binaire, on utilise les définitions suivantes.

$$\textbf{VP (vrais positifs)} = \sum_{i=1}^{m} \mathbf{1}_{y^{(i)}=1 \wedge \hat{y}^{(i)}=1} \tag{3.5}$$

$$\textbf{FP (faux positifs)} = \sum_{i=1}^{m} \mathbf{1}_{y^{(i)}=0 \wedge \hat{y}^{(i)}=1} \tag{3.6}$$

$$\textbf{VN (vrais négatifs)} = \sum_{i=1}^{m} \mathbf{1}_{y^{(i)}=0 \wedge \hat{y}^{(i)}=0} \tag{3.7}$$

$$\textbf{FN (faux négatifs)} = \sum_{i=1}^{m} \mathbf{1}_{y^{(i)}=1 \wedge \hat{y}^{(i)}=0} \tag{3.8}$$

Par conséquent, $\mathrm{VP} + \mathrm{FP} + \mathrm{VN} + \mathrm{FN} = m$.

Définition 3.7. L'**exactitude** (*accuracy*) est la proportion d'éléments correctement classifiés.

$$\text{Exactitude} = \frac{\mathrm{VP} + \mathrm{VN}}{m} \tag{3.9}$$

Définition 3.8. La **précision** (*precision*) est la proportion de vrais positifs parmi les éléments classifiés positifs.

$$\text{Précision} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{3.10}$$

Définition 3.9. Le **rappel** (*recall*) est la proportion d'éléments positifs correctement classifiés.

$$\text{Rappel} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{3.11}$$

Ces trois métriques (l'exactitude, la précision et le rappel) ne peuvent pas être des fonctions de perte : elles ne sont pas dérivables par rapport à $\boldsymbol{\theta}$ et ne diminuent pas lorsque le modèle s'améliore. Toutefois, il est facile de les interpréter : par exemple, une exactitude de 95 % sur un certain jeu de données signifie que 95 % de ces données ont été classifiées correctement ($y^{(i)} = \hat{y}^{(i)}$).

3.2.3 Descente de gradient et rétropropagation de l'erreur

La descente de gradient est un algorithme utilisé pour trouver l'ensemble des paramètres $\boldsymbol{\theta}$ qui minimise la perte J.

Définition 3.10. La **descente de gradient** (*batch gradient descent*, BGD ou tout simplement *gradient descent*) est un algorithme qui modifie itérativement les paramètres d'un modèle en dérivant la perte J dans le but de la minimiser :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{3.12}$$

L'hyperparamètre $\epsilon \in (0, +\infty)$ est appelé le **taux d'apprentissage** (*learning rate*).

Le calcul de $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ se nomme **rétropropagation de l'erreur** et est évalué en utilisant la dérivation en chaîne.

3.2.4 Optimisation de la descente de gradient

En pratique, la descente de gradient à la définition 3.10 est rarement implémentée, car elle requiert de conserver en mémoire les gradients de toutes les données d'entraînement. La descente de gradient stochastique est une alternative moins coûteuse en espace mémoire. Cet algorithme évalue plutôt un sous-ensemble à la fois. La taille du sous-ensemble est fixée initialement en fonction de l'espace mémoire disponible.

Définition 3.11. La **descente de gradient stochastique** (*stochastic gradient descent*, ou SGD) est un algorithme qui met à jour les paramètres du modèle après l'évaluation d'un sous-ensemble $\mathbb{B} \subset \mathbb{X}$ de taille B et choisi aléatoirement, appelé mini-lot (*minibatch*).

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{B} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{B} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{3.13}$$

Il existe des algorithmes plus perfectionnés que SGD, comme Adam, dont l'efficacité a été démontrée empiriquement [45].

3.3 Architectures neuronales et types de couches

Les notions d'apprentissage supervisé présentées précédemment supposent l'existence d'un modèle M à entraîner, sans toutefois préciser la nature de M. Cette section introduit l'utilisation de réseaux de neurones comme modèle.

3.3.1 Couche linéaire

La couche linéaire, également dite dense ou fortement connectée (*fully connected*) est l'unité de base d'un réseau de neurones. Considérons ici qu'une couche contient des données sous la forme d'un vecteur de nombres réels.

Définition 3.12. Un **hyperparamètre** définit le comportement d'un algorithme, d'une architecture ou d'un modèle. Il est fixé initialement et ne change pas pendant l'apprentissage.

Remarquons que le taux d'apprentissage ϵ et la taille des mini-lots B de la descente de gradient stochastique (définition 3.11) sont des hyperparamètres.

Définition 3.13. Un **paramètre**, contrairement à un hyperparamètre, n'est pas fixe : il est modifié par un apprentissage. Un paramètre est généralement initialisé aléatoirement.

La définition de couche linéaire contient à la fois des hyperparamètres et des paramètres : les hyperparamètres sont fixés manuellement avant de débuter l'apprentissage, tandis que les paramètres sont initialisés aléatoirement, puis modifiés par l'algorithme d'apprentissage.

Définition 3.14. Une **couche linéaire** est l'image d'une fonction $f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ d'équation

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{3.14}$$

où :

- $\boldsymbol{x}$ est la couche d'entrée ou précédente ;
- les hyperparamètres n_x et n_y sont respectivement le nombre de neurones en entrée et en sortie ;
- les paramètres $\boldsymbol{W} \in \mathbb{R}^{n_y \times n_x}$ et $\boldsymbol{b} \in \mathbb{R}^{n_y}$ sont respectivement appelés les **poids** et les **biais**.

En pratique, le terme **couche linéaire** peut aussi désigner la fonction elle-même selon le contexte.

Le plus simple des réseaux de neurones contient donc au minimum deux couches. La première est la couche d'entrée (le vecteur $\boldsymbol{x}$) contenant les informations reçues pour effectuer la prédiction ; par exemple, un élément de $\mathbb{X}$. La seconde est une couche linéaire, $f(\boldsymbol{x})$. Comme il s'agit aussi de la dernière couche, on l'appelle couche de sortie, notée $\boldsymbol{y}$.

Par convention, $\boldsymbol{\theta}$ représente tous les paramètres d'une fonction. De plus, par simplicité, on peut omettre de spécifier les paramètres d'une fonction. La paramétrisation est alors implicite. Dans le contexte d'une couche linéaire f paramétrée par $\boldsymbol{W}$ et $\boldsymbol{b}$, les notations $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$, $f(\boldsymbol{x}; \boldsymbol{\theta})$ et $f(\boldsymbol{x})$ sont équivalentes.

3.3.2 Classification binaire avec un perceptron multicouche

Comme son nom l'indique, une couche linéaire ne peut modéliser qu'un comportement linéaire. Les fonctions d'activation remédient à cette limitation.

Définition 3.15. Une **fonction d'activation** est une fonction non linéaire σ qui est appliquée à la sortie d'une couche neuronale. Le choix de σ peut être vu comme un hyperparamètre.

En combinant des fonctions d'activation à une suite de couches linéaires, on peut ainsi approximer une gamme plus vaste de fonctions. C'est ce que l'on appelle un perceptron multicouche (*multilayer perceptron*, ou MLP).

Définition 3.16. Un **perceptron multicouche** est une fonction $f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ d'équation

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \left(f^{[L+1]} \circ f^{[L]} \circ \cdots \circ f^{[1]}\right)(\boldsymbol{x}; \boldsymbol{\theta}) \tag{3.15}$$

où

— les couches linéaires intermédiaires $(\boldsymbol{h}^{[1]}, \ldots, \boldsymbol{h}^{[L]})$ et la couche linéaire de sortie $(\boldsymbol{y} = \boldsymbol{h}^{[L+1]})$ sont définies par

$$\begin{gathered} f^{[l]} : \mathbb{R}^{n_h^{[l-1]}} \to \mathbb{R}^{n_h^{[l]}} \\ \boldsymbol{h}^{[l]} = f^{[l]}(\boldsymbol{h}^{[l-1]}; \boldsymbol{\theta}) = \sigma^{[l]}\left(\boldsymbol{W}^{[l]}\boldsymbol{h}^{[l-1]} + \boldsymbol{b}^{[l]}\right), \forall l \in \{1, \ldots, L+1\} \end{gathered} \tag{3.16}$$

— l'hyperparamètre L est le nombre de couches intermédiaires, dites couches cachées ;

— les hyperparamètres $n_h^{[l]}$ sont les tailles des couches linéaires ;

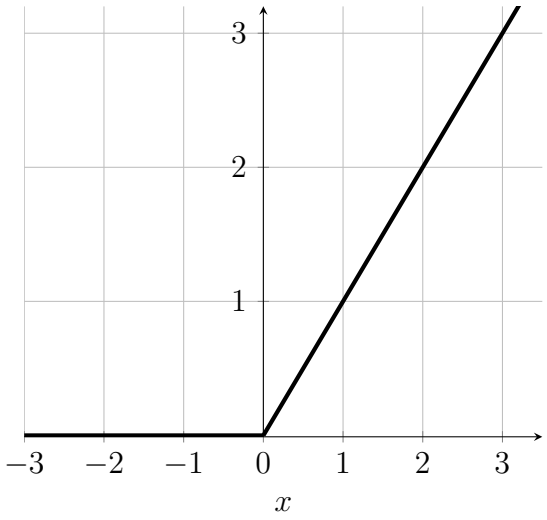
— $n_x = n_h^{[0]}$ et $n_y = n_h^{[L+1]}$;

— $\sigma^{[l]}$ est la fonction d'activation de la l^{e} couche.

De nos jours, la fonction d'activation ReLU (*Rectified Linear Unit*) [46] est parmi les plus utilisées pour les couches cachées étant donné sa simplicité et sa performance.

Définition 3.17. La fonction d'activation **ReLU** : $\mathbb{R} \to \mathbb{R}$ est définie par

$$\mathrm{ReLU}(x) = \max\{0, x\}. \tag{3.17}$$



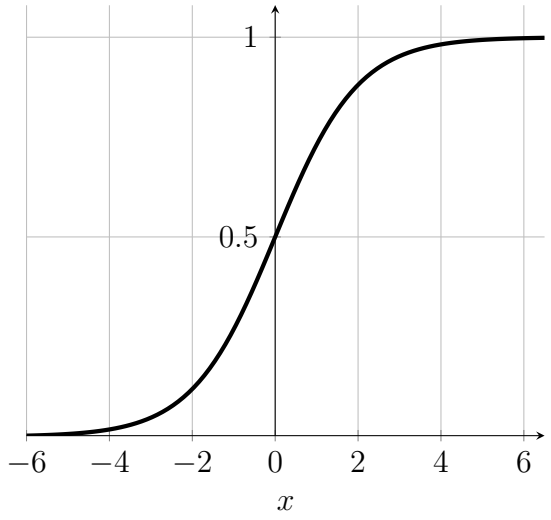
Lorsque ReLU est appliquée à un tenseur, elle est appliquée à chacun de ses éléments. Par exemple, la généralisation ReLU : $\mathbb{R}^n \to \mathbb{R}^n$ est définie par

$$\mathrm{ReLU}(\boldsymbol{x})_i = \max\{0, x_i\}. \tag{3.18}$$

Lorsqu'on souhaite utiliser un MLP pour une tâche de classification binaire, on choisit $n_y = 1$, c'est-à-dire que le réseau ne produit qu'un nombre (figure 3.2). On voudra que la sortie soit 1 pour une prédiction positive, et 0 pour une prédiction négative. Pour forcer une sortie comprise entre 0 et 1, on choisit la fonction d'activation sigmoïde pour $\sigma^{[L+1]}$. La figure 3.2 illustre un tel réseau.

Définition 3.18. La fonction **sigmoïde** est une fonction d'activation définie par

$$\begin{aligned} \sigma &: \mathbb{R} \to (0,1) \\ \sigma(x) &= \frac{1}{1+e^{-x}}. \end{aligned} \tag{3.19}$$



3.3.3 Classification binaire d'images avec un réseau de neurones convolutif

Les perceptrons multicouches sont conçus pour recevoir en entrée un vecteur de données. Que faire si on veut plutôt classifier des images stockées dans des matrices ou des tenseurs ? Une option simple serait de transformer les représentations matricielles en vecteurs. Par exemple, une image de 28×28 pixels deviendrait un vecteur de 784 éléments. Cela fonctionne pour de petites images simples [47]. Or, ce faisant, on perd la relation spatiale entre des pixels à proximité les uns des autres. C'est pourquoi un être humain peut rapidement saisir le sens d'une photo sous forme matricielle, mais pas sous forme vectorielle. Les couches convolutives ont été inventées pour exploiter la structure spatiale des données audiovisuelles [48, 26]. Depuis un peu plus de 10 ans, les réseaux de neurones convolutifs (*convolutional neural networks*, ou CNN) et leurs

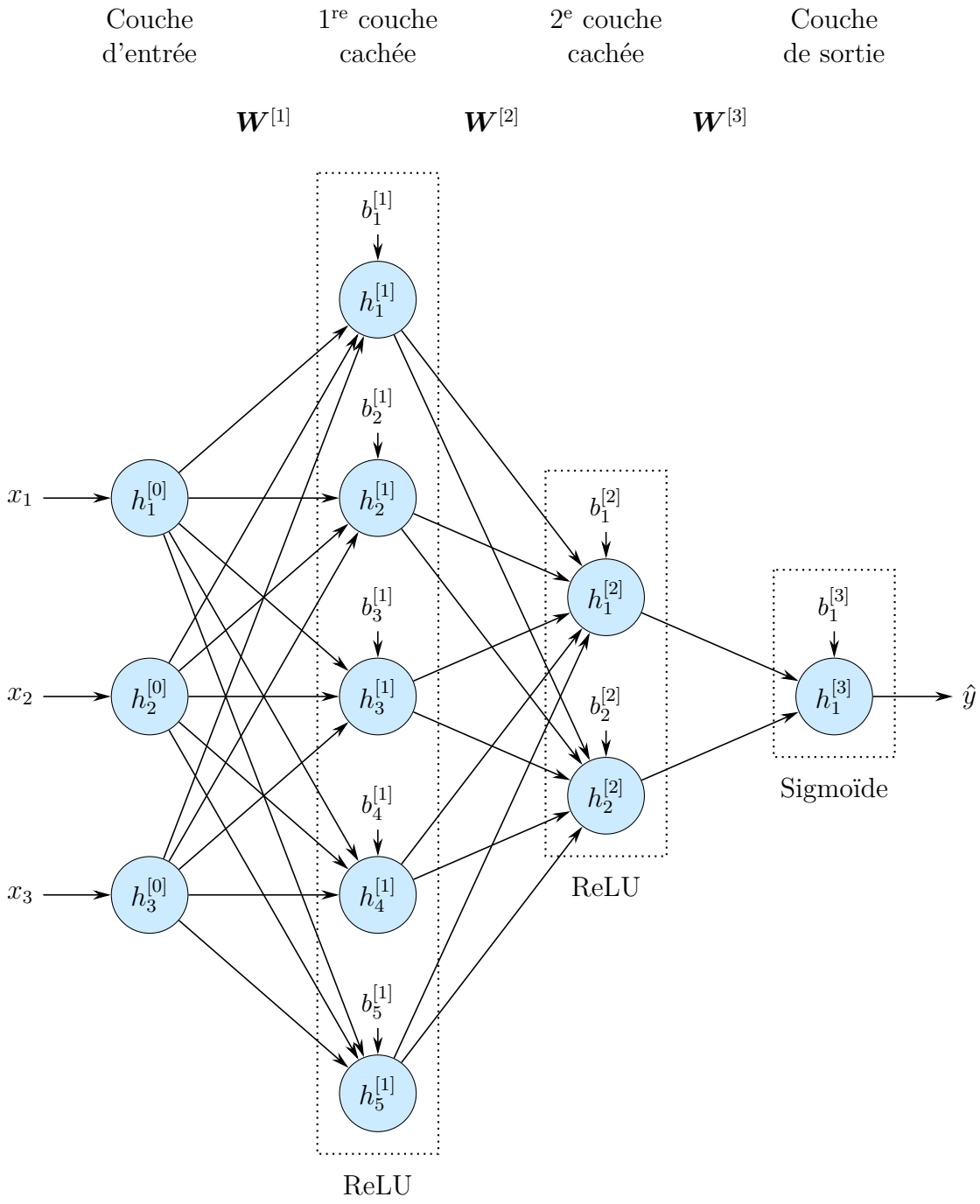


FIGURE 3.2 – Exemple d'architecture d'un MLP (définition 3.16) classificateur binaire avec deux couches cachées. Les données reçues par le réseau constituent la couche d'entrée $\boldsymbol{h}^{[0]} = \boldsymbol{x}$, où $n_x = n_h^{[0]} = 3$. La première couche cachée, où $\sigma^{[1]} = \text{ReLU}$, est obtenue par $\boldsymbol{h}^{[1]} = f^{[0]}(\boldsymbol{h}^{[0]}) = \text{ReLU}(\boldsymbol{W}^{[1]}\boldsymbol{h}^{[0]} + \boldsymbol{b}^{[1]})$. Ce processus est répété jusqu'à obtenir $\hat{y}$, la prédiction produite par le réseau. Comme la dernière fonction d'activation ($\sigma^{[3]}$) est sigmoïde, $\hat{y}$ est compris entre 0 et 1.

variantes constituent l'approche dominante pour de nombreuses tâches en vision par ordinateur [27].

Définition 3.19. La **convolution entre deux fonctions** réelles f et g est définie par

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\mathrm{d}t. \tag{3.20}$$

Définition 3.20. La **corrélation croisée entre deux fonctions** réelles f et g est définie par

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(t)g(x+t)\mathrm{d}t. \tag{3.21}$$

Définition 3.21. Une **couche convolutive** qui reçoit en entrée un tenseur $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ produit en sortie un tenseur $\mathbf{H} \in \mathbb{R}^{H' \times W' \times C'}$ défini par

$$\mathsf{H}_{i,j,d} = \sigma\left(b_d + \sum_{a=1}^{K}\sum_{b=1}^{K}\sum_{c=1}^{C} \mathsf{X}_{i+a-1,j+b-1,c}\mathsf{K}_{a,b,c,d}\right) \tag{3.22}$$

où :

- H, W et C sont respectivement la hauteur, la largeur et le nombre de canaux de l'image en entrée ;
- $H' = H - K + 1$ et $W' = W - K + 1$;
- les paramètres $\mathbf{K} \in \mathbb{R}^{K \times K \times C \times C'}$ et $\boldsymbol{b} \in \mathbb{R}^{C'}$ sont respectivement les filtres (*filters* ou *kernels*) et les biais de la couche ;
- l'hyperparamètre K définit la taille des filtres carrés de la couche ;
- l'hyperparamètre C' est le nombre de canaux du tenseur de sortie ;
- $\sigma : \mathbb{R} \to \mathbb{R}$ est une fonction d'activation.

Cette définition suppose que les filtres de la couche sont carrés ($K \times K$), car c'est habituellement ce qui est fait en pratique. En théorie, ils pourraient être rectangulaires.

Remarquons que la définition de la convolution de fonctions comporte une intégrale (éq. 3.20), tandis qu'une couche convolutive n'en utilise pas. L'opération est discrétisée en utilisant des sommes et appliquée à deux dimensions (index a et index b). Cette procédure est illustrée à la figure 3.3. Le filtre $\mathbf{K}$ se déplace rangée par rangée et colonne par colonne sur la matrice en entrée $\mathbf{X}$. À chaque position, les éléments vis-à-vis de $\mathbf{X}$ et $\mathbf{K}$ sont multipliés puis additionnés, ce qui constitue essentiellement un produit scalaire. Cette somme est placée dans la matrice résultante $\mathbf{H}$. Par exemple, pour la première opération de la figure 3.3, le nombre 156 est placé dans la matrice résultante, car

$$0 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 + 6 \cdot 3 + 7 \cdot 4 + 8 \cdot 5 + 2 \cdot 6 + 3 \cdot 7 + 4 \cdot 8 = 156.$$

La fenêtre $\mathbf{K}$ se déplace ensuite sur toutes les positions possibles pour effectuer ce calcul et obtenir la matrice $\mathbf{H}$.

De plus, la convolution de fonctions (éq. 3.20) utilise $x - t$ comme décalage, tandis qu'on retrouve $i + a$ et $j + b$ à la définition précédente (éq. 3.22). C'est qu'une couche convolutive implémente en réalité l'opération de **corrélation croisée** ($\star$, éq. 3.21). Théoriquement, cette différence n'est pas importante, car dans les deux cas, des données équivalentes sont produites. Toutefois, lorsqu'on représente une corrélation croisée comme à la figure 3.3, le résultat est intuitif : si le filtre est dans le coin supérieur gauche, la valeur obtenue se retrouve dans le coin supérieur gauche de la matrice résultante. Toutefois, si l'opération de convolution était appliquée à la lettre, la valeur obtenue se retrouverait dans le coin inférieur droit, ce qui n'est pas très intuitif visuellement. C'est pourquoi les couches convolutives emploient plutôt la corrélation croisée.

Les couches convolutives possèdent d'autres hyperparamètres pour ajuster la taille du tenseur de sortie. Le remplissage (*padding*) ajoute un certain nombre de bordures de pixels vides autour de l'image en entrée. Le *stride* (par défaut 1) modifie le pas

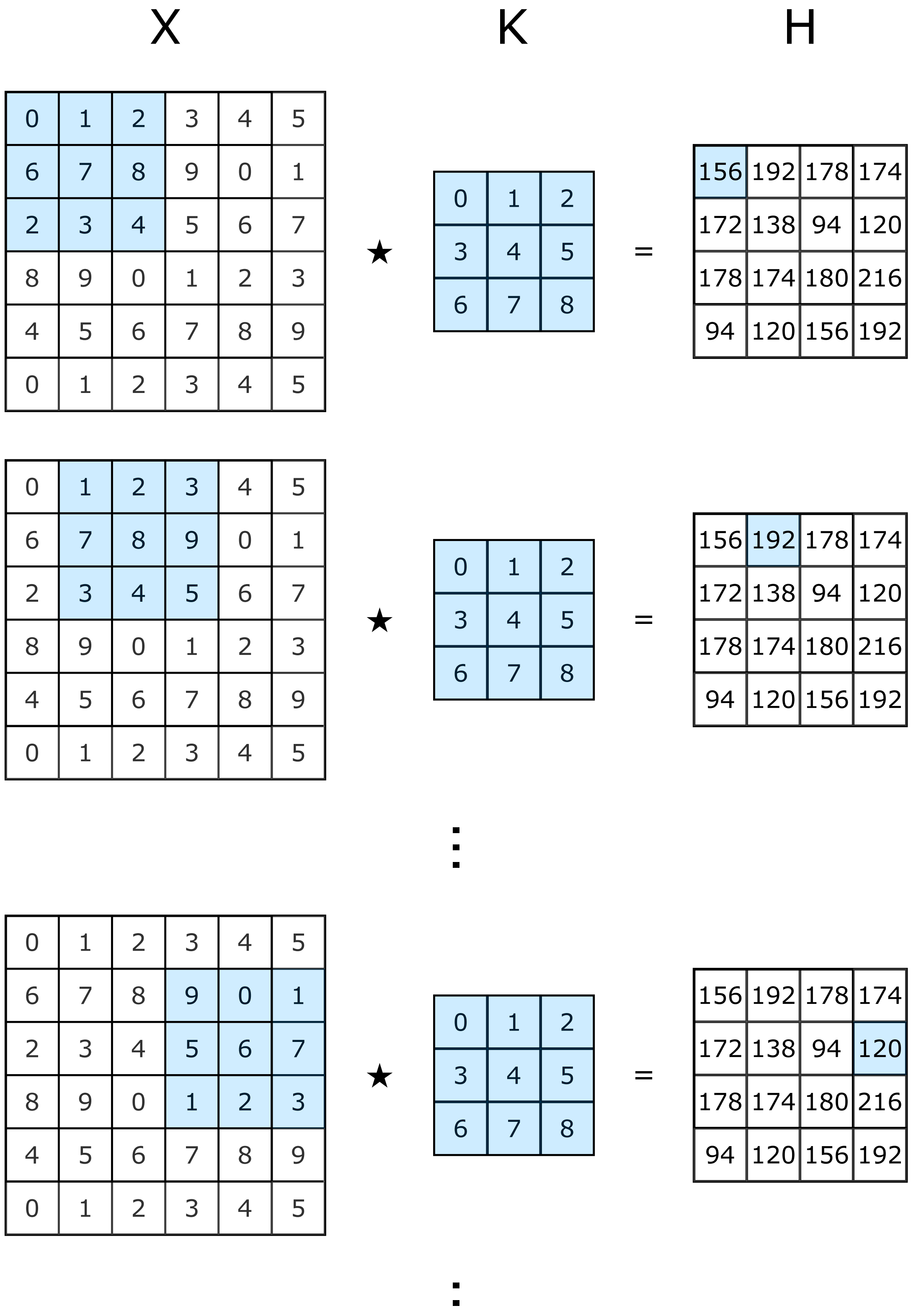


FIGURE 3.3 – L'opération de corrélation croisée discrète effectuée par une couche convolutive.

qu'effectue la fenêtre ($\mathbf{K}$) lorsqu'elle se déplace sur l'image ($\mathbf{X}$). L'augmenter réduit drastiquement W' et H' :

$$W' = \frac{W - K + 2 \times \text{padding}}{\text{stride}} + 1 \quad \text{et} \quad H' = \frac{H - K + 2 \times \text{padding}}{\text{stride}} + 1. \tag{3.23}$$

En règle générale, une architecture CNN de classificateur binaire est formée de deux parties :

$$\text{Classifier}(\mathbf{X}) = \text{MLP}(\text{Flatten}(\text{Encoder}(\mathbf{X}))) \tag{3.24}$$

La première, appelée l'**encodeur** (*encoder*, *feature extractor* ou *backbone*), est une suite de couches convolutives qui produit un tenseur de caractéristiques. Ce tenseur est ensuite transformé en vecteur (ses éléments sont mis les uns à la suite des autres), puis envoyé dans la deuxième partie, un perceptron multicouche. Tel que présenté à la sous-section précédente, ce MLP n'aura qu'un neurone de sortie doté de la fonction d'activation sigmoïde.

3.3.4 Apprentissage par transfert

L'entraînement de CNN peut demander beaucoup de données et de ressources. C'est pourquoi, lorsque possible, on démarre l'entraînement avec un encodeur préentraîné sur d'autres données. C'est ce qui est appelé l'**apprentissage par transfert** (*transfer learning*). Il faudra nécessairement un entraînement subséquent (*finetuning*) pour l'adapter à son nouveau contexte d'utilisation. Malgré tout, cette approche permet très souvent d'accélérer le processus et d'obtenir de meilleurs résultats [49].

3.3.5 Architecture siamoise

Le terme **réseau siamois** a été utilisé pour la première fois par Bromley et coll. [28] en 1993 pour authentifier des images de signatures. On souhaitait comparer deux signatures pour déterminer si elles avaient été rédigées par la même personne. L'une des deux provient d'une banque de données, tandis que la seconde est celle qui doit être validée. Il s'agit d'une tâche de classification binaire, mais avec deux entrées. Un CNN traditionnel ne peut effectuer cette tâche, car il ne peut recevoir qu'une image en entrée.

Un réseau siamois résout ce problème en acceptant deux images :

$$\text{SiameseClassifier}(\mathbf{X}_1, \mathbf{X}_2) = \text{MLP}(\text{Combine}(\text{Encoder}(\mathbf{X}_1), \text{Encoder}(\mathbf{X}_2))) \quad (3.25)$$

Pour ce faire, le même encodeur est utilisé sur les deux images. Les deux tenseurs produits représentent donc les mêmes caractéristiques. Ceux-ci sont ensuite combinés en un vecteur qui est classifié par un MLP.

L'opération de combinaison la plus commune est la conversion en vecteur (Flatten) de Encoder($\mathbf{X}_1$) et Encoder($\mathbf{X}_2$) suivie de la concaténation.

Définition 3.22. La **concaténation** de deux vecteurs

$$\boldsymbol{u} = \begin{pmatrix} u_1 & \cdots & u_{n_1} \end{pmatrix}^T \in \mathbb{R}^{n_1}$$

et

$$\boldsymbol{v} = \begin{pmatrix} v_1 & \cdots & v_{n_2} \end{pmatrix}^T \in \mathbb{R}^{n_2}$$

est définie par

$$\boldsymbol{u} \parallel \boldsymbol{v} = \begin{pmatrix} u_1 & \cdots & u_{n_1} & v_1 & \cdots & v_{n_2} \end{pmatrix}^T \in \mathbb{R}^{n_1+n_2} \tag{3.26}$$

Dans ce cas, l'équation du classificateur siamois devient

$$\text{SiameseClassifier}(\mathbf{X}_1, \mathbf{X}_2) = \text{MLP}(\text{Flatten}(\text{Encoder}(\mathbf{X}_1)) \parallel \text{Flatten}(\text{Encoder}(\mathbf{X}_2))) \tag{3.27}$$

3.3.6 Couche convolutive équivariante aux rotations

De par la nature des opérations de convolution et de corrélation croisée, une couche convolutive est une fonction équivariante au groupe des translations du plan [50]. Autrement dit, si une image subit une translation, la sortie du CNN subira la même translation tout en produisant les mêmes caractéristiques (figure 3.4).

Définition 3.23. Soit une fonction f et un groupe G qui agit sur le domaine de f (D_f) et son image. La fonction f est **équivariante** à G si

$$\forall x \in D_f, \forall T \in G, f(T[x]) = T[f(x)]. \tag{3.28}$$

Par exemple, le barycentre d'une forme géométrique dans le plan est équivariant aux groupes des translations et des rotations du plan.

Cette propriété est fondamentale et permet, en appliquant des opérations de moyennes et de maximums à la fin de l'encodeur CNN, d'obtenir un réseau invariant aux translations même si les couches convolutives, seules, ne le sont pas.

Définition 3.24. Soit une fonction f et un groupe G qui agit sur le domaine de f (D_f) et son image. La fonction f est **invariante** à G si

$$\forall x \in D_f, \forall T \in G, f(T[x]) = f(x). \tag{3.29}$$

Par exemple, l'aire d'une forme géométrique dans le plan est invariante aux groupes des translations et des rotations du plan.

Un CNN n'est cependant pas équivariant aux rotations (figure 3.5). C'est pourquoi Cohen et coll. ont introduit les *group equivariant CNN* (G-CNN) [50, 51]. Un G-CNN est équivariant à un groupe cyclique de rotations fixé au départ, par exemple, C_4, le groupe des rotations du carré (0°, 90°, 180° et 270°).

Dans un G-CNN, les filtres convolutifs sont évalués à chaque rotation θ du groupe cyclique choisi, en plus de toutes les positions x et y. Au lieu d'une matrice (2D, x et y), le G-CNN produit un tenseur à 3 dimensions (x, y et θ). Cette opération est équivariante aux translations du plan et aux rotations du groupe choisi (figure 3.6).

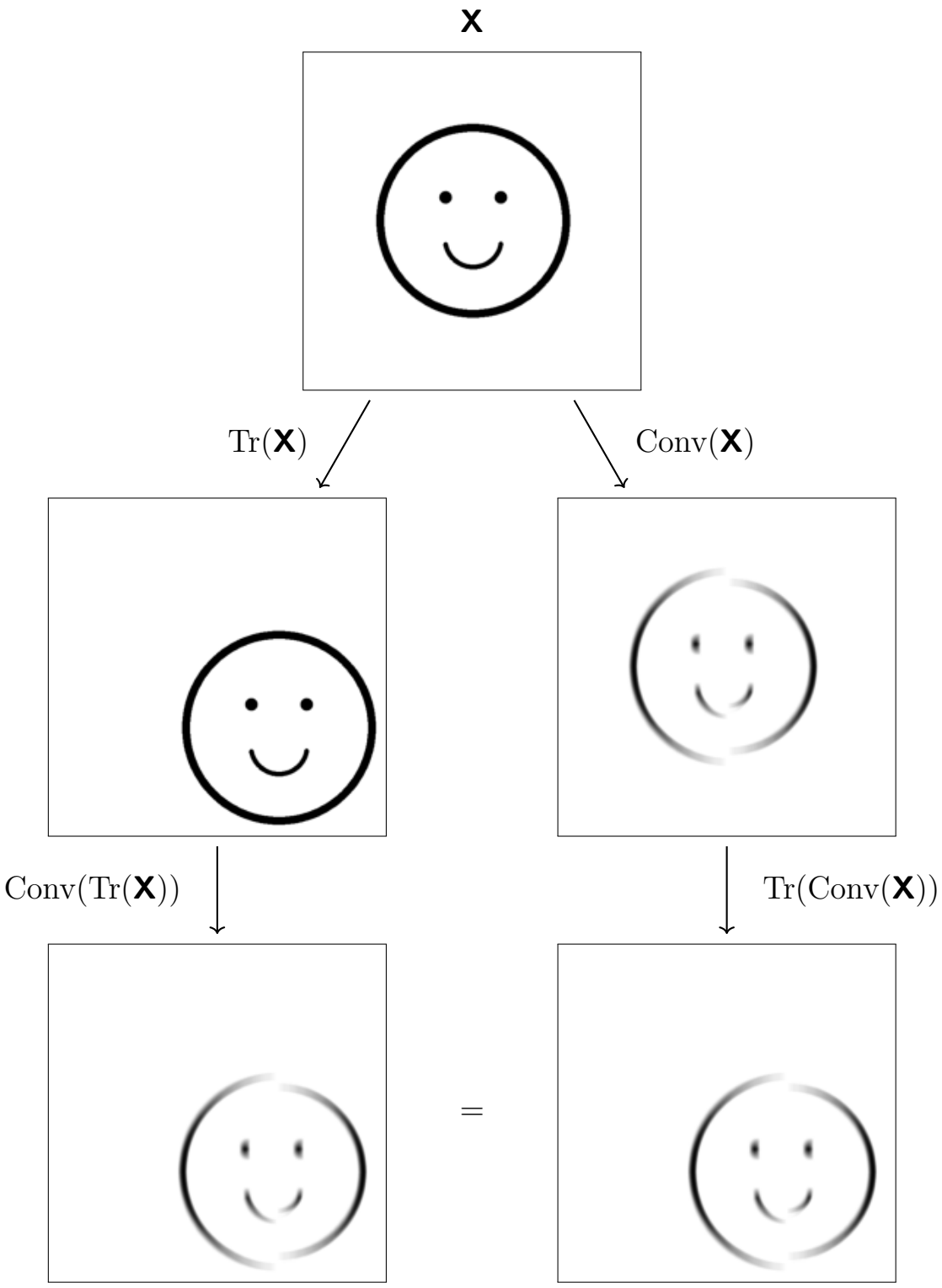


FIGURE 3.4 – Une couche convolutive est équivariante aux translations.

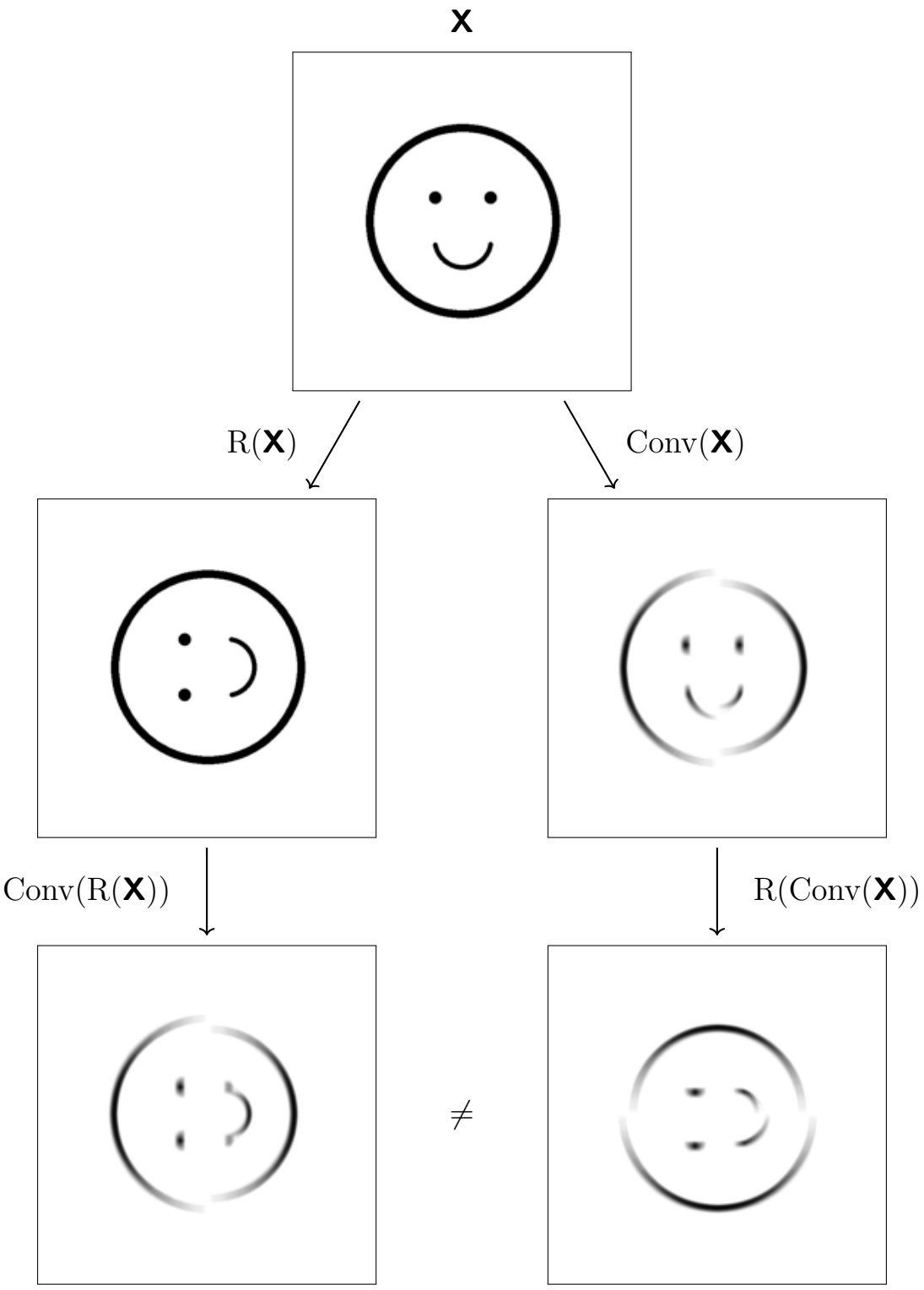


FIGURE 3.5 – Une couche convolutive n'est pas équivariante aux rotations.

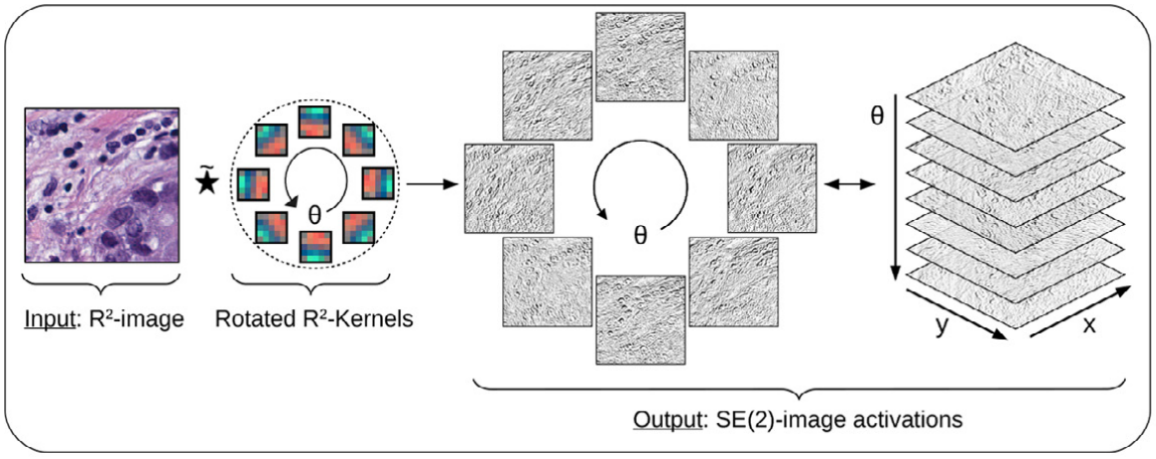


FIGURE 3.6 – Dans un G-CNN, le même filtre convolutif est appliqué à différentes rotations. Dans cet article de Lafarge et coll. [52], le groupe cyclique C_8 a été choisi, donc les filtres sont appliqués à chaque multiple de 45°. Cette opération est équivariante au groupe des translations du plan et au groupe C_8.

Comme le calcul de cette opération peut consommer beaucoup de temps et d'espace mémoire, elle est plutôt implémentée en utilisant des transformées de Fourier [53], servant notamment à la compression d'images et d'autres informations géométriques. Un tel réseau est appelé CNN orientable (*steerable CNN*).

3.4 Conclusion

Dans ce chapitre, nous avons défini plusieurs notions liées à l'apprentissage profond, dont la fonction de perte entropie croisée, les métriques communes en classification binaire, la descente de gradient, les perceptrons multicouches, les fonctions d'activation ReLU et sigmoïde, les architectures siamoises ainsi que les couches convolutives standards et équivariantes aux rotations. Ces concepts constituent la base de notre méthodologie, présentée au chapitre suivant.

Chapitre 4

Méthodologie

À partir du cadre théorique établi précédemment, nous présentons dans ce chapitre la méthodologie que nous avons élaborée pour résoudre la problématique abordée dans ce mémoire. Nous commençons par expliquer les algorithmes générant les données d'entraînement et de validation, puis nous exposons les réseaux de neurones en mesure de prédire l'adjacence de fragments et les approches choisies pour entraîner ces modèles.

4.1 Génération de données artificielles

La littérature présente peu de jeux de données d'images fragmentées dans le but de les reconstruire. Le et Li [9] ont introduit un jeu de données de 31 photos de paysage brisées en 9, 36 ou 100 morceaux. De plus, dans le cadre du concours DAFNE [19], 65 fresques ont chacune été artificiellement séparées en au moins 500 petits fragments érodés.

Ces deux jeux de données sont trop loin des objectifs de ce mémoire pour être utilisés. Le premier comporte peu d'images pour entraîner un modèle d'apprentissage profond, tandis que les fragments du second sont trop petits et lisses. Il faut donc essentiellement se fier à leurs couleurs pour déterminer les paires adjacentes, alors que la couleur, seule, n'est pas toujours une information fiable pour la reconstruction archéologique de poteries.

C'est pourquoi il s'est avéré nécessaire de produire un grand nombre de fragments en prévision de l'entraînement de réseaux de neurones. Le processus de génération combine deux algorithmes : le premier crée aléatoirement des patrons de fracture et le second applique des patrons de fracture sur une image pour la fragmenter.

4.1.1 Génération de patrons de fracture

Cette sous-section s'inspire du bruit de Perlin [54]. Ken Perlin a créé cet algorithme en 1983 pour rendre plus réalistes les textures et les surfaces dans les films d'animation. Pour ce faire, il emploie des nombres générés aléatoirement dans une grille 2D qui sont ensuite interpolés.

Ici, nous utilisons une version simplifiée et unidimensionnelle. Notre algorithme, que l'on nomme **algorithme de génération de patron de fracture**, produit un patron de fracture, une fonction $F : [0, 1] \to \mathbb{R}$. Il possède 5 paramètres constants en entrée :

- N : le nombre de générateurs de bruits aléatoires simples f_i composant le bruit final F (un nombre naturel, par exemple, 5) ;
- A_1 : l'amplitude initiale (un nombre réel positif, par exemple, 1) ;
- G_A : le gain de l'amplitude, habituellement inférieur à 1 (un nombre réel positif, par exemple, 0,5) ;
- P_1 : le nombre initial de points (un nombre naturel, par exemple, 4) ;

— G_P : le gain du nombre de points, habituellement supérieur à 1 (un nombre réel positif, par exemple, 2).

L'algorithme suit les étapes suivantes :

1. Affecter $i \leftarrow 1$.
2. Générer $P_i - 2$ nombres aléatoires tirés d'une loi uniforme $\mathcal{U}_{[-A_i, A_i]}$. On note ces nombres $p_{(i,1)}, \ldots, p_{(i,P_i-2)}$. On fixe $p_{(i,0)}$ et $p_{(i,P_i-1)}$ à 0. Ces P_i nombres correspondent aux ordonnées d'abscisses réparties équitablement sur l'intervalle $[0, 1]$, incluant les bornes. Par exemple, si $P_i = 4$, les points générés sont $(0, 0)$, $(\frac{1}{3}, p_{(i,1)})$, $(\frac{2}{3}, p_{(i,2)})$ et $(1, 0)$. Ces points définissent la fonction f_i (éq. 4.2).
3. Affecter $i \leftarrow i + 1$, $A_i \leftarrow G_A \cdot A_{i-1}$, $P_i \leftarrow \lfloor G_P \cdot P_{i-1} \rfloor$.
4. Répéter les étapes 2 et 3 jusqu'à obtenir N suites de points.

La définition du patron final est

$$\begin{aligned} F &: [0, 1] \to \mathbb{R} \\ F(x) &= \sum_{i=1}^{N} f_i(x) \end{aligned} \tag{4.1}$$

où

$$f_i(x) = \begin{cases} p_{(i,0)} & \text{si } x = 0, \\ p_{(i,P_i-1)} & \text{si } x = 1, \\ (1-t)p_{(i,s)} + t p_{(i,s+1)} & \text{sinon, où } r = (P_i - 1)x,\ s = \lfloor r \rfloor \text{ et } t = r - s. \end{cases} \tag{4.2}$$

La définition des f_i contient une interpolation linéaire entre les points générés : des segments sont construits entre les points de la suite qui définit la fonction. Notons que s est la partie entière de r, tandis que t est la partie fractionnaire de r. La figure 4.1 montre cinq fonctions f_i, tandis que la figure 4.2 présente la somme de celles-ci, c'est-à-dire le patron de fracture F.

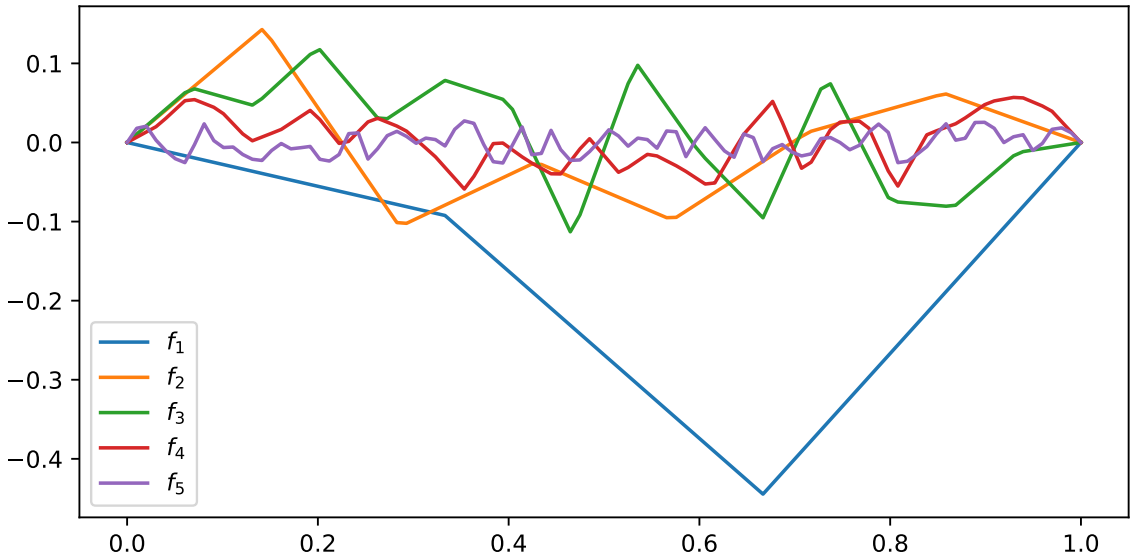


FIGURE 4.1 – 5 fonctions f_i générées avec les paramètres $N = 5$, $A_1 = 0.5$, $G_A = 0.5$, $P_1 = 4$ et $G_P = 2$. Lorsque i augmente, l'amplitude diminue (G_A est inférieur à 1) et le nombre de segments augmente (G_P est supérieur à 1). Leur somme crée le patron de fracture de la figure 4.2. Remarquons également que toutes les fonctions débutent et terminent à 0 : $f_i(0) = f_i(1) = 0$. Cette propriété est nécessaire pour l'algorithme de fragmentation d'images présenté plus bas.

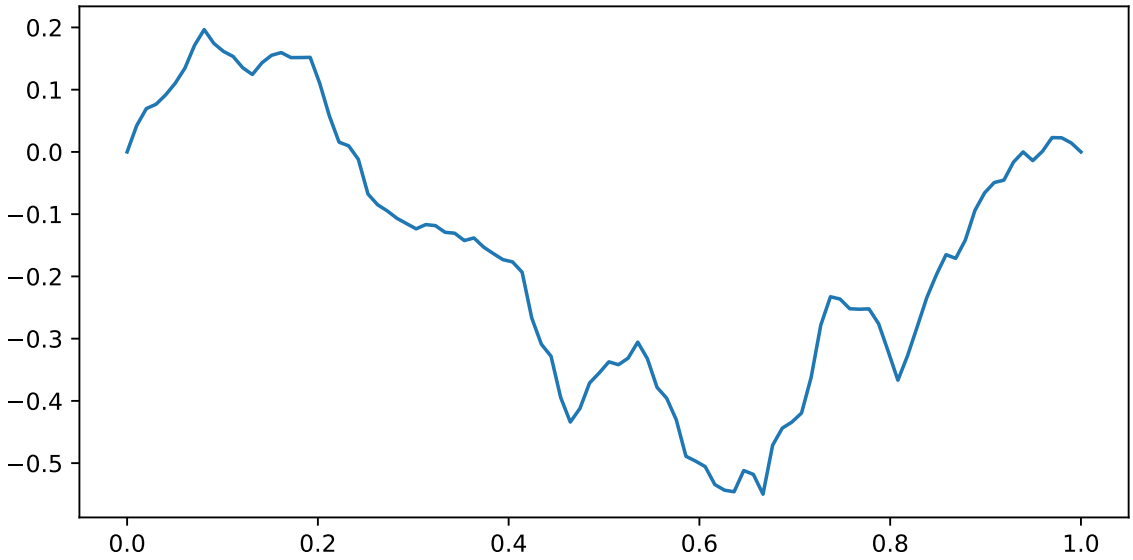


FIGURE 4.2 – Le patron de fracture F issu de la somme des 5 fonctions f_i de la figure 4.1.

4.1.2 Génération d'images fragmentées

Ce second algorithme, que l'on nomme **algorithme de fragmentation d'images**, permet de fragmenter une image rectangulaire. Pour ce faire, il faut appliquer récursivement des patrons de fracture sur l'image et les fragments obtenus. On vérifie d'abord si la hauteur de l'image est supérieure à sa largeur. Si c'est le cas, la première fracture sera horizontale. Sinon, elle sera verticale. Pour une fracture horizontale, on choisit aléatoirement un point sur le côté gauche de l'image et un sur le côté droit. Dans le cas d'une fracture verticale, on choisit plutôt un point sur le côté du haut, puis le côté du bas. On génère ensuite un patron de fracture selon l'algorithme de génération de patron de fracture et on le place sur l'image de sorte à relier les deux points. L'image d'origine est alors divisée en deux parties. On répète cette procédure récursivement jusqu'à obtenir le nombre désiré de fragments (figure 4.3). Pour appliquer l'algorithme à un morceau qui n'est plus rectangulaire, on trace virtuellement le rectangle circonscrit au fragment : c'est ce rectangle qui sera utilisé par l'algorithme pour choisir les points de début et de fin de la fracture.

4.2 Prédiction de matrices d'adjacence à l'aide d'un CNN siamois invariant aux rotations

Notre méthodologie de prédiction comporte deux étapes et, donc, deux réseaux de neurones distincts. Intéressons-nous au premier, qui s'occupe de générer la matrice d'adjacence entre deux fragments.

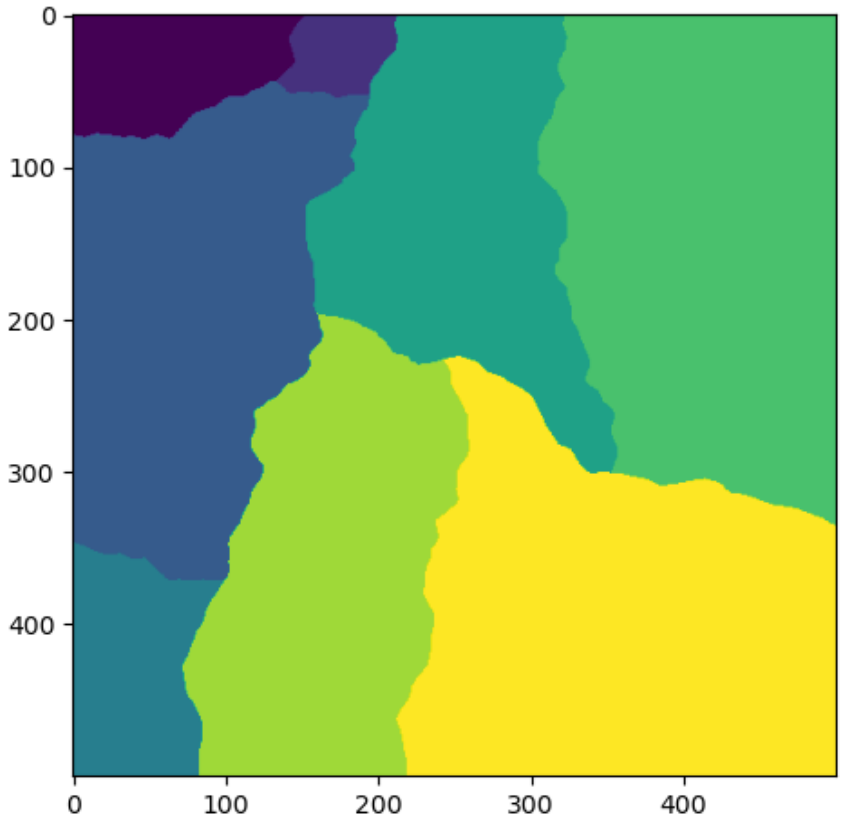


FIGURE 4.3 – Carré de 500×500 pixels divisé en 8 morceaux.

4.2.1 Matrices d'adjacence

Supposons que l'on souhaite évaluer l'adjacence de deux fragments F_1 et F_2. Soit C_1 le contour de F_1, L_1 la longueur de C_1, C_2 le contour de F_2 et L_2 la longueur de C_2. Comme les fragments sont des images, les contours sont composés d'un nombre fini de points : L_1 et L_2 se mesurent en pixels. On appellera **matrice d'adjacence** de F_1 et F_2 par le modèle M une matrice $\boldsymbol{A}$ de dimensions $L_1 \times L_2$ telle que

$$A_{i,j} = M(\text{voisinage du } i^{\text{e}} \text{ point de } C_1, \text{voisinage du } j^{\text{e}} \text{ point de } C_2), \tag{4.3}$$

où M est un CNN classificateur binaire siamois. Puisque les contours de deux fragments n'ont pas nécessairement la même longueur, L_1 et L_2 ne sont pas égaux. La matrice $\boldsymbol{A}$ n'est donc pas carrée. Les éléments de $\boldsymbol{A}$ sont des nombres réels compris entre 0 et 1 indiquant, selon le modèle, si les deux voisinages reçus semblent adjacents ou non. Par voisinage, on entend une partie de l'image du fragment prise autour du point, par exemple, un disque ayant un rayon de 15 pixels (figure 4.4).

4.2.2 Données

On utilise l'algorithme de fragmentation d'images décrit au début de ce chapitre pour générer des fragments pour l'entraînement. Lors d'une itération, toutes les paires adjacentes du jeu de données sont vues une fois par le modèle. Pour balancer l'entraînement, on choisit autant de paires non adjacentes parmi le grand nombre de choix possibles. Lorsqu'il s'agit d'une paire adjacente, on choisit aléatoirement un point sur la frontière commune des deux fragments adjacents, puis on extrait le voisinage de ce point sur chaque fragment. S'il s'agit d'une paire non adjacente, on extrait un voisinage quelconque sur les contours des deux fragments. Dans les deux cas, on obtient deux voisinages (deux petites images) qui constituent l'entrée du CNN classificateur binaire siamois.

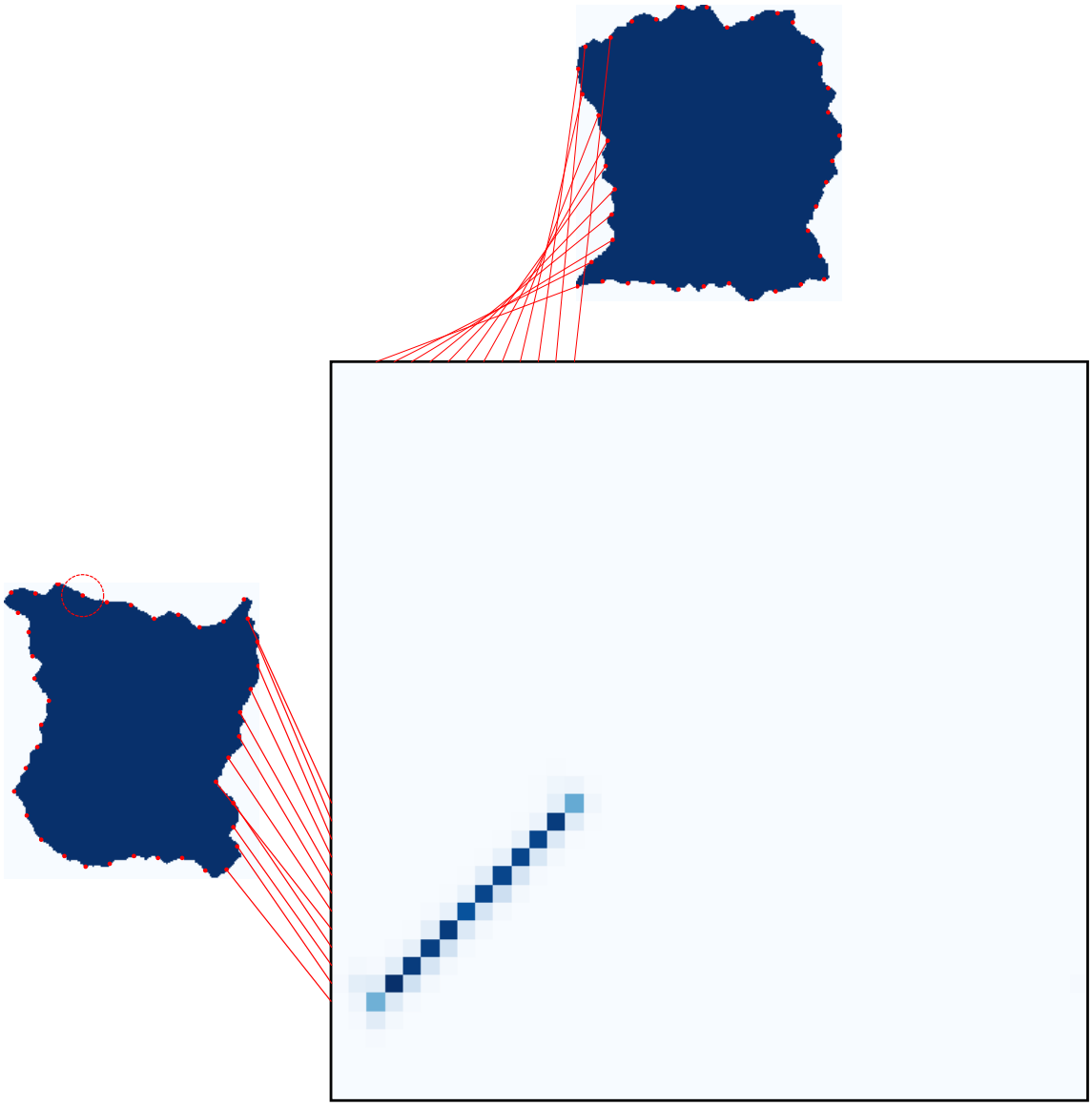


FIGURE 4.4 – Matrice d'adjacence idéale pour deux fragments adjacents. Les points rouges sur le contour des fragments ont été choisis avec un pas de 25 pixels et le cercle pointillé est le voisinage associé au point en son centre. La matrice d'adjacence est représentée sous forme de carrés colorés, où des entrées foncées correspondent à des nombres près de 1 et des entrées pâles, près de 0. Chaque ligne de la matrice correspond à un voisinage du premier fragment et chaque colonne correspond à un voisinage du deuxième fragment. La couleur à l'intersection d'une ligne et d'une colonne correspond au score d'adjacence entre les deux voisinages. Dans ce cas idéalisé, presque toutes les entrées de la matrice sont nulles, sauf pour une diagonale qui coïncide avec la frontière commune entre les deux fragments. Si les deux fragments n'étaient pas adjacents, on aurait souhaité que la matrice soit complètement nulle.

Pendant l'entraînement, les centres des voisinages extraits sont légèrement modifiés aléatoirement. Notons (x, y) le centre d'un voisinage. Le processus d'entraînement reçoit plutôt ces coordonnées sous la forme $(x+\Delta_x, y+\Delta_y)$. Les variables Δ_x et Δ_y sont des nombres aléatoires compris entre $-\delta$ et δ ($\delta \in \mathbb{N}$), ce qu'on note $\Delta_x, \Delta_y \sim \mathcal{U}\{-\delta, \delta\}$ (loi uniforme discrète), où δ est le déplacement aléatoire maximal. Le modèle apprend donc à reconnaître l'adjacence même si les morceaux ne sont pas parfaitement alignés, ce qui est rarement le cas en pratique. L'importance de cette procédure est détaillée dans le prochain chapitre à la sous-section 5.2.4.

Après chaque itération d'entraînement vient une itération de validation pour tester le modèle sur des données qu'il n'a jamais vues auparavant. Aucun apprentissage ne survient pendant la validation. Les résultats obtenus servent uniquement à calculer la métrique d'exactitude (sous-section 3.2.2). Comme c'est cette étape qui permet de juger de la qualité de l'apprentissage, il est souhaitable que le processus soit reproductible à chaque itération et, donc, déterministe. C'est pourquoi on ne déplace pas aléatoirement les voisinages pendant la validation.

4.2.3 Architecture du réseau

Comme énoncé dans les chapitres précédents, nous utilisons une architecture invariante aux rotations puisque les fragments peuvent avoir n'importe quelle orientation. Une implémentation de cette architecture est disponible dans la bibliothèque logicielle Python `e2cnn` de Weiler et Cesa [55]. Le choix des couches (figure 4.5) est aussi fortement inspiré de leurs travaux.

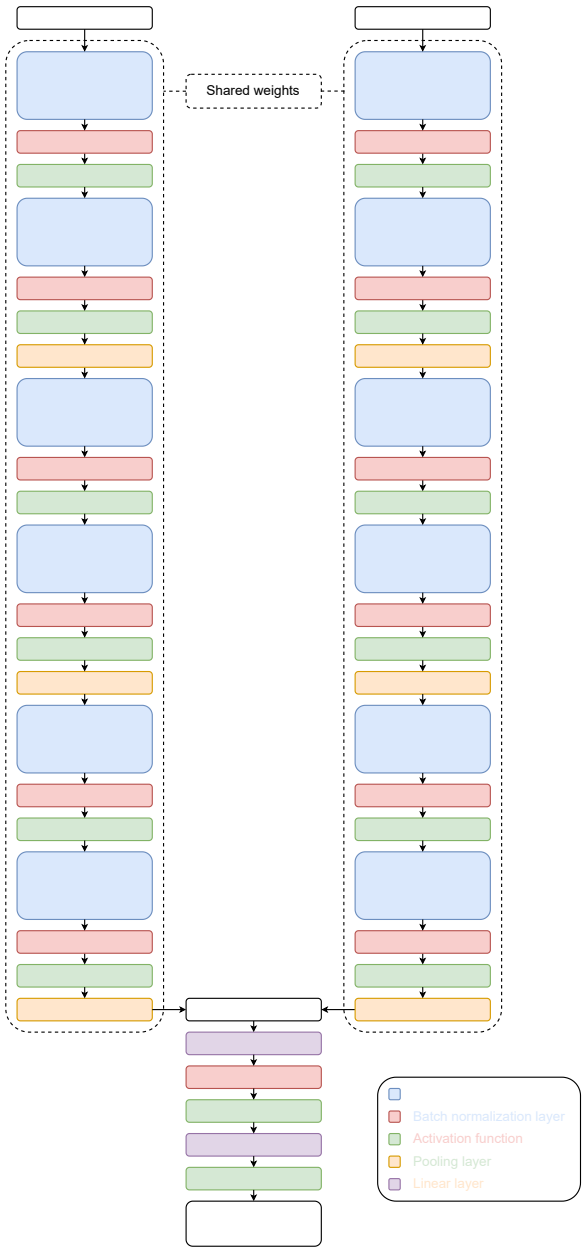


FIGURE 4.5 – L'architecture du CNN classificateur binaire siamois et équivariant aux rotations. Elle est fortement inspirée des travaux de Weiler et Cesa [55]. Ce modèle est siamois, car il possède deux branches aux poids partagés pour recevoir les voisinages des deux fragments à comparer. Les vecteurs résultants de ces deux branches sont ensuite concaténés et envoyés dans un MLP. La sortie finale passe par la fonction d'activation sigmoïde, comme il est commun de le faire pour un classificateur binaire. C_N est le groupe cyclique de rotations par rapport auquel le réseau est équivariant (sous-section 3.3.6). L'effet du choix du groupe cyclique (par exemple, C_4, C_8...) est évalué dans les expérimentations de la sous-section 5.1.2.

4.2.4 Entraînement

Étant donné que `e2cnn` est basée sur la bibliothèque logicielle `PyTorch`, le reste du script d'entraînement utilise également `PyTorch` [56]. L'optimiseur Adam est utilisé (sous-section 3.2.4). Les valeurs des hyperparamètres, notamment le nombre d'itérations (*epochs*), la taille des mini-lots (*batch size*) et le taux d'apprentissage (*learning rate*), sont précisées selon l'expérimentation.

4.3 Prédiction de l'adjacence par classification des matrices d'adjacence

La figure 4.4 montre qu'une matrice d'adjacence de deux fragments adjacents est censée comporter une diagonale de valeurs s'approchant de 1.

Il devient donc possible d'entraîner un second modèle, un traditionnel CNN classificateur binaire, capable de différencier des matrices d'adjacence de fragments adjacents (comportant théoriquement des diagonales) de matrices d'adjacence de fragments non adjacents. Il faut toutefois avoir préalablement entraîné un modèle capable de générer des matrices d'adjacence fiables. C'est ce qui a été présenté à la section précédente.

En résumé, le premier modèle construit une matrice d'adjacence en évaluant localement l'adjacence à partir de paires de voisinages, puis le deuxième prend la décision finale (la paire est adjacente ou non) en classifiant la matrice d'adjacence résultante.

Un modèle utilisé à la première étape (évaluation de voisinages) est appelé **modèle local** tandis qu'un modèle classifiant des matrices d'adjacence est appelé **modèle global**.

4.3.1 Données

Une fois qu'un modèle local est suffisamment précis pour prédire l'adjacence de voisinages, on peut l'utiliser pour générer la matrice d'adjacence complète de deux fragments. Pour reprendre la notation présentée à la sous-section 4.2.1, il faut évaluer $L_1 \times L_2$ paires de voisinages pour remplir la matrice. Rappelons que L_1 (resp. L_2) est le nombre de voisinages à évaluer sur le contour du premier fragment (resp. le contour du second fragment).

Toutefois, en pratique, l'évaluation de toutes ces paires est longue et n'est pas essentielle. Nous utilisons plutôt un échantillonnage pour réduire le nombre de voisinages à évaluer. Par exemple, si on choisit un pas de 3 entre chaque pixel, on évalue le voisinage autour du 1$^{\text{er}}$ point du contour, on ignore les 2 points suivants, puis on évalue autour du 4$^{\text{e}}$ point, etc. Comme il n'y a que 3 pixels entre chaque voisinage évalué, l'impact sur la précision est faible, mais le nombre d'éléments de la matrice est réduit d'un facteur de 9 : les dimensions de A deviennent $\left\lceil \frac{L_1}{3} \right\rceil \times \left\lceil \frac{L_2}{3} \right\rceil$. L'effet de la grandeur du pas est étudié dans les expérimentations à la sous-section 5.2.3.

De plus, comme L_1 et L_2 ne sont pas égaux et varient pour chaque paire de fragments, les matrices d'adjacence ne sont pas carrées et n'ont pas des dimensions fixes. Le modèle global doit être conçu en conséquence.

L'entraînement d'un modèle global dépend de l'entraînement préalable d'un modèle local. Pour générer les jeux de données d'entraînement et de validation du modèle global, on transforme les jeux de données ayant servi à l'entraînement et à la validation du modèle local en matrices d'adjacence à l'aide de ce même modèle local. On obtient ainsi un grand nombre de matrices d'adjacence, la moitié issue de paires adjacentes, l'autre moitié issue de paires non adjacentes.

4.3.2 Architecture du réseau

Tel qu'énoncé précédemment, le modèle global doit être un CNN classificateur binaire. La bibliothèque logicielle `timm` [57] propose plusieurs architectures populaires dans la littérature [58], dont celles de la famille ResNet [59]. Les variantes les plus simples, ResNet18 et ResNet34, sont testées dans les expérimentations à la sous-section 5.2.2.

Notons que ces modèles ResNet contiennent une couche *max pool* à la fin des couches convolutives de leur encodeur (éq. 3.24). Étant donné que chaque couche convolutive réduit la hauteur (H') et la largeur (W') du tenseur et augmente le nombre de canaux (C'), le tenseur produit par l'encodeur possède une faible hauteur, une faible largeur, mais un grand nombre de canaux (définition 3.21). La couche *max pool* vient réduire la largeur et la hauteur du tenseur à 1 en calculant la valeur maximale de chaque matrice $H' \times W'$ composant le tenseur de dimensions $H' \times W' \times C'$. Le tenseur obtenu est donc de taille fixe ($1 \times 1 \times C'$), et ce, malgré les dimensions $L_1 \times L_2$ variables des matrices d'adjacence.

4.3.3 Entraînement

Comme pour l'implémentation du modèle local, le script d'entraînement se base sur `PyTorch` [56], l'optimiseur Adam est utilisé [45] et les différents hyperparamètres (le nombre d'itérations, la taille des mini-lots et le taux d'apprentissage) sont précisés dans les expérimentations.

4.3.4 Conclusion

Ce chapitre a introduit les algorithmes utilisés pour générer les jeux de données et notre approche composée de deux modèles :

— un modèle local équivariant aux rotations qui génère les matrices d'adjacence ;

— un modèle global qui classifie les matrices d'adjacence.

Le chapitre qui suit teste ces modèles afin de trouver la combinaison d'hyperparamètres produisant les meilleurs résultats.

Chapitre 5

Résultats et analyses des expérimentations

Les expérimentations ont été exécutées sur un ordinateur équipé d'une carte graphique NVIDIA de type *GeForce RTX 2080 TI*. Les scripts utilisent Python 3.7.7 et les bibliothèques logicielles suivantes :

— `PyTorch` [56], `e2cnn` [55] et `timm` [57] pour les réseaux de neurones profonds ;

— `wandb` [60] pour le suivi des résultats des expérimentations ;

— l'écosystème `Hugging Face` (`datasets` [61], `accelerate` [62], `evaluate`) pour la gestion des jeux de données.

5.1 Prédiction de matrices d'adjacence

Lorsque ce n'est pas précisé, toutes les expérimentations de cette section utilisent un taux d'apprentissage de 5×10^{-4}, 15 itérations, une taille de mini-lots de 32, le groupe cyclique de rotations C_8 et un déplacement aléatoire maximal du centre des

voisinages (δ) de 3. Les jeux de données sont composés de fragments provenant de carrés monochromes déchirés en 16 morceaux. Il s'agit d'un ensemble d'hyperparamètres de départ qui, lors de nos essais aléatoires, ont constamment produit un bon entraînement.

De plus, certaines expérimentations avec un plus grand degré de variabilité ont été répétées à quelques reprises avec des germes aléatoires (*random seeds*) différents. Il a été démontré qu'ils peuvent influencer, dans une certaine mesure, les résultats obtenus [63].

5.1.1 Analyse de l'effet du nombre de paires constituant le jeu de données d'entraînement

La figure 5.1 présente l'exactitude (*accuracy*) maximale atteinte sur le jeu de données de validation en fonction du nombre de paires qui composent le jeu de données d'entraînement. On constate que la variance est plus élevée lorsque le jeu de données est plus petit. La variance se stabilise à partir de 2 400, où le gain en exactitude est ensuite faible. Pour la suite des entraînements, nous utilisons donc un jeu de données d'entraînement composé de 2 400 paires.

5.1.2 Analyse de l'effet du groupe cyclique utilisé par les couches convolutives équivariantes aux rotations

L'exactitude de validation maximale a été mesurée suite à un entraînement pour les 4 groupes cycliques de rotations suivants : C_1 (aucune équivariance), C_4, C_8 et C_{16}. La figure 5.2 montre qu'il y a un grand gain entre C_1 et C_4 (près de 25 %). L'augmentation de l'exactitude est tout de même importante entre C_4 et C_8 (environ

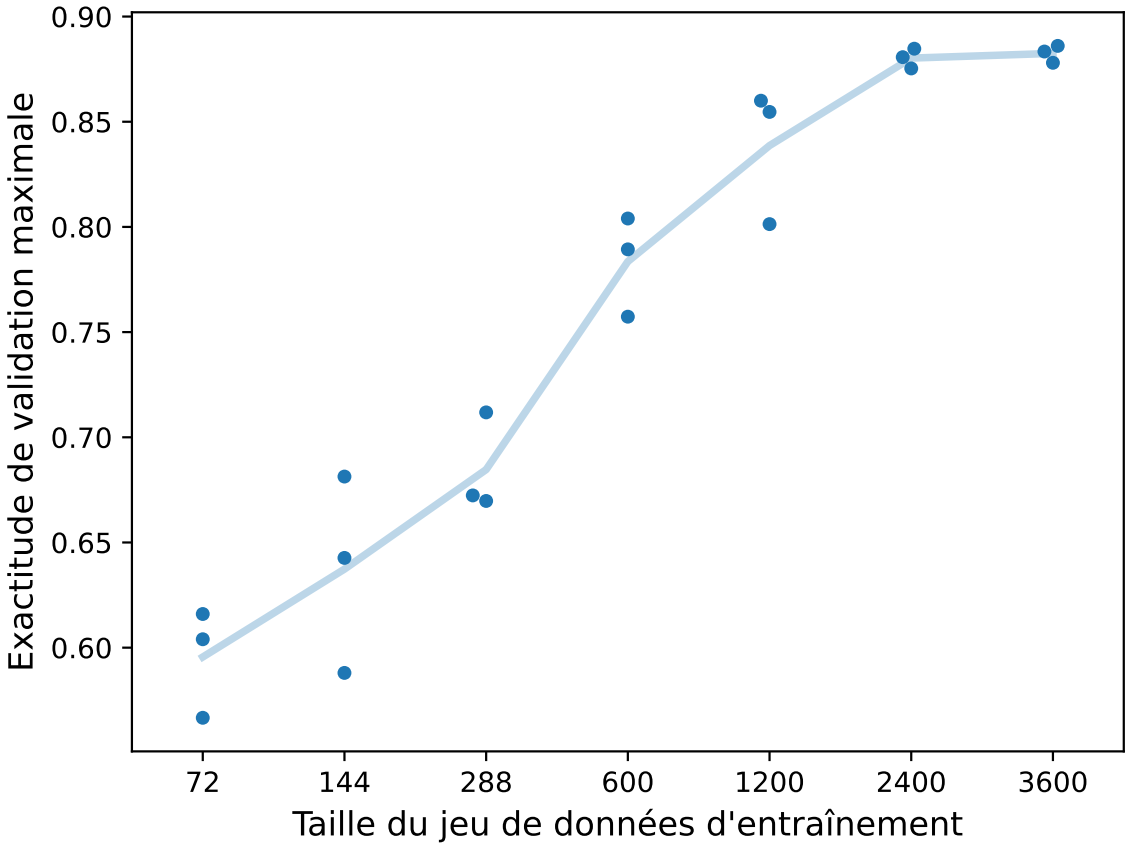


FIGURE 5.1 – L'exactitude maximale atteinte sur le jeu de données de validation en fonction de la taille (en nombre de paires) qui compose le jeu de données d'entraînement.

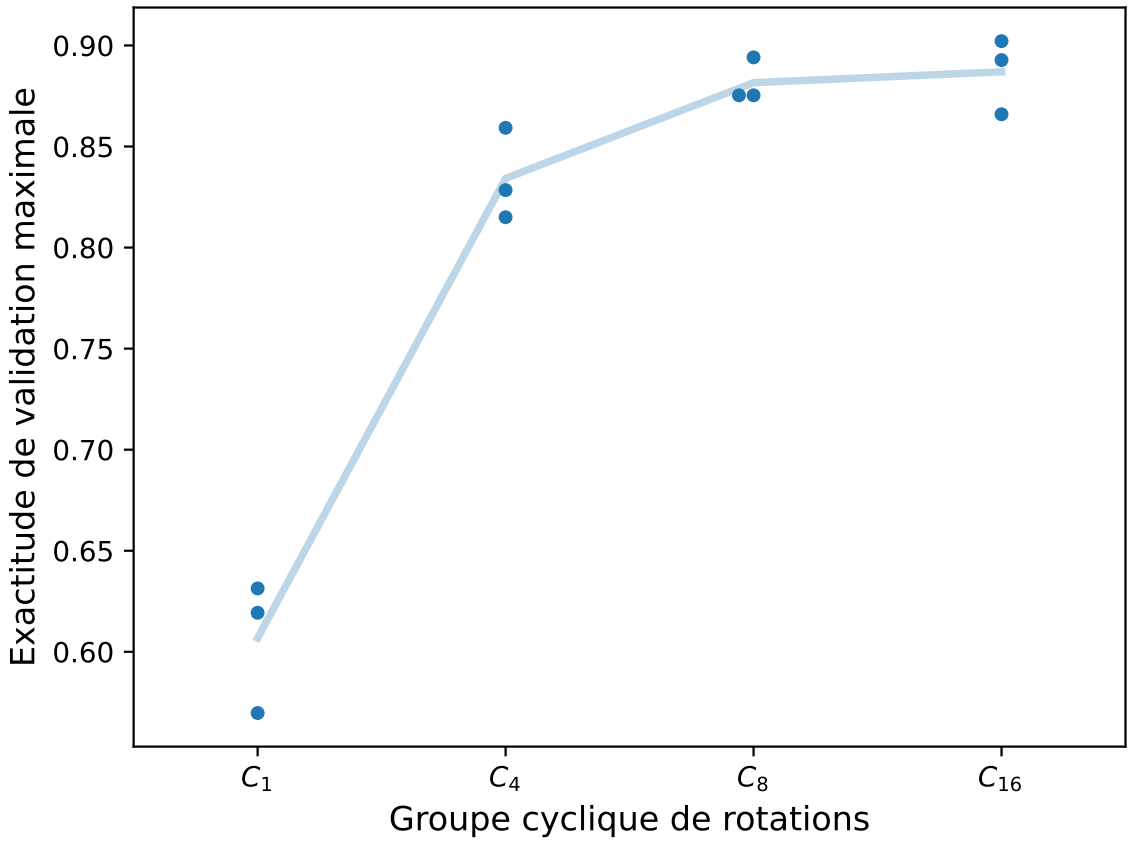


FIGURE 5.2 – L'exactitude maximale atteinte sur le jeu de données de validation en fonction du groupe cyclique de rotations utilisé par les couches convolutives équivariantes aux rotations.

5 %), mais elle est presque nulle entre C_8 et C_{16}. Pour toutes les expérimentations suivantes, nous continuons donc d'utiliser C_8.

5.2 Classification des matrices d'adjacence

Pour générer ce jeu de données, des modèles de la section précédente ont été utilisés sur les données ayant servi à leur entraînement. Cela a permis de générer les matrices d'adjacence pour les classifier.

Lorsque ce n'est pas précisé, toutes les expérimentations de cette section utilisent

Architecture	**Préentraîné**	**Exactitude maximale de validation (%)**
ResNet18	Non	72,32
ResNet18	Oui	72,45
ResNet34	Non	72,45
ResNet34	Oui	72,45

TABLE 5.1 – Exactitude maximale de validation en fonction de l'architecture de classificateur binaire.

un taux d'apprentissage de 10^{-3}, 30 itérations et une taille de mini-lots de 64. Comme à la section précédente, nous avons choisi ces hyperparamètres comme point de départ à la suite d'essais aléatoires.

5.2.1 Analyse de l'effet de la taille du voisinage

La figure 5.3 affiche clairement le lien entre l'exactitude du modèle et la taille du voisinage : un plus petit voisinage entraîne de meilleurs résultats. Puisqu'un plus petit voisinage contient moins d'informations, le modèle peut plus facilement se concentrer sur le centre, la zone d'intérêt, et est moins « distrait » par ce qui se trouve en périphérie.

Les expérimentations suivantes utilisent toutes une taille de voisinage de 29 pixels.

5.2.2 Analyse de l'effet du modèle

Comme classificateur binaire, les deux plus petits modèles de la famille ResNet sont évalués : ResNet18 et ResNet34 [59]. Nous testons également les versions préentraînées sur le jeu de données ImageNet.

On constate au tableau 5.1 que la différence entre les quatre essais n'est pas signi-

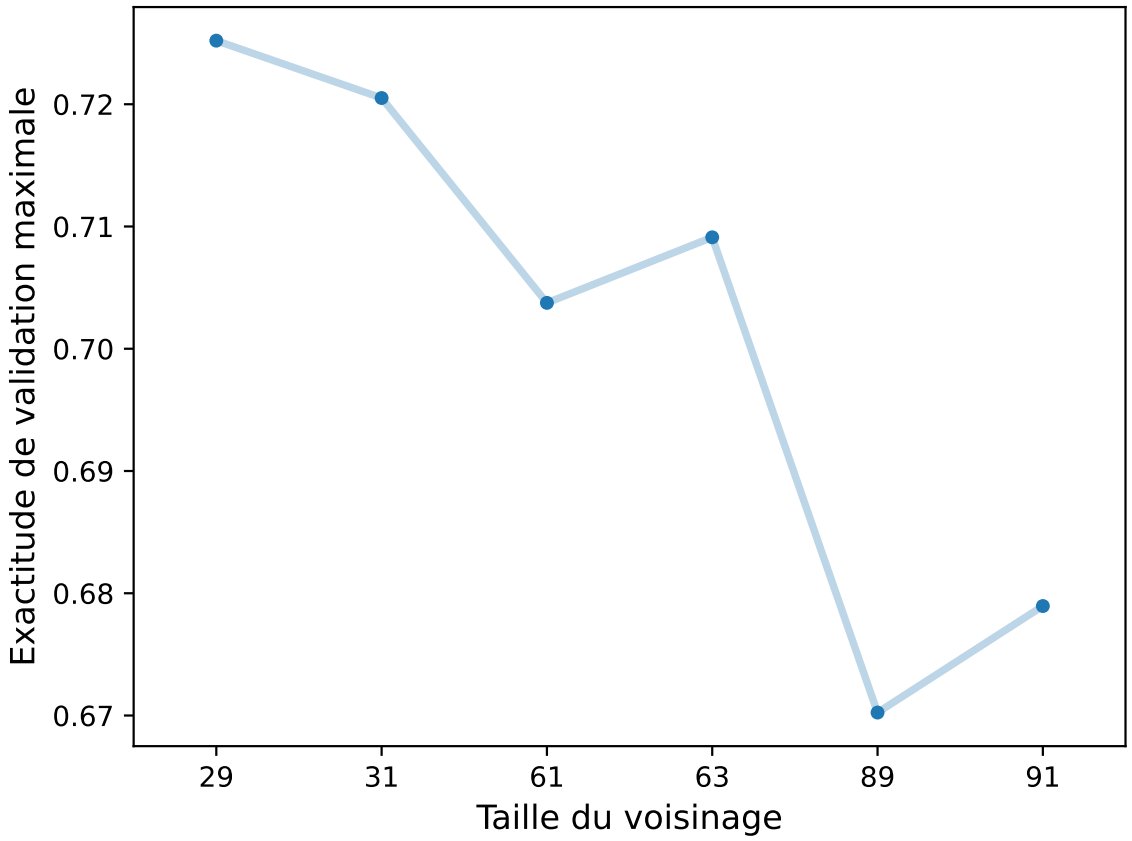


FIGURE 5.3 – L'exactitude maximale atteinte sur le jeu de données de validation en fonction de la taille du voisinage qu'utilise le modèle.

ficative. Pour beaucoup de tâches de classification d'images, l'utilisation d'un modèle préentraîné augmente significativement sa performance et diminue le temps requis pour apprendre [58]. Or, comme les matrices d'adjacence ne ressemblent pas à des images conventionnelles, l'effet du préentraînement n'est pas significatif.

L'architecture ResNet34, qui comporte plus de couches, ne procure aucun avantage significatif. Comme les matrices d'adjacence ne sont pas des images complexes, ResNet18 semble suffisant.

5.2.3 Analyse de l'effet de la grandeur du pas

Tel que discuté à la sous-section 4.3.1, cet hyperparamètre correspond au pas entre les voisinages évalués pour générer les matrices d'adjacence. Plus le pas est grand, plus la génération des matrices est rapide, mais moins il y a de données dans la matrice.

La figure 5.4 montre que l'exactitude diminue lorsque la grandeur du pas augmente. Toutefois, les différences entre un pas de 1, 3 ou 5 sont petites, alors que l'exactitude maximale diminue de 5 % pour un pas de 10. Considérant qu'ici, la taille du voisinage est de 29 pixels, un pas de 10 est trop grand. Nous conservons donc le pas de 3 : la perte d'exactitude est faible et cela permet d'évaluer 9 fois moins de voisinages pour générer les matrices d'adjacence.

5.2.4 Analyse de l'effet de la variation aléatoire de la position de la fenêtre pendant l'entraînement

Tel que présenté à la sous-section 4.2.2, le centre des voisinages est modifié aléatoirement pendant l'entraînement, où δ est le déplacement aléatoire maximal du centre

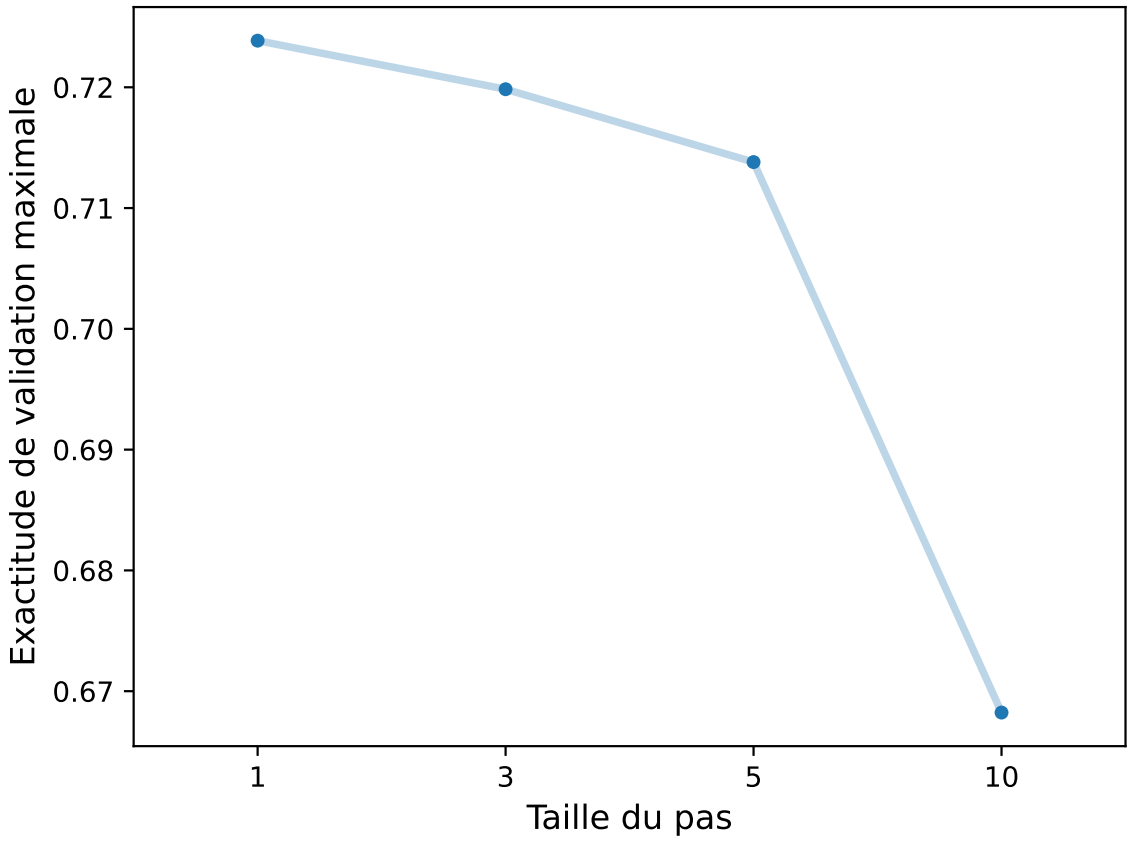
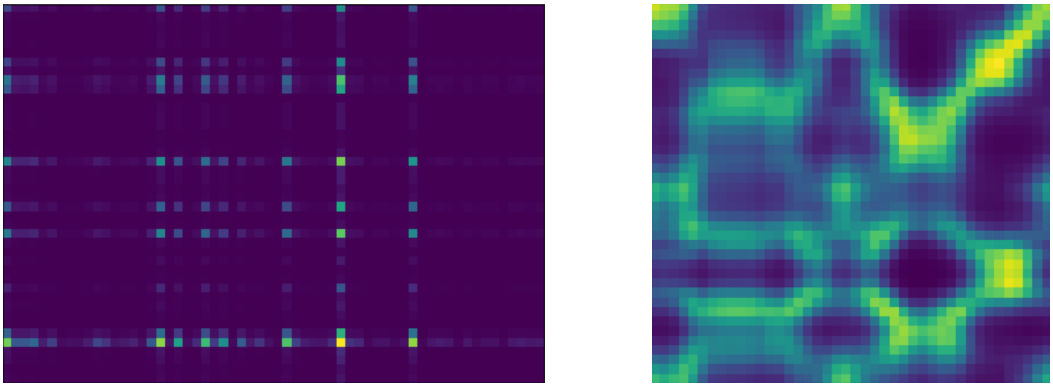


FIGURE 5.4 – L'exactitude maximale atteinte sur le jeu de données de validation en fonction du pas utilisé pour générer les matrices d'adjacence.



(a) Sans déplacement aléatoire pendant l'entraînement ($\delta = 0$).

(b) Avec déplacement aléatoire ($\delta = 3$).

FIGURE 5.5 – Analyse de l'effet de la variation aléatoire de la position de la fenêtre pendant l'entraînement.

des voisinages. Dans les expérimentations précédentes, $\delta = 3$ est utilisé. La figure 5.5a montre ce qui se produit lorsque $\delta = 0$. Comme le modèle a été habitué à ne voir que des voisinages parfaitement alignés, il n'est pas capable de générer de bonnes matrices d'adjacence, ce qui empêche tout apprentissage. On préfère des matrices d'adjacence comme à la figure 5.5b : une diagonale parfaite se trouve dans son coin supérieur droit, ce qui indique que les deux fragments qu'elle représente sont adjacents. C'est ce qui a permis d'obtenir tous les résultats de ce chapitre.

5.3 Conclusion

Nous avons analysé dans ce chapitre l'effet de plusieurs hyperparamètres. Pour le modèle local, le groupe cyclique de rotations C_8 procure un gain d'exactitude significatif par rapport à C_1 et C_4. Quant au modèle global, le modèle ResNet18 semble adéquat. Nos résultats montrent également qu'un petit voisinage amène un gain d'exactitude, qu'un pas trop grand nuit au modèle global et que la variation aléatoire de la position de la fenêtre pendant l'entraînement est essentielle. En somme, notre approche réussit à prédire l'adjacence avec une exactitude de validation maximale de 72,45 %.

Chapitre 6

Conclusion

Nous avons étudié dans ce mémoire une nouvelle approche pour prédire l'adjacence de fragments 2D. Notre méthodologie emploie deux types de réseaux de neurones. Le premier est un réseau convolutif siamois équivariant aux rotations utilisé pour générer les matrices d'adjacence entre des paires de fragments, tandis que le second est un réseau classificateur binaire recevant les matrices d'adjacence générées. Pour mener à bien l'entraînement, nous avons également introduit deux algorithmes pour générer des fragments. Notre méthode atteint 72,45 % d'exactitude sur un jeu de données composé de carrés monochromes fragmentés en 16 morceaux. Notre objectif initial est atteint : il est possible d'utiliser l'apprentissage profond pour déterminer si des paires de fragments 2D sont adjacentes, et ce, sans restreindre leur forme et leur orientation.

Dans de futurs travaux, il serait intéressant d'entraîner nos modèles sur des fragments comportant des couleurs (par exemple, de vrais artéfacts archéologiques). L'ajout de ces nouvelles informations devrait normalement aider les modèles à prédire l'adjacence avec une meilleure exactitude. De plus, l'acquisition de données réelles permettrait de tester notre méthodologie dans une situation d'application concrète et de valider son efficacité.

Finalement, la généralisation de notre approche pour les fragments 3D reste à explorer. Cela augmenterait son niveau d'applicabilité dans plusieurs domaines, notamment en archéologie, où l'épaisseur des tessons est un renseignement important dont il faut tenir compte pour mener à bien leur reconstruction. Cela devrait aussi améliorer l'exactitude des prédictions : en ayant accès aux fragments 3D, le modèle disposerait de davantage d'informations pour comparer les voisinages.

Bibliographie

[1] C. Ostertag et M. Beurton-Aimar, « Matching ostraca fragments using a siamese neural network », Pattern Recognition Letters, vol. 131, p. 336–340, mars 2020.

[2] L. Wei, W. Yu, M. Li et X. Li, « Skull assembly and completion using template-based surface matching », in 2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, IEEE, mai 2011.

[3] G. Papaioannou, T. Schreck, A. Andreadis, P. Mavridis, R. Gregor, I. Sipiran et K. Vardis, « From reassembly to object completion », Journal on Computing and Cultural Heritage, vol. 10, p. 1–22, avril 2017.

[4] T. M. Paixão, R. F. Berriel, M. C. Boeres, A. L. Koerich, C. Badue, A. F. D. Souza et T. Oliveira-Santos, « Self-supervised deep reconstruction of mixed strip-shredded text documents », Pattern Recognition, vol. 107, p. 107535, nov. 2020.

[5] D. Eslami, L. D. Angelo, P. D. Stefano et E. Guardiani, « A semi-automatic reconstruction of archaeological pottery fragments from 2D images using wavelet transformation », Heritage, vol. 4, p. 76–90, jan. 2021.

[6] A. Pirrone, M. B. Aimar et N. Journet, « Papy-s-net : A siamese network to match papyrus fragments », in Proceedings of the 5th International Workshop on Historical Document Imaging and Processing, HIP '19, (New York, NY, USA), p. 78–83, Association for Computing Machinery, 2019.

[7] M.-M. Paumard, D. Picard et H. Tabia, « Jigsaw puzzle solving using local feature co-occurrences in deep neural networks », in 2018 25th IEEE International Conference on Image Processing (ICIP), IEEE, oct. 2018.

[8] M. Noroozi et P. Favaro, « Unsupervised learning of visual representations by solving jigsaw puzzles », in Computer Vision – ECCV 2016, p. 69–84, Springer International Publishing, 2016.

[9] C. Le et X. Li, « JigsawNet : Shredded image reassembly using convolutional neural network and loop-based composition », IEEE Transactions on Image Processing, vol. 28, p. 4000–4015, août 2019.

[10] E. Justino, L. S. Oliveira et C. Freitas, « Reconstructing shredded documents through feature matching », Forensic Science International, vol. 160, p. 140–147, juil. 2006.

[11] M. A. Fischler et R. C. Bolles, « Random sample consensus », Communications of the ACM, vol. 24, p. 381–395, juin 1981.

[12] S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi, « Optimization by simulated annealing », Science, vol. 220, p. 671–680, mai 1983.

[13] P. Besl et N. D. McKay, « A method for registration of 3-d shapes », IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, p. 239–256, fév. 1992.

[14] T. Theoharis, C. Schellewald, P. Perakis, A. Andreadis, G. gios Papaioannou, M. S. Pavlos Mavridis, K. Sfikas, I. Pratikakis, F. Arnaoutoglou, I. Sipiran, R. Gregor, T. Schreck et D. Rieke-Zapp, « PRESIOUS Project – D5.4 First evaluation report », mars 2015.

[15] U. Ramer, « An iterative procedure for the polygonal approximation of plane curves », Computer Graphics and Image Processing, vol. 1, p. 244–256, nov. 1972.

[16] D. H. Douglas et T. K. Peucker, « Algorithms for the reduction of the number of points required to represent a digitized line or its caricature », Cartographica : The International Journal for Geographic Information and Geovisualization, vol. 10, p. 112–122, déc. 1973.

[17] K. Zhang et X. Li, « A graph-based optimization algorithm for fragmented image reassembly », Graphical Models, vol. 76, p. 484–495, sept. 2014.

[18] N. Derech, A. Tal et I. Shimshoni, « Solving archaeological puzzles », Pattern Recognition, vol. 119, p. 108065, nov. 2021.

[19] P. Dondi, L. Lombardi et A. Setti, « DAFNE : A dataset of fresco fragments for digital anastlylosis », Pattern Recognition Letters, vol. 138, p. 631–637, oct. 2020.

[20] P. Barra, S. Barra, M. Nappi et F. Narducci, « SAFFO : A SIFT based approach for digital anastylosis for fresco recOnstruction », Pattern Recognition Letters, vol. 138, p. 123–129, oct. 2020.

[21] D. T. Dimov, « Rotation-invariant NCC for 2d color matching of arbitrary shaped fragments of a fresco », Pattern Recognition Letters, vol. 138, p. 431–438, oct. 2020.

[22] N. Lermé, S. L. Hégarat-Mascle, B. Zhang et E. Aldea, « Fast and efficient reconstruction of digitized frescoes », Pattern Recognition Letters, vol. 138, p. 417–423, oct. 2020.

[23] J. Canny, « A computational approach to edge detection », IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, p. 679–698, nov. 1986.

[24] A. Haar, « Zur theorie der orthogonalen funktionensysteme », Mathematische Annalen, vol. 69, p. 331–371, sep 1910.

[25] I. Daubechies, Ten Lectures on Wavelets. Society for Industrial and Applied Mathematics, jan 1992.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard et L. D. Jackel, « Backpropagation Applied to Handwritten Zip Code Recognition », Neural Computation, vol. 1, p. 541–551, 12 1989.

[27] Y. LeCun, Y. Bengio et G. Hinton, « Deep learning », Nature, vol. 521, p. 436–444, may 2015.

[28] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger et R. Shah, « Signature Verification using a "Siamese" Time Delay Neural Network », International Journal of Pattern Recognition and Artificial Intelligence, vol. 07, p. 669–688, août 1993.

[29] C. Doersch, A. Gupta et A. A. Efros, « Unsupervised visual representation learning by context prediction », in 2015 IEEE International Conference on Computer Vision (ICCV), IEEE, déc. 2015.

[30] M.-M. Paumard, D. Picard et H. Tabia, « Deepzzle : Solving visual jigsaw puzzles with deep learning and shortest path optimization », IEEE Transactions on Image Processing, vol. 29, p. 3569–3581, 2020.

[31] C. Ostertag et M. Beurton-Aimar, « Using graph neural networks to reconstruct ancient documents », in Pattern Recognition. ICPR International Workshops and Challenges, p. 39–53, Springer International Publishing, 2021.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li et L. Fei-Fei, « ImageNet : A large-scale hierarchical image database », in 2009 IEEE Conference on Computer Vision and Pattern Recognition, p. 248–255, 2009.

[33] K. Musgrave, S. Belongie et S.-N. Lim, « A metric learning reality check », in Computer Vision – ECCV 2020, p. 681–699, Springer International Publishing, 2020.

[34] S. Chopra, R. Hadsell et Y. LeCun, « Learning a similarity metric discriminatively, with application to face verification », in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), IEEE, 2005.

[35] R. Hadsell, S. Chopra et Y. LeCun, « Dimensionality reduction by learning an invariant mapping », in 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06), IEEE, 2006.

[36] F. Schroff, D. Kalenichenko et J. Philbin, « FaceNet : A unified embedding for face recognition and clustering », in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, juin 2015.

[37] N. N. Vo et J. Hays, « Localizing and orienting street views using overhead imagery », in Computer Vision – ECCV 2016, p. 494–509, Springer International Publishing, 2016.

[38] S. Horiguchi, D. Ikami et K. Aizawa, « Significance of softmax-based features in comparison to distance metric learning-based features », IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 5, p. 1279–1285, 2020.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville et Y. Bengio, « Generative adversarial nets », in Advances in Neural Information Processing Systems (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence et K. Weinberger, éds), vol. 27, Curran Associates, Inc., 2014.

[40] A. Rafique, T. Iftikhar et N. Khan, « Adversarial placement vector learning », in 2019 2nd International Conference on Advancements in Computational Sciences (ICACS), IEEE, fév. 2019.

[41] R. Li, S. Liu, G. Wang, G. Liu et B. Zeng, « JigsawGAN : Auxiliary learning for solving jigsaw puzzles with generative adversarial networks », IEEE Transactions on Image Processing, vol. 31, p. 513–524, 2022.

[42] I. Goodfellow, Y. Bengio et A. Courville, Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[43] A. Zhang, Z. C. Lipton, M. Li et A. J. Smola, « Dive into deep learning », arXiv preprint arXiv :2106.11342, 2021.

[44] A. Ng, « Standard notations for deep learning ».

[45] D. P. Kingma et J. Ba, « Adam : A method for stochastic optimization », in ICLR (Poster), 2015.

[46] X. Glorot, A. Bordes et Y. Bengio, « Deep sparse rectifier neural networks », in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (G. Gordon, D. Dunson et M. Dudík, éds), vol. 15in Proceedings of Machine Learning Research, (Fort Lauderdale, FL, USA), p. 315–323, PMLR, 11–13 Apr 2011.

[47] D. C. Cireşan, U. Meier, L. M. Gambardella et J. Schmidhuber, « Deep, Big, Simple Neural Nets for Handwritten Digit Recognition », Neural Computation, vol. 22, p. 3207–3220, 12 2010.

[48] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano et K. Lang, « Phoneme recognition using time-delay neural networks », IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, no. 3, p. 328–339, 1989.

[49] S. Kornblith, J. Shlens et Q. V. Le, « Do better ImageNet models transfer better ? », in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), p. 2656–2666, IEEE, 2019.

[50] T. Cohen et M. Welling, « Group equivariant convolutional networks », in Proceedings of The 33rd International Conference on Machine Learning (M. F. Balcan et K. Q. Weinberger, éds), vol. 48, (New York, New York, USA), p. 2990–2999, PMLR, 20–22 Jun 2016.

[51] T. S. Cohen, Equivariant convolutional networks. Thèse doctorat, University of Amsterdam, 2021.

[52] M. W. Lafarge, E. J. Bekkers, J. P. Pluim, R. Duits et M. Veta, « Roto-translation equivariant convolutional networks : Application to histopathology image analysis », Medical Image Analysis, vol. 68, p. 101849, feb 2021.

[53] R. N. Bracewell, Fourier Transform and Its Applications. Maidenhead, England : McGraw Hill Higher Education, 3 éd., nov. 1999.

[54] K. Perlin, « An image synthesizer », ACM SIGGRAPH Computer Graphics, vol. 19, p. 287–296, juil. 1985.

[55] M. Weiler et G. Cesa, « General E(2)-equivariant steerable CNNs », in Advances in Neural Information Processing Systems (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éds), vol. 32, Curran Associates, Inc., 2019.

[56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf,

E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai et S. Chintala, « Pytorch : An imperative style, high-performance deep learning library », in Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éds), p. 8024–8035, Curran Associates, Inc., 2019.

[57] R. Wightman, « Pytorch image models ». https://github.com/rwightman/pytorch-image-models, 2019.

[58] J. Howard et S. Gugger, Deep learning for coders with fastai and PyTorch : AI applications without a PhD. O'Reilly Media, Inc., 2020.

[59] K. He, X. Zhang, S. Ren et J. Sun, « Deep residual learning for image recognition », in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, juin 2016.

[60] L. Biewald, « Experiment tracking with weights and biases », 2020. Software available from wandb.com.

[61] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush et T. Wolf, « Datasets : A community library for natural language processing », in Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing : System Demonstrations, (Online and Punta Cana, Dominican Republic), p. 175–184, Association for Computational Linguistics, nov. 2021.

[62] S. Gugger, L. Debut, T. Wolf, P. Schmid, Z. Mueller et S. Mangrulkar, « Accelerate : Training and inference at scale made simple, efficient and adaptable. ». https://github.com/huggingface/accelerate, 2022.

[63] D. Picard, « Torch.manual_seed(3407) is all you need : On the influence of random seeds in deep learning architectures for computer vision », 2021.