# Segmentation d'images de microscopie cellulaire par apprentissage profond dans des contextes très peu supervisés

Arnaud Deleruyelle

**Université de Lille**

**THÈSE**

Pour l'obtention du grade de

**DOCTEUR**

Spécialité : **Automatique, Génie informatique, Traitement du Signal et des Images**

Présentée par

**Arnaud Deleruyelle**

---

# Segmentation d'images de microscopie cellulaire par apprentissage profond dans des contextes très peu supervisés

---

Soutenue le 16 novembre 2022 devant le jury d'examen :

| | | |
|---|---|---|
| **Présidente** | Caroline Petitjean | PR, Université de Rouen Normandie |
| **Rapporteur** | Alexandre Benoit | PR, Université Savoie Mont Blanc |
| **Rapporteur** | Karteek Alahari | CR Inria (HDR), Inria Grenoble |
| **Examinatrice** | Aurélie Bugeau | PR, Université de Bordeaux |
| **Examinatrice** | Émilie Kaufmann | CR CNRS, Inria Lille - Nord Europe |
| **Examinateur** | Jean Martinet | PR, Université Côte d'Azur |
| **Directeur de thèse** | John Klein | MCF (HDR), Université de Lille |
| **Co-Encadrant** | Cristian Versari | MCF, Université de Lille |

Thèse préparée au Centre de Recherche en Informatique Signal et Automatique de Lille
CRIStAL, UMR CNRS 9189

Université de Lille

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

# *Remerciements*

Une thèse ne se résume pas qu'à la rédaction d'un manuscrit, c'est également une expérience humaine enrichissante. Ces quelques lignes qui vont suivre ne suffiront probablement pas à exprimer toute la gratitude que j'éprouve envers les personnes m'ayant aidé dans ces travaux.

J'aimerais tout d'abord remercier mon directeur de thèse, John Klein, pour m'avoir offert l'opportunité de mener à bien ces travaux de recherche. Merci John pour ton professionnalisme, ta rigueur, ta patience, ta positivité, ta pédagogie, merci d'avoir gardé le cap lorsque nous prenions du retard et aussi de ton aide précieuse sur la toute dernière ligne droite. Je remercie mon co-encadrant Cristian Versari pour tous nos échanges passionnants sur l'apprentissage profond ainsi que des merveilleuses idées qui en découlent.

Je souhaite à présent remercier le jury qui a accepté d'évaluer mes travaux. Merci à mes deux rapporteurs Karteek Alahari et Alexandre Benoit pour les remarques constructives, mais également car vous m'avez permis d'approfondir mon sujet, et d'y apporter des pistes d'évolution. Je m'adresse également aux autres membres du jury qui ont permis à travers la soutenance, de mettre en lumière les limites du cadre applicatif de mes stratégies ainsi que de proposer des améliorations potentielles. Merci Jean Martinet d'avoir pris le temps d'apporter des corrections sur mon manuscrit encore après la soutenance.

Je me tourne ensuite vers mes deux équipes de recherche SIGMA et BioComputing dont je ne pourrais citer tout le monde, tellement j'ai pu croiser de personnes formidables. Je remercie Kévin, Rony et Martin avec qui j'ai eu le plaisir de partager le même bureau, et de passer de très bons moments. Je remercie Émilie qui m'a permis de me remotiver quand le moral n'y était pas. Je remercie Quentin, Jérémie, Patrick, Pierre, Cédric et Maxime pour leur enthousiasme, leur sympathie et les échanges enrichissants que nous avons eu. Je remercie également le reste des équipes qui m'ont permis de m'intégrer aisément. J'ai eu la chance d'évoluer dans un cadre stable avec un entourage bienveillant.

Il est impossible pour moi d'oublier de parler de Rui. J'ai eu l'immense plaisir de te connaître au début de ma thèse. Merci pour les moments de décompression, les rires, les joies mais aussi les précieuses aides. Merci également d'avoir été tolérant, notamment quand je n'étais pas très assidu sur les séances de sport !

Merci à Nicolas. Je me souviens de notre rencontre lors du 1er cours à l'Université. Nous ne savions absolument pas ce que nous faisions là, et maintenant nous sommes tous deux docteurs 9 ans après. J'ai vraiment eu beaucoup de chances de connaître un merveilleux ami, merci d'être resté fidèle à toi-même après toutes ces années.

Merci Vivien pour tous nos moments passés à échanger sur notre passion commune. Il est rare de garder des amis du lycée, surtout quand on peut toujours autant se divertir sur les mêmes sujets.

Merci à ma famille qui m'a toujours encouragé dans mes études. Merci à Isabelle et Éric pour votre amour, pour avoir cru en moi jusqu'au bout et de m'avoir donné un cadre idéal pour me dépasser. Merci Grégory pour m'avoir partagé le goût pour l'informatique. Ce n'est pas anodin si les trois frères travaillent désormais dans ce domaine. Merci Maxime pour tous les moments de détente, souvent remplis de nostalgie. Je n'oublie pas Jo, Alya, Laurence, Sophie, Liliane et Pierre pour votre soutien tout au long de mon doctorat.

Pour terminer, je m'adresse à ma chérie Maryline, merci pour ton amour inconditionnel. Merci d'avoir apaisé mes moments de doutes et de m'avoir supporté, notamment lorsque je devais travailler au milieu de la nuit. Tu as toujours été là pour moi durant toutes ces années, merci pour tout.

# Table des matières

# Table des figures

# Liste des tableaux

# Résumé

La segmentation est un domaine du traitement d'images qui consiste à partitionner une image en différentes régions correspondant à des objets d'intérêts dans l'image. Les méthodes les plus performantes en la matière s'appuient sur l'apprentissage profond par réseau de neurones et aboutissent à des résultats remarquables avec de nombreuses applications possibles en médecine ou biologie par exemple.
Pour obtenir ce niveau de performance, l'apprentissage profond se nourrit de vastes bases d'images dites annotées, c'est à dire qu'elles contiennent non pas seulement l'image devant subir la segmentation mais aussi le résultat attendu sous forme de masque binaire pour la segmentation sémantique qui est l'objet de ces travaux de thèse. Ceci présente un défaut majeur dans de nombreuses tâches de segmentation où l'obtention des annotations est particulièrement pénible à obtenir. Il convient le plus souvent de demander à un expert d'effectuer un contourage manuel des régions à segmenter. Il n'est pas raisonnable d'espérer construire de très grands jeux de données entièrement annotés pour chaque tâche de segmentation.
La problématique des travaux de thèse développés dans ce manuscrit s'attache à trouver des mécanismes permettant d'entraîner efficacement un réseau de neurones à segmenter quand la proportion d'images annotées est très faible (de l'ordre d'une dizaine d'instances voire moins). Une première contribution proposée s'appuie sur le principe de l'augmentation de données, i.e. la construction artificielle de nouveaux exemples annotés. L'approche proposée permet de choisir directement au fur et à mesure de l'apprentissage du réseau, les mécanismes d'augmentation les plus profitables, et ce pour un surcoût calculatoire très modeste. La seconde contribution déploie une chaîne de traitements composée de deux réseaux de neurones supplémentaires et s'appuie sur de l'auto-supervision pour pallier la faible quantité de données annotées. Le second réseau de la chaîne, dit professeur, a pour but de corriger la sortie du réseau cible (celui censé effectuer la segmentation). Le troisième réseau a pour objectif de reconstruire l'image d'entrée à partir de la sortie du professeur, et ce afin d'éviter d'apprendre des fonctions triviales au faible pouvoir généralisant.

# Abstract

Segmentation is a field of image processing that consists in partitioning an image into regions so that the regions correspond to certain objects of interest in the image. The most efficient methods in this field are based on deep learning by neural network and lead to remarkable results with many possible applications in medicine or biology for example.
To obtain this level of performances, deep learning feeds on vast databases of annotated images, i.e. they contain not only the image to be segmented but also the expected result in the form of a binary mask for semantic segmentation which is the subject of this thesis. This is a major flaw as in many segmentation tasks, obtaining the annotations is particularly tedious. It is most often necessary to ask an expert to manually delineate the regions to be segmented. It is not thus reasonable to expect to build very large datasets fully annotated for each segmentation task.
The problematic of the thesis work developed in this manuscript is to find mechanisms to efficiently train a neural network to segment when the proportion of images with annotations is very low (of the order of a dozen instances or less). A first contribution is based on the principle of data augmentation, i.e. the artificial construction of new annotated examples. The proposed approach allows to directly choose the most profitable augmentation mechanisms as the network is learned, at a very low computational cost. The second contribution deploys a pipeline composed of two additional neural networks and relies on self-supervision to compensate for the small amount of annotated data. The second network of the chain, called referee, aims at correcting the output of the target network (the one supposed to perform the segmentation). The third network aims at reconstructing the input image from the referee's output, in order to avoid learning trivial functions with low generalization power.

# Chapitre 1

# Introduction

## 1.1 Contexte

L'intelligence artificielle (IA) est un champ d'étude qui occupe l'esprit d'un grand nombre de chercheurs, particulièrement depuis le 20ème siècle. Est-il possible de simuler l'intelligence humaine ? Quelles sont les limites de cette intelligence, pouvons-nous la définir ? À l'heure d'aujourd'hui, la recherche sur ce sujet a pris une ampleur sans précédent. Des machines peuvent dès à présent effectuer des tâches d'une grande complexité, comme AlphaFold [61], un modèle conçu par DeepMind et capable de prédire la forme 3D de protéines, donnant accès à des informations et des possibilités jusqu'alors inaccessibles. Il s'agit là d'un problème dont la complexité dépasse largement la capacité humaine à modéliser et mémoriser.
Paradoxalement, une machine est capable de battre aisément un humain sur des tâches jugées complexes comme celle évoquée ci-dessus, mais des tâches plus abordables comme la traduction d'un texte mettent encore en difficulté même les IA les plus évoluées. Des progrès impressionnants en la matière ont tout de même été réalisés récemment grâce aux modèles de type *transformer* [28, 38], mais la difficulté du problème d'un point de vue informatique peut s'expliquer du fait que la traduction d'un texte demande des connaissances intuitives que l'humain possède, généralement grâce à un apprentissage sur le long terme ainsi qu'une capacité à créer de vastes chaînes de connaissances. L'enjeu pour la machine est donc d'être capable de formaliser, construire et d'incorporer cette base de connaissances. La vision par ordinateur est sûrement l'un des exemples les plus troublants. Un humain sait nativement voir, et identifier différents objets qui composent son champ visuel. La vision stéréoscopique de l'humain lui permet également d'apercevoir son environnement en 3D et de se le représenter mentalement. De cette manière, il ne se contente pas uniquement de reconnaître la forme des objets qui l'entourent, mais il peut en identifier la taille, ainsi que leurs éloignements respectifs. L'humain possède donc un degré d'analyse et de mise en contexte bien supérieur à celui des modèles classiquement utilisés en IA. C'est aussi pour cela que la communauté s'accorde pour le moment à dire que parler d'"intelligence" pour une machine reste très prématuré puisqu'elle ne comprend pas les concepts qu'elle manipule. Les travaux de thèse qui font l'objet de ce mémoire s'inscrivent dans la thématique de l'analyse d'images et de la vision par ordinateur par apprentissage artificiel. Un constat frappant qui a motivé en partie ces travaux est la différence considérable en terme de supervision nécessaire pour entraîner une machine comparativement à la quantité

nécessaire pour un humain, afin de résoudre des taches relatives à l'analyse visuelle.
Les compétences innées à l'humain sont exceptionnelles et il n'est pas rare de voir les chercheurs en IA s'inspirer des neurosciences pour créer leurs modèles. On peut notamment citer les réseaux de neurones ainsi que la variante impulsionnelle [128], ou bien la récompense des modèles d'apprentissage par différence temporelle en partie inspirée de la dopamine [89]. De nombreux travaux s'inspirent du système de vision humain [7], mais il n'est cependant pas réaliste de pouvoir obtenir un modèle entièrement basé sur celui-ci. En effet, le cerveau utilise plus de la moitié de ses fonctionnalités afin de construire une image à partir des signaux nerveux envoyés par la rétine. Pouvoir formaliser informatiquement une tel procédé est pour le moment hors d'atteinte.

Les stratégies utilisées en vision par ordinateur consistent principalement à obtenir un maximum d'informations à partir de la scène observée. A l'instar de la voiture autonome [126], un algorithme de décision est ensuite appliqué à partir de la base d'informations collectées. Par information, on peut évoquer le fait de récupérer les différents plans de l'image, mais également d'extraire le contour ainsi que la forme de chaque objet présent. Une dernière étape consistant à attribuer une étiquette sur les différents objets peut-être effectuée. L'opération permettant de pouvoir identifier l'emplacement exact d'un objet dans une image se nomme la **segmentation** et fait l'objet des travaux présentés dans ce manuscrit. Comme évoqué plus haut, l'ambition de ces travaux sera essentiellement dirigée vers des solutions à base d'apprentissage automatique pouvant être entraînées à partir d'une supervision très réduite.

## 1.2 Cadre de la segmentation

La segmentation consiste à partitionner une image en différentes régions. Il est alors possible de repérer l'emplacement exact d'un objet dans l'image. La nature des éléments à segmenter peut varier et ne doit pas nécessairement représenter une région homogène dans l'image. Il est par exemple possible de segmenter un humain (pourtant constitué de différentes régions, comme les cheveux, la peau ou les vêtements) dans une image de photographie. Les cadres d'utilisation peuvent également être différents. Dans certains cas, l'objectif peut consister à séparer le fond de l'avant-plan d'une image. La segmentation d'un objet mobile dans une séquence vidéo est très populaire car elle permet de suivre la trajectoire précise de l'objet d'intérêt pour entre autres étudier son comportement et ses propriétés.

Les travaux présentés vont se concentrer principalement sur un cadre applicatif très spécifique, la segmentation d'images biomédicales. Les utilisations possibles de tels procédés sont nombreux, allant de la détection automatique de tumeurs [41], de malformations, de lésions ou de tissus malades. La figure 1.1 montre un exemple de segmentation sur le jeu de données *Digital*

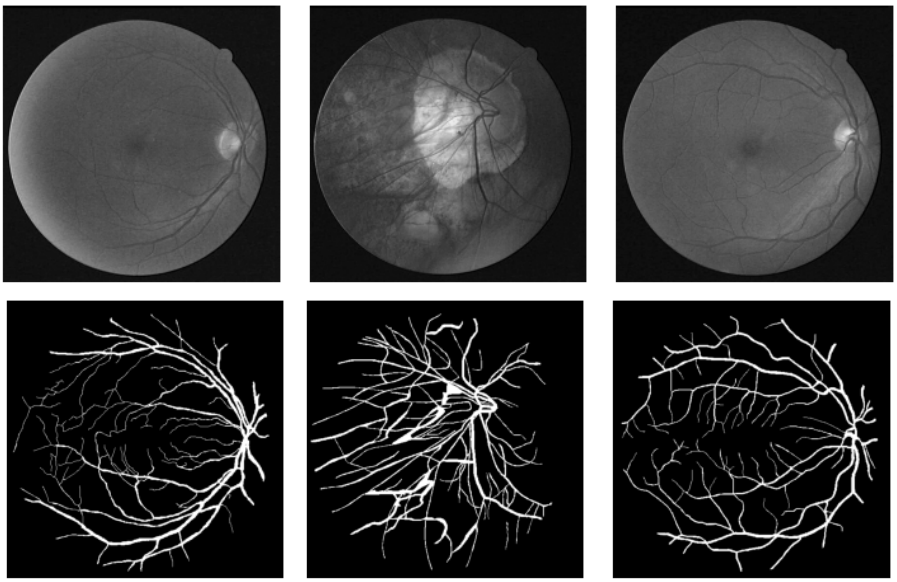

FIGURE 1.1 – Paires d'images provenant du jeu de données DRIVE.

*Retinal Images for Vessel Extraction* (DRIVE), où les vaisseaux sanguins associés à la rétine de l'œil sont segmentés. Obtenir une cartographie complète de ces vaisseaux est une mine d'or pour les médecins, dont une partie des applications possibles est listée ci-dessous :

1. Dépistage de la rétinopathie diabétique.
2. Analyse de correspondance entre l'hypertension et la taille des vaisseaux sanguins.
3. Chirurgie laser assistée par ordinateur.
4. Génération et simulation de cartes rétiniennes.
5. Identification biométrique.

De nombreux jeux de données, i.e. des ensembles de paires d'images (image brute, image segmentée), de ce type sont disponibles en ligne, gratuitement, et font même parfois l'objet de concours à grande échelle avec récompense à la clé (concours Kaggle etc.). La mise en ligne d'une telle quantité d'images, parfois amené avec un appât du gain non négligeable nous montrent l'importance de ce champ d'étude pour le biomédical.

Segmenter correctement les vaisseaux sanguins comme le montre la figure 1.1 n'est pas une tâche aisée et peut représenter un défi de taille, même pour un humain.
En micro-biologie cellulaire, la segmentation est également un axe de recherche important. L'intérêt est de pouvoir comprendre comment une cellule interagit avec son environnement qui peut être constitué d'autres cellules, de fluides ou de structures plus macroscopiques comme les parois d'un organe. Les présents travaux s'intéresseront en particulier à un jeu de données fourni par le laboratoire de micro-mécanique de l'Université de Technologies de Compiègne. Les données correspondantes montrent des micro-capsules,

aux propriétés mécaniques similaires aux globules rouges, se déformant lors d'un écoulement à travers un micro-canal dont le diamètre est proche de la taille des vaisseaux sanguins.

Les meilleurs algorithmes de segmentation permettant de résoudre ces tâches utilisent l'**apprentissage profond**, actuellement à l'état de l'art. Ces stratégies se basent sur un corpus d'images annotées par un expert afin de pouvoir apprendre et inférer différentes propriétés liées au problème. Une grande partie des recherches présentées dans ce document visent à détailler et comprendre les mécanismes propres à ces algorithmes. Ils mettent également en lumière une faiblesse de ces approches, nécessitant une base de données souvent conséquente afin d'obtenir des résultats satisfaisants. L'annotation de nouvelles données par l'utilisateur est un processus fastidieux, dans certains cas il se doit d'impliquer une personne experte, ce qui peut complètement bloquer la phase de développement. Les solutions permettant de pallier ce défaut consistent à générer artificiellement les données manquantes, ou bien à modifier l'approche afin d'utiliser un algorithme moins sensible à la taille du jeu de données initial et qui s'appuiera sur des biais inductifs ou des connaissances a priori.

Les travaux réalisés au cours de ce doctorat visent à donner des réponses plus performantes à ce problème, notamment par la mise en place de deux stratégies permettant d'optimiser chacun des deux axes mentionnés. Une description plus approfondie des contributions est donnée dans la section suivante.

## 1.3 Contributions

Ce manuscrit fait état de deux contributions majeures, sanctionnées à l'internationale.
La première est liée à la génération artificielle de données afin d'augmenter la taille du jeu d'images utilisé par les algorithmes de segmentation à base d'apprentissage automatique. Les stratagèmes permettant de créer de nouvelles données à partir d'un corpus initial sont vastes, mais peuvent présenter des défauts. Par exemple, bruiter trop fortement une image peut finalement perturber le fonctionnement d'un certain nombre d'algorithmes de segmentation qui n'est pas compensé par l'enrichissement de l'échantillon de départ. La contribution apportée vise à pouvoir identifier automatiquement les mécanismes de génération de données artificielles étant contre-productifs pour la résolution du problème demandé. De cette manière, il est possible de les écarter pour ne pas perturber l'exécution de l'algorithme de segmentation. Notre cas d'étude s'est concentré sur les architectures de réseaux de neurones profonds dont le fonctionnement est longuement détaillé dans le chapitre 3 et ont recours à la descente de gradient comme procédure d'apprentissage. La validation du protocole proposé se fait sur quatre jeux de données biomédicales. Plus précisément, deux d'entre eux font partie d'un challenge de traçage de cellules en ligne, un autre provient du jeu de données DRIVE mentionné dans la partie précédente. Le dernier

jeu de données nommé Capsule est issue de la collaboration avec l'UTC de Compiègne, elle aussi mentionnée plus haut. Il consiste à visualiser l'écoulement d'une micro-capsule dans un tube bidirectionnel à l'aide d'une caméra haute-fréquence. Enfin, ces travaux ont fait l'objet d'une publication à l'internationale par le biais d'une acceptation à la conférence sur l'acoustique, le langage et le traitement du signal (ICASSP 2022) [26]. Les deux premiers chapitres de ce manuscrit ont pour objectif d'offrir une revue de la littérature ainsi qu'un rappel des fondements théoriques sur lequel se basent nos travaux. Le lecteur averti peut alors directement se référer au chapitre 4 s'il souhaite passer à la description détaillée des travaux de recherche.

La seconde contribution repose sur la création d'un protocole visant à fournir une segmentation de bonne qualité, et ce pour un jeu de données généralement très pauvre en images annotées. La stratégie mentionnée réutilise des algorithmes à l'état de l'art en segmentation, et vise à améliorer leurs souplesses d'utilisation. Bien qu'il ne soit pour le moment pas possible de s'affranchir complètement de la nécessité d'avoir un jeu de données sans aucune supervision, l'architecture proposée permet entre autres d'obtenir des résultats satisfaisants à partir de 3 données annotées. De telles performances impliquent généralement des contreparties. Dans notre cas, des concessions sont faites sur le temps de calcul, le contenu des images traitées, ainsi que sur la nécessité d'avoir un large corpus d'images non annotées à disposition. Ce second point est très représenté dans les problèmes actuels, de nombreux challenges de segmentation en ligne (Kaggle ou autre) proposent des jeux de données ne contenant qu'une faible proportion annotée. Pour le temps de calcul, il est vrai que la mise en place des différentes briques de notre stratégie demande des ressources conséquentes. Cependant, une fois la phase d'apprentissage de notre algorithme terminée, son utilisation en phase de test ne présente pas de surcoût calculatoire. De plus, l'apprentissage le plus intensif du protocole revêt une forme de généricité. En effet, nos expériences montrent que cette étape ne nécessite pas d'être reproduite pour résoudre d'autres tâches de segmentation (on parle de transférabilité). Le cadre applicatif de notre protocole est pour le moment restreint aux jeux de données monocellulaires, et a pour objectif principal de répondre positivement à la collaboration avec l'UTC de Compiègne. Le lecteur averti pourra directement passer à la lecture du chapitre 5 afin d'obtenir de plus amples informations sur le sujet.

## 1.4 Organisation du mémoire

Le corps du manuscrit est découpé en 4 chapitres dont les grandes lignes vont être résumées ci-dessous.

À la suite du présent chapitre introductif, le chapitre 2 vise à introduire les stratégies de segmentation ne reposant pas sur de la supervision ni de vastes bases d'images. Ces algorithmes font l'objet d'une vaste littérature dont les

grandes lignes sont expliquées. Ces stratégies reposant sur une simple analyse de l'image proposent des performances plus modestes que celles obtenues par l'apprentissage profond. Malgré tout, elles ont l'avantage d'être génériques. La première section de ce chapitre est consacrée à l'utilisation de stratégies de pré-traitement de données, ce qui revêt une importance majeure pour tout utilisateur (y compris pour les méthodes à base d'apprentissage automatique). De plus, avoir une idée des performances obtenues sans utilisation de données annotées nous donne un aperçu de la difficulté du problème si on souhaite retirer intégralement toute forme de supervision.

Le chapitre 3 se concentre sur la présentation de l'apprentissage profond. C'est un chapitre qui se veut simple à comprendre, où chaque brique du fonctionnement des réseaux de neurones sera expliquée. La compréhension du chapitre est alors graduelle et adaptée afin de toucher un plus grand nombre de lecteurs (néophytes inclus). La seconde partie du chapitre est dédiée aux réseaux de neurones prenant en entrée des images, les différentes architectures spécialisées pour la segmentation y sont alors décrites. L'enjeu principal étant de donner toutes les bases nécessaires pour la compréhension des deux contributions qui suivent la lecture.

Le chapitre 4 est associé à la première contribution de nos travaux de recherche. Les intérêts que portent notre algorithme sont mis en lumière, avant de rentrer dans la phase de formalisation de notre procédé s'appuyant sur l'apprentissage en ligne et l'algorithme HEDGE. Une description de la littérature connexe est aussi de mise. La dernière section du chapitre se consacre aux perspectives offertes par nos travaux, ainsi que les évolutions possibles de notre approche.

Le chapitre 5 recense toutes les informations sur la seconde contribution. Une section introductive vise à donner au lecteur les grandes intuitions permettant de comprendre l'intérêt de chaque brique de la stratégie expliquée par la suite et à mieux situer notre contribution par rapport aux autres travaux de la littérature. La dernière section du chapitre ne se contente pas uniquement de lister les évolutions possibles, elle met également en avant certaines limites de notre approche et vise à y apporter un regard critique.

Le chapitre 6 sert à conclure le manuscrit. Il repasse brièvement sur l'importance des travaux réalisés et détaillés dans chaque chapitre. L'intérêt de chaque contribution est également rappelé. Enfin, les perspectives de travaux futurs, ainsi que les discussions qu'ils engendrent sont évoquées. Ces lignes directrices sont assez larges et présentent différents niveaux de priorité au regard des limitations qu'elles permettront de lever.

# Chapitre 2

# Segmentation d'images

La segmentation est une tâche majeure en traitement du signal. Elle consiste à partitionner les pixels de l'image en fonction de critères prédéfinis. L'objectif étant de pouvoir identifier les zones ou régions (en terme d'ensemble de positions sur la grille pixellaire) occupées respectivement par un ou différents objets présents dans une image, ou bien d'en extraire le fond. La segmentation peut être aussi considérée comme une tâche de classification à part entière où chaque pixel de l'image doit être classifié. S'il n'y a que deux classes, alors on parle de *binarisation* ou encore de **segmentation sémantique**. L'image binaire ainsi obtenue est appelée **masque**. Par opposition, on peut aussi vouloir attribuer à chaque objet de l'image son propre masque ce qui revient à attribuer un code (typiquement une couleur arbitraire) à chaque objet. On parle alors de **segmentation par instance**. Les travaux présentés dans ce manuscrit se concentrent sur la segmentation sémantique dont une partie est dédiée à la reconnaissance et la caractérisation automatique de cellules ou micro-structures de manière globale dans une image de microscopie. À noter que le passage à une analyse cellule par cellule ou structure par structure peut faire l'objet de travaux futurs.
La recherche à ce sujet est riche et vaste. Sans avoir la prétention d'être exhaustif, ce chapitre vise à détailler les différentes approches de l'état de l'art permettant de segmenter une image et qui ne s'appuient pas sur une quelconque base de supervision. En particulier les méthodes évoquées dans ce chapitre ne nécessitent en entrée que l'image à segmenter et éventuellement quelques paramètres liés à l'algorithme.

**Notations générales** Avant de présenter les concepts évoqués ci dessus, il est opportun de s'entendre sur quelques notations mathématiques qui seront reprises dans tout ce manuscrit. L'objet d'intérêt de ces travaux étant des images, nous commençons par préciser que, de manière générale, la notation par défaut d'une image sera $\mathbf{x}$ et tout vecteur sera également associé à un symbole en gras. D'un point de vue informatique, une image est un tableau 2D indexé par un couple d'entier qu'on notera souvent $(i, j)$. Ces paires $(i, j)$ aussi appelées pixels sont organisées en réseau 2D nommé grille pixellaire. Pour faire référence au niveau de gris (ou à la couleur) d'un pixel, on notera la valeur correspondante $x_{ij}$. Mais si pour un certain algorithme, la notion de grille 2D n'a pas un intérêt particulier, on verra $\mathbf{x}$ comme un vecteur colonne dont la $i^{\text{ème}}$ composante $x_i$ sera aussi appelée pixel. La taille d'une image $\mathbf{x}$ candidate à la segmentation sera notée $H \times W$, chacun de ces

entiers représentant donc la valeur maximale respective des indices $i$ et $j$. De même $N = WH$ est la quantité de pixels présents dans l'image.
Sans précision particulière, une image sera supposée en niveau de gris et $x_{ij} \in [0;1]$. Plus rarement, une image pourra être vue comme un signal analogique, à savoir une fonction notée $x : \mathbb{R}^2 \mapsto [0;1]$. Elle est alors vue comme une fonction de deux variables notées $u$ et $v$ pour chacun des axes de $\mathbb{R}^2$
L'objectif de la segmentation est d'obtenir des images binaires, appelées masques, et qui caractérisent la région occupée par un objet d'intérêt dans une image. Ces masques seront généralement notés $\mathbf{y}$ et tout pixel $y_{ij} \in \{0;1\}$.
Enfin, une lettre calligraphique telle que $\mathcal{V}$ désignera un ensemble tandis qu'une lettre majuscule telle que $X$ désignera une variable aléatoire.

## 2.1 Pré-traitement des images

De nombreux algorithmes de segmentation s'utilisent non pas directement sur l'image mais sur une version modifiée de celle-ci. On parle de pré-traitement pour qualifier un ensemble de processus visant à améliorer la qualité de l'image ou, dans notre cas, faire ressortir des attributs saillants fournissant une représentation commode pour opérer par la suite un second algorithme de reconnaissance de formes. Cette catégorie d'algorithmes est très vaste et inclut entre autres du lissage, de la restauration d'images, ou encore des modifications plus importantes comme la création de la carte des gradients associée à l'image. Par gradient, on considère de fait les algorithmes mettant en évidence les contours des objets. Calculer les variations du contraste de l'image est un pré-requis dans de nombreux algorithmes de segmentation.

### 2.1.1 Gradients

Une image peut-être vu comme un ensemble de pixels correspondant à un signal discret en 2D. Si l'image est en nuance de gris, alors chaque pixel est représenté par une seule valeur, correspondant à l'intensité lumineuse. Mathématiquement, avant son acquisition et sa transformation en donnée informatique, on peut définir une image par une fonction $f : \mathbb{R}^2 \mapsto \mathbb{R}$ : L'ensemble de départ correspond aux coordonnées d'un point alors que la valeur d'arrivée est l'intensité du pixel. Quand cette fonction est de plus différentiable, le calcul du gradient permet de déterminer les variations d'intensité pour tout point de $\mathbb{R}^2$ et donc aussi pour ceux situés sur la grille de pixels.
Le gradient d'une fonction $f(u,v)$, également défini par l'expression :

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial u} \\ \\ \frac{\partial f}{\partial v} \end{bmatrix} \tag{2.1}$$

C'est un vecteur de dimension 2 composé des dérivées partielles par rapport aux coordonnées $u$ et $v$. Ce vecteur pointe dans la direction dont la croissance est la plus importante. La norme quant à elle, indique l'intensité de la variation. Il semble donc intéressant d'étudier les normes les plus élevées car il y a de bonnes chances que ce critère localise le contour d'un objet.

Le souci dans le calcul des gradients vient du fait que nos images ne sont pas réellement définies dans $\mathbb{R}^2$ mais dans une grille discrète comme évoqué plus haut et que l'on peut voir comme un sous-ensemble de $\mathbb{Z}^2$. Autrement dit, nous ne connaissons la valeur des pixels que pour des coordonnées flottantes. De ce fait, nous allons calculer des approximations des gradients à travers l'utilisation de filtres de convolution.

### 2.1.2 Filtres de détection de contour

Le filtre de Sobel [63] permet d'approximer le gradient de chaque point d'une image. Il se base sur l'utilisation de noyaux de convolution [119] afin de calculer la différence d'intensité entre des pixels proches.
Le produit de convolution noté $\mathbf{m} * \mathbf{n}$ consiste, pour un noyau de convolution $\mathbf{n}$ et une matrice $\mathbf{m}$ de même taille, à effectuer une symétrie centrale [1] sur les composantes de $\mathbf{n}$ puis à sommer le produit de toutes les valeurs ayant la même position. Dans le cas usuel, la matrice $\mathbf{m}$ filtrée est plus grande, alors il faut répéter le processus pour chaque pixel de $\mathbf{m}$ ayant un voisinage valide de dimension égale à celle de $\mathbf{n}$.
Le gradient étant la dérivée partielle des deux coordonnées, le filtre de Sobel consiste à appliquer en réalité deux filtres. Un pour approximer la dérivée partielle de la 1ère coordonnée, et une autre pour la seconde.
Les noyaux de convolution associés aux filtres de Sobel sont les suivants :

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

TABLE 2.1 – Les filtres de Sobel

Ces filtres possèdent une valeur deux fois supérieurs pour les pixels relatifs au voisinage 4, en comparaison des ceux associés au voisinage 8. Les pixels "plus proches" selon ce type de voisinage ont donc une plus grande importance dans le calcul du gradient.

Un autre type de filtre, dit de Prewitt permet de donner autant d'importance à ces deux voisinages.

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

TABLE 2.2 – Les filtres de Prewitt

Il est aussi opportun de calculer les dérivées secondes directement pour notre problème de reconnaissance de contours. Dans ce cadre, on cherchera à annuler la somme des dérivées secondes dans les deux directions. On obtient

1. Si $\mathbf{n}$ est de taille $3 \times 3$, la symétrie en question consiste à échanger la composante $n_{11}$ avec $n_{33}$ et ainsi de suite pour toute paire de composante symétriquement opposées autour de $n_{22}$.

alors un filtre unique, dit Laplacien. Cependant ces approches basées sur les dérivées secondes rencontrent des problèmes de stabilités liés au bruit. Elles sont plus sensibles aux modifications légères de l'image initiale.

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

TABLE 2.3 – Filtre de Laplacien

Les résultats des différents filtres (en norme pour les gradients) en application sont illustrés en fig 2.1-2.4.

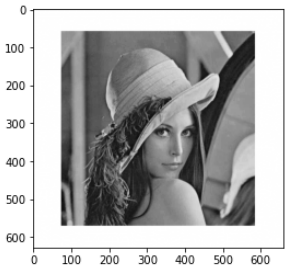

FIGURE 2.1 – Image de Lena

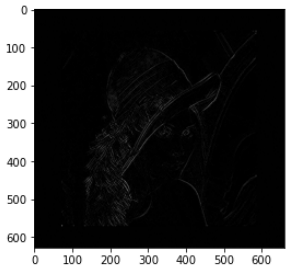

FIGURE 2.2 – Filtres de Sobel

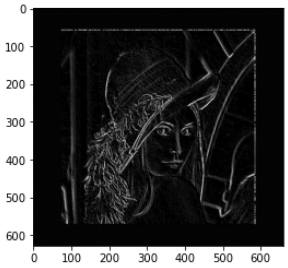

FIGURE 2.3 – Filtres de Prewitt

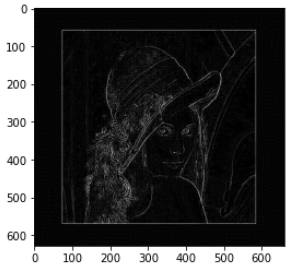

FIGURE 2.4 – Filtre Laplacian

### 2.1.3 Lissage d'images

Le principal problème du calcul approché des gradients de l'image se situe dans la gestion du bruit. Une simple modification de quelques valeurs peut complètement changer le résultat du filtrage. C'est pourquoi ces opérations de pré-traitements sont potentiellement elles-mêmes précédées d'une phase de lissage permettant de réduire le bruit. Ces filtrages permettent d'améliorer la précision des stratégies de détections de contours au détriment d'une perte d'informations dans l'image initiale. En effet, ces modifications ne sont pas réversibles.
Bien que de nombreuses techniques permettent de lisser une image, les filtres moyenneurs, medians [53] et gaussiens [75] sont les plus utilisés.

Le filtre moyenneur va remplacer la valeur d'un pixel par la valeur moyenne de ce même pixel et de ses voisins. Même idée pour le filtre médian qui va remplacer la valeur d'un pixel par la valeur médiane de ce même pixel et de ses voisins. C'est une approche très simple à mettre en place.

| | | |
|---|---|---|
| 15 | 16 | 16 |
| 15 | 255 | 18 |
| 14 | 15 | 19 |

TABLE 2.4 – Voisinage de médiane 16 autour d'un pixel blanc

Le filtre dit gaussien repose sur la création d'un noyau dont les valeurs correspondent à une fonction gaussienne. Nos filtres de convolution ayant deux coordonnées en chaque point, la valeur d'un point sera donc issue du produit de deux gaussiennes :

$$n_{ij} = \frac{1}{\sqrt{2\pi\sigma}} \exp^{-\frac{(i_c-i)^2+(j_c-j)^2}{2\sigma^2}} \tag{2.2}$$

avec $(i_c; j_c)$ la paire d'indices correspondant au centre du filtre. Cela correspond au calcul de la distance spatiale entre le pixel central du filtre et celui de coordonnées $(i, j)$. Un exemple de noyau gaussien est présenté ci-dessous :

| | | | | |
|---|---|---|---|---|
| 0.005 | 0.018 | 0.027 | 0.018 | 0.005 |
| 0.018 | 0.060 | 0.089 | 0.060 | 0.018 |
| 0.027 | 0.089 | 0.132 | 0.089 | 0.027 |
| 0.018 | 0.060 | 0.089 | 0.060 | 0.018 |
| 0.005 | 0.018 | 0.027 | 0.018 | 0.005 |

TABLE 2.5 – Noyau gaussien avec $\sigma = 1.12$

L'efficacité de ce type de stratégie dépend grandement de la taille du noyau. Un noyau de taille 5x5 est en général le minimum afin d'obtenir un résultat souhaitable. Augmenter la taille du noyau possède le désavantage d'augmenter le temps de calcul du filtrage, ainsi que d'augmenter l'effet du floutage. Une taille trop faible peut cependant n'avoir que peu d'impact. Il n'y a pas de consensus sur la meilleure taille de filtres mais la majorité des applications utilisent ceux de taille 5x5 et 7x7. Les figures 2.5 à 2.8 montrent l'intérêt d'opérer un lissage avant une détection de contours.

### 2.1.4 Algorithme de Canny

Le filtrage de Canny[18] est un grand classique de la détection de contour. L'algorithme se décompose en 3 étapes distinctes :

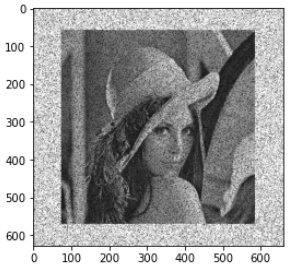

FIGURE 2.5 – Image de Lena bruitée

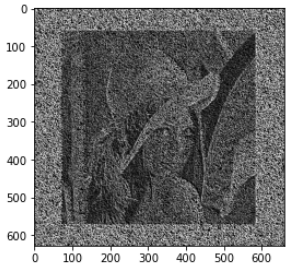

FIGURE 2.6 – Filtres de Prewitt sur Lena bruitée

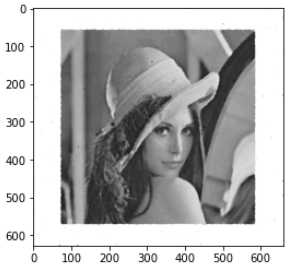

FIGURE 2.7 – Filtre médian sur Lena bruitée

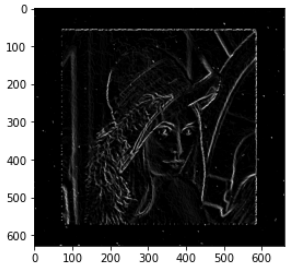

FIGURE 2.8 – Filtres de Prewitt sur Lena débruitée

**Algorithme 1 :** Filtre de Canny

Réduction du bruit,
Calcul des gradients,
Double seuillage par hystérésis.

La première étape du filtrage de Canny est la réduction du bruit. Les stratégies reposant sur le calcul de gradients ont la faiblesse d'être fortement perturbées. Un lissage de l'image permet donc naturellement d'améliorer la robustesse de l'algorithme. C'est un filtre gaussien qui est utilisé dans le cadre de cet algorithme. Les librairies utilisent généralement un filtre de taille 5x5, bien qu'une modification légère de la taille du filtre n'altère en rien les résultats du filtrage.

La seconde partie de l'algorithme est une simple étape de détection de contours par utilisation de filtres de convolution. À partir de deux filtres de dimension 3x1 et 1x3, il est possible de calculer des gradients horizontaux et verticaux. La manière de procéder est la même que pour les filtres de Sobel ou que le lissage d'image.

A noter qu'il est également possible d'obtenir la direction des gradients et non uniquement l'intensité en utilisant la fonction arctangente. En effet, la direction des gradients est toujours perpendiculaire au contour recherché. La fonction arctangente est directement appliquée sur la sortie des filtres selon

| -1 | 0 | 1 |
|---|---|---|

| -1 |
|---|
| 0 |
| 1 |

TABLE 2.6 – Les filtres de Canny

la formule suivant pour la position $(i, j)$ :

$$\text{Angle}_{ij} = tan^{-1}\left(\frac{\delta_{u,ij}}{\delta_{v,ij}}\right) \tag{2.3}$$

où $\delta_u$ et $\delta_v$ sont les images de gradients verticaux et horizontaux obtenus respectivement à l'issue de l'étape précédente.
La dernière étape consiste en un seuillage des différents gradients obtenus. La mesure d'intensité obtenue lors de l'étape précédente est un bon indicateur mais ne suffit pas à elle seule. C'est pourquoi un seuillage par hystérésis est utilisé. À l'aide de deux valeurs, généralement laissées au choix de l'utilisateur, il est possible de directement associer les différents gradients à un contour ou non. Si un gradient relatif à un pixel possède une valeur comprise entre les deux bornes, haute puis basse choisies par l'utilisateur, alors le pixel est considéré comme appartenant au contour si l'un de ses voisins est déjà associé au contour. Si la valeur est supérieure à la borne haute, le pixel est automatique considéré comme un contour, car un gradient élevé indique une zone de forte transition. Enfin, si le gradient ne dépasse la valeur basse, alors il est automatiquement rejeté.

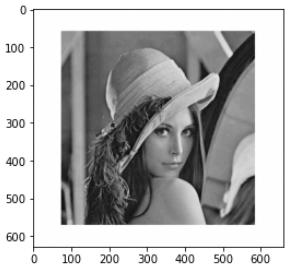


FIGURE 2.9 – Image de Lena

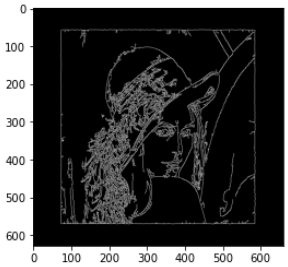


FIGURE 2.10 – Canny avec seuillage [30,220]

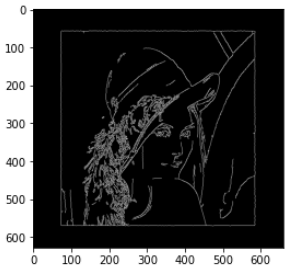


FIGURE 2.11 – Canny avec seuillage [110,220]

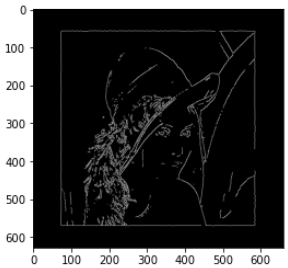


FIGURE 2.12 – Canny avec seuillage [200,220]

### 2.1.5 Gradient morphologique

Le gradient morphologique de Beucher [103] a pour avantage de se baser sur des opérateurs dits morphologiques dont le calcul nécessite un nombre réduit d'opérations.
On peut définir la carte des gradients morphologiques d'une image $\mathbf{x}$ par la formule suivante :

$$\delta\mathbf{x} = (\mathbf{x} \oplus \mathcal{B}) - (\mathbf{x} \ominus \mathcal{B}) \tag{2.4}$$

avec $\oplus$ et $\ominus$ les opérateurs morphologiques de dilatation et d'érosion alors que $\mathcal{B}$ représente la boule unité ouverte centrée sur $(0,0)$. La boule représentant un voisinage basé sur la distance euclidienne par rapport à l'origine du repère, et ce dans un certain rayon $r$.
Pour une position pixellaire $(i,j)$, la dilatation morphologique est définie par :

$$(\mathbf{x} \oplus \mathcal{B})_{ij} = \max_{(i',j') \in \mathcal{B}+(i,j)} x_{i'j'}, \tag{2.5}$$

où $\mathcal{B} + (i,j)$ représente la boule $\mathcal{B}$ translatée en $(i,j)$. C'est une opération qui est en général non réversible. L'érosion morphologique qui a pour objectif d'inverser le processus et provoque donc des pertes d'informations. Elle s'obtient par

$$(\mathbf{x} \ominus \mathcal{B})_{ij} = \min_{(i',j') \in \mathcal{B}+(i,j)} x_{i'j'} \tag{2.6}$$

Les travaux de Beucher expliquent que les résultats obtenus en utilisant les gradients morphologiques sont de même niveau que ceux avec des gradients euclidiens, hormis quelques cas particuliers. De plus, l'utilisation de ces gradients est à la base d'une stratégie de segmentation connue nommée ligne de partage des eaux. Cette stratégie sera visitée plus tard dans le mémoire.

### 2.1.6 Changement de représentation et calcul d'attributs

Dans la suite de ce chapitre, plusieurs techniques de segmentation sont présentées. Elles reposent essentiellement sur une analyse de similarité entre les valeurs $x_i$ et $x_j$ associées aux pixels correspondants afin de déterminer si ils appartiennent ou non à la même région . Pour présenter ces méthodes sans alourdir le manuscrit, on considérera que les valeurs en question sont simplement les niveaux de gris et la dissimilarité entre les pixels numérotés $i$ et $j$ est caractérisée par $|x_i - x_j|$ (sauf mention contraire explicite).
En réalité, le succès des méthodes de segmentation dépend grandement de la nature de ce qui permet de juger de la (dis)similarité entre pixels. La plupart du temps, il est très utile de changer de représentation en passant par le calcul, en chaque pixel, d'un vecteur d'attributs. Dans ce cas, et pour ne pas complexifier les notations, on verra $x_i$ comme un vecteur dans $\mathbb{R}^d$.
Certains des pré-traitements évoqués dans les paragraphes précédents peuvent déjà être utilisés à cette fin. En plus de ces derniers, il existe deux grandes familles d'attributs qu'on calcule fréquemment et qu'on présente brièvement ci-après :

— Attributs de texture : la texture d'une région de l'image correspond à une sensation visuelle de "degré de rugosité". Mathématiquement, on la caractérise par des variations locales des niveaux de gris qui suivent un certain *pattern*. Un exemple de texture est le damier qui est caractérisé par l'analyse d'un voisinage $2 \times 2$. En traitement d'images, la plupart des attributs qu'on calcule sont de nature statistique, comme ceux de Haralick [42] liés aux cooccurrences de niveaux de gris entre pixels voisins ou les *local binary patterns* [92]. Des outils temps-fréquence (filtres de Gabor [31]) ou fractales (exposant de Hölder [57]) peuvent aussi s'avérer très utiles.
— Attributs de couleur : quand l'image de départ est en couleur, $x_i$ est déjà un vecteur de $\mathbb{R}^3$ contenant une information colorimétrique primaire (rouge, vert, bleu). On peut mentionner qu'il est possible de changer d'espace couleur dans le but de séparer des informations de luminance et chrominance (voir l'étude [16] pour plus d'informations). Au delà de cette question, l'information couleur est parfois transformée en attributs statistiques notamment sous la forme d'histogrammes [120].

Bien sûr, plusieurs natures d'attributs peuvent être utilisées ou combinées. L'ensemble des attributs sélectionnés est ensuite concaténé dans un vecteur. Les méthodes de segmentation présentées dans la suite du chapitre sont généralement facilement transposables aux situations où le niveau de gris a été substitué par un vecteur d'attributs, ce qui se nomme l'*embedding*.

## 2.2 Méthodes basées sur la détection de régions

L'une des approches les plus simples pour segmenter une image consiste à la diviser en plusieurs régions en utilisant des mesures de similarité. Chaque région correspond alors à une classe de segmentation et les pixels peuvent rejoindre ou non une région à l'aide d'un critère prédéfini et intrinsèquement lié à la méthode choisie. Parmi ces différentes stratégies, nous allons expliquer plus en profondeur la croissance de régions, l'approche par séparation-fusion ainsi que les lignes de partage des eaux. Pour obtenir un aperçu plus détaillé des différentes approches, le lecteur pourra se référer au papier suivant [62].

### 2.2.1 Méthode par croissance de régions

La méthode par croissance de région (*Region Growing*) [90, 143, 60] est une stratégie simple et efficace de segmentation. Elle se base sur un système de voisinage pour identifier si un pixel appartient ou non à une classe.
On peut résumer l'algorithme par 3 étapes importantes. Dans un premier temps, il faut choisir des pixels de départ dans l'image, car le principe de la stratégie est de considérer les points initiaux comme des classes, puis d'élargir les différentes classes associées. On nomme les points de départ, les germes. Afin d'étendre les zones associées aux germes, il faut définir un critère d'homogénéité. Enfin, les zones associées à chaque germe vont être

agrandies à l'aide d'une fonction de voisinage qui sera appelée itérativement jusqu'à ce que tous les pixels de l'image soient associés à une zone. Ces zones, associées aux germes sont en réalité les classes de notre segmentation. Le pseudo-code très général de l'algorithme est donné ci-dessous.

**Algorithme 2 :** Croissance par régions

```
germesZones = initialPlacementGerme() ;
tant que Toute l'image n'est pas segmentée faire
    pour i=0 ; i<size(germesZones) ;i++ faire
        V_i = Voisinage(germesZones_i) ;
        V_i^(val) = VoisinsValides(V_i) ;
        pour j=0 ; j< size(V_i^(val)) ;j++ faire
            si critHomogénéité(germesZones_i, V_{i,j}^(val)) alors
                ajouter V_{i,j}^(val) à germesZones_i ;
            fin
        fin
    fin
fin
```

Le placement initial des pixels dit germes est déterminant dans le fonctionnement de cette procédure. Si par exemple, un germe est initialement placé dans une zone ayant de grandes perturbations d'intensité lumineuse, alors la croissance ne pourra pas s'effectuer convenablement. C'est pourquoi le placement initial des germes se base sur un compromis entre l'intensité moyenne des pixels autour de la zone, l'espacement moyen entre les autres germes ainsi qu'un écart de valeur entre les différents pixels initiaux. Concernant le critère d'homogénéité, on considère que les pixels voisins dans une même région ont normalement une valeur proche. La différence entre les deux pixels est donc majoritairement utilisée pour déterminer si ils appartiennent à la même région, mais dans une image texturée, il conviendra d'utiliser un changement de représentation à l'aide d'une méthode d'extraction d'attributs pour évaluer l'homogénéité. Enfin, la fonction de voisinage utilisée dans l'image correspond classiquement à un voisinage de 4-connexité et de 8-connexité au sens de la grille pixellaire.
Cet algorithme a fait l'objet de nombreuses améliorations au fil du temps. Les parties suivantes ont pour objectif de donner un rapide aperçu de l'état de l'art en exposant différentes optimisations.

S. A. Hojjatoleslami et J. Kittler[50] ont proposé une nouvelle fonction d'acceptation. Deux nouvelles mesures sont calculées, le contraste moyen et le contraste périphérique. Ces mesures ne vont plus uniquement se baser sur le voisinage d'un pixel, mais également prendre en compte tous les pixels extérieurs de la région, que nous appellerons pixels de bordure. La mesure de contraste moyen est calculée par la différence entre les pixels d'une région, et les nouveaux pixels de voisinage dont la formule est montrée ci-dessous :

$$\text{cont}_i = \frac{1}{i}\sum_{j=1}^{i} x_j - \frac{1}{k-i}\sum_{j=i+1}^{k} x_j \tag{2.7}$$

avec $x_1, ..., x_i$ représentant les pixels de la région et $x_{i+1}, ..., x_k$ sont les pixels de voisinage. Le contraste périphérique est lui défini par la différence entre les pixels de bordure et ceux voisins de la bordure. Utiliser de nouvelles métriques basées sur une différence moyenne de plusieurs pixels permet d'être plus robuste au bruit auquel la fonction générique de calcul de similarité est particulièrement sensible.
Les travaux de S. A. Hojjatoleslami et J. Kittler montrent également que les résultats obtenus pour les deux métriques sont très proches en pratique. Seul des effets de floutages prononcés peuvent créer un écart. Ainsi, il est recommandé de segmenter uniquement grâce à la seconde métrique qui est moins coûteuse en temps de calcul. En effet, les pixels de bordure sont peu nombreux en comparaison à l'intégralité des pixels de la région.

Rolf Adams et Leanne Bischof[1] proposent une optimisation du temps de calcul de l'algorithme par croissance de régions. L'idée est de définir un ordre précis dans le déroulement de l'algorithme, notamment pour la gestion du voisinage. En effet, pour chaque pixel de notre région, nous devons définir un voisinage, et ensuite utiliser une mesure de similarité pour accepter ou non le nouveau pixel donc notre région. L'idée innovante ici est de ne pas effectuer cette étape naïvement pour chaque pixel de notre région, mais plutôt de les sélectionner dans un ordre précis. Les pixels de notre région vont donc être triés à partir de la différence entre la valeur du pixel, et la moyenne des pixels de la région. Le postulat dit qu'il est plus probable pour un pixel dont la valeur est très proche à celle de la moyenne de sa région, d'avoir des voisins appartenant également à cette même région. Les travaux de Rolf Adams et Leanne Bischof montrent que pour des résultats similaires, le temps de calcul est bien plus rapide. L'utilisation d'une liste chainée pour stocker les pixels d'une région permet d'effectuer des insertions et suppressions sans perte de temps.
Ces travaux sont la base d'une riche littérature. On peut par exemple citer Andrew Mehnert et Al.[85] qui ont travaillé sur la gestion des collisions et des égalités au sein de la liste chainée avec l'objectif d'améliorer l'efficacité globale de l'algorithme.

### 2.2.2 Algorithme de séparation-fusion

À l'inverse de la croissance en région. Un algorithme de séparation-fusion [51, 22] va considérer au départ l'image entière comme une région. Puis dans un processus itératif, si les régions ne respectent pas un critère d'homogénéité, alors elles vont être découpées. De plus, une fonction dite de fusion permet d'assembler les régions isolées présentant des similarités. Cette idée générale d'algorithme en deux étapes est très célèbre et est utilisée dans de nombreux domaines. Dans le cas de la segmentation d'images, le critère d'homogénéité est généralement identique de ceux de l'approche par croissance de régions. Pour l'étape de séparation, une région qui ne respecte par le

critère d'homogénéité sera typiquement divisée en quatre sous-régions rectangulaires de surfaces égales. La fusion et la séparation de régions reposent sur un critère prédéfini de similarité dont la définition est cruciale pour le succès de la méthode. En outre, deux régions ne peuvent fusionner que si elles sont sécantes. L'algorithme prend fin quand la phase de séparation ne crée aucune nouvelle région.

### 2.2.3 Transformation Watershed

Les transformations par ligne de partage des eaux[10, 12, 11] désigne une famille de méthodes mathématiques permettant de segmenter des images en niveaux de gris par l'intermédiaire de la création d'une carte en 3 dimensions. En considérant notre image comme un relief topographique, alors il est possible de segmenter l'image en isolant naturellement les bassins versants.

En pratique, la création du relief n'est jamais effectuée directement sur l'image initiale. On utilise des phases de pré-traitements qui se basent sur le calcul de gradients, ou d'une autre stratégie permettant de mettre en évidence les zones de transitions. L'algorithme de ligne de partage des eaux n'est alors réalisé qu'en dernier.
Une fois notre image pré-traitée correctement, l'utilisation de l'algorithme de partage des eaux (LPE) peut-être résumé à un processus d'inondations. En simulant l'inondation d'un relief topologique, alors les zones les plus basses vont être sous l'eau alors que les reliefs resteront dégagés. Dans notre cas, si l'image a reçu un pré-traitement idéal associé à la mise en évidence des contours, alors notre image possède des valeurs élevées autour des contours (reliefs), et des valeurs très faibles sur les zones sans transition. Par conséquent, en inondant les minimas locaux de notre image, cela mettra en évidence les reliefs représentant les contours, ainsi que les zones sous l'eau nommées bassins versants qui correspondent en réalité aux différentes classes de la segmentation souhaitée.

L'ordre d'inondation est un point clé de l'algorithme et la littérature à ce sujet est vaste. Nous allons présenter ici la méthode de Meyer et Beucher (1991) [86]. A l'aide d'une file (FIFO), il faut gérer tous les voisins des minimas locaux et insérer ceux ayant la valeur la plus faible en priorité. Ensuite, pour chaque élément de la file, on génère les voisins associés et on classe l'élément en fonction de la majorité du voisinage. Les voisins n'ayant aucune classe associée sont ajoutés à la file. L'algorithme s'arrête quand la file est vide, et les pixels non classifiés sont donc les contours ou reliefs de l'image.

## 2.3 Méthodes basées sur les contours

Au lieu de s'intéresser aux régions, d'autres approches effectuent une segmentation en se basant sur la notion de contours. Les contours délimitent les

différentes classes de notre image, l'objectif est généralement de pouvoir obtenir de manière précise le contour de chaque objet, et d'effectuer la segmentation ensuite. Ces approches utilisent pour la majorité un pré-traitement de lissage (filtre gaussien etc..). Les contours actifs [64] ainsi que l'algorithme de Chan-Vese [21] font partis des approches les plus utilisées à ce jour, nous en discuterons dans les prochaines sections.

### 2.3.1 Contour actif

On désigne par contour actif [64], également nommé *snake* un ensemble de points définissant une courbe paramétrée (type spline) dans un espace en deux dimensions. Initialement placée très approximativement autour de l'objet d'intérêt, la courbe va s'en approcher lentement au cours d'un processus itératif qui va lui permettre d'épouser la forme de l'objet. L'objectif est de rapprocher notre *snake* des zones de forts gradients tout en respectant des contraintes. Pour ce faire, Le déplacement de la courbe se base sur la notion de forces internes, externes et liées à l'image (attâche aux données). Les forces internes sont responsables de la régularisation du contour (vitesse de variations et courbures), alors que les forces externes sont associées à différentes contraintes pouvant être éventuellement ajoutées selon la nature de l'image et les connaissances a priori dont on dispose. En l'absence de telle information, le terme externe est omis. Enfin les forces de l'image ont pour but de nous rapprocher de zone de l'image à fort gradients ou présentant certaines propriétés d'intérêt.
Soit $\boldsymbol{\vartheta}(s) = (u(s), v(s))$ la courbe continue engendrée par le *snake* où $s$ désigne un paramètre appartenant à l'intervalle $[a, b]$ et permettant de parcourir la courbe avec la contrainte $v(a) = v(b)$. On cherche alors à minimiser l'énergie totale du modèle représentée par la formule suivante :

$$E_{snake} = \int_a^b E_{interne}\left(\boldsymbol{\vartheta}(s)\right) + E_{image}\left(\boldsymbol{\vartheta}(s)\right) + E_{externe}\left(\boldsymbol{\vartheta}(s)\right) \, \mathrm{d}s \tag{2.8}$$

Nous donnons ci-dessous quelques détails sur la formulation des trois énergies. L'énergie interne se définit par

$$E_{interne} = \frac{1}{2}\left(\alpha(s)\left\|\frac{d\boldsymbol{\vartheta}}{ds}\right\|^2 + \beta(s)\left\|\frac{d^2\boldsymbol{\vartheta}}{ds^2}\right\|^2\right) \tag{2.9}$$

avec $\alpha(s)$ et $\beta(s)$ des poids à choisir par l'utilisateur. Le premier terme à minimiser encourage des contours peu longs et le second des contours à faible courbure. Ces contraintes agissent sur la régularité de la forme du *snake*.
Concernant le terme d'attâche aux données, il se compose classiquement lui même de trois sous-termes :

$$E_{image} = w_{ligne}E_{ligne} + w_{terminaison}E_{terminaison} + w_{contour}E_{contour}, \tag{2.10}$$

avec des poids associés en face de chaque terme. La gestion de ces différents poids est aussi à la charge de l'utilisateur. La manière la plus simple de définir $E_{ligne}$ est de considérer uniquement l'intensité lumineuse de l'image $x\left(u\left(s\right), v\left(s\right)\right) = x\left(\boldsymbol{\vartheta}\left(s\right)\right)$. De cette façon, notre *snake* sera plus facilement attiré par les lignes blanches ou noires selon le signe du poids $w_{ligne}$. Cette

stratégie pour définir $E_{ligne}$ n'est pas unique, mais nous ne pourrons expliciter dans ce manuscrit toutes les différentes possibilités. Le sous-terme $E_{contour}$ est défini par la formule :

$$E_{contour} = - \left\| \nabla x \left( u \left( s \right), v \left( s \right) \right) \right\|^2 \tag{2.11}$$

Notre nuage de points (les points de contrôle définissant le *snake*) est donc naturellement attiré par les forts gradients qui sont intrinsèquement liés aux contours de l'objet.
L'énergie associée aux terminaisons des coins et des contours utilise l'angle de la courbure de l'objet. Pour obtenir un tel terme, on passe déjà par un filtrage gaussien tel que vu en (2.2). Notons $x_g$ l'image ainsi filtrée. En considérant le gradient des angles par :

$$\theta \left( s \right) = tan^{-1} \frac{\frac{\partial x}{\partial u} \left( u \left( s \right), v \left( s \right) \right)}{\frac{\partial x}{\partial v} \left( u \left( s \right), v \left( s \right) \right)} \tag{2.12}$$

ainsi que le vecteur unitaire perpendiculaire à la direction du gradient par :

$$\mathbf{r} = \left( - \sin \theta, \cos \theta \right) \tag{2.13}$$

alors l'énergie associée à la terminaison peut s'écrire comme la dérivée de l'angle sur sa direction :

$$E_{term} = \frac{\partial \theta}{\partial \mathbf{r}}. \tag{2.14}$$

Enfin, l'énergie externe étant à définir au cas par cas, selon qu'un tel contexte soit disponible, elle ne possède pas d'expression générale. L'algorithme marche correctement sans l'utiliser, mais il est également possible de rajouter des contraintes particulières pour rendre l'algorithme plus spécifique.

Afin de minimiser l'énergie, la méthode numérique utilisée se base sur les équations d'Euler afin d'approximer le résultat [64]. Un pseudo-code (Alg. 3) fournit un résumé de l'algorithme itératif de minimisation des contours actifs.

**Algorithme 3 :** Contours actifs

```
Snake = initialPlacementUtilisateur() ;
tant que le critère d'arret n'est pas respecté faire
    pour i=0 ; i<size(snake) ;i++ faire
        V = voisinage(snake_i) pour j=0 ; j<size(V) ;j++ faire
            E_j = calculEnergie() ;
        fin
        j* = arg min_j E_j ;
        snake_i = V_j* ;
    fin
fin
```

La méthode de contours actifs présente plusieurs désavantages :

— La stratégie n'est pas entièrement automatique, le positionnement initial du contour doit être fait par l'utilisateur.
— Les points du *snake* ont beaucoup de mal à pouvoir rentrer dans des concavités. Cette approche marche donc efficacement pour les objets convexes, ce qui limite les cas d'utilisation.
— L'équation aux dérivées partielles issue de l'application du théorème d'Euler Lagrange à la minimisation de l'énergie globale fait apparaître une dérivée d'ordre 4, ce qui implique des instabilités vis à vis des méthodes numériques correspondantes.

Les figures 2.13 à 2.16 montrent quelques étapes d'un contourage actif par *snake* d'une cellule dans une image de microscopie, en partant d'une initialisation favorable.

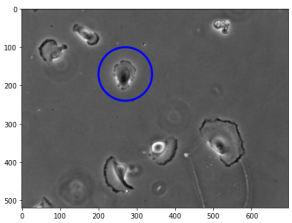


FIGURE 2.13 – Placement initial du *snake*.

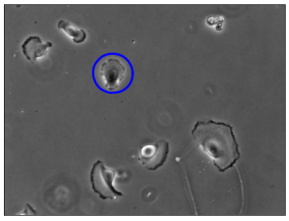

FIGURE 2.14 – *snake* après 100 itérations.

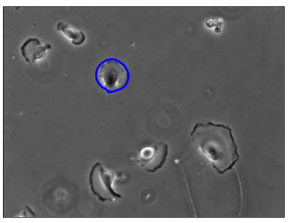

FIGURE 2.15 – *snake* après 150 itérations.

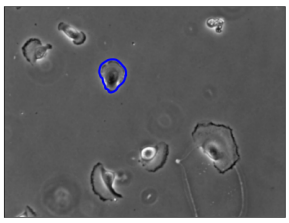

FIGURE 2.16 – *snake* après 200 itérations.

### 2.3.2 Contours actifs de Chan-Vese

La méthode des contours actifs de Chan-Vese [21] appartient à une vision purement variationelle du problème contrairement à la méthode précédente qui est paramétrée par les points de contrôle de la spline. En ce sens, l'image recherchée est nécessairement dans un premier temps vue comme une fonction de $\mathbb{R}^2$, solution d'un problème de minimisation d'une fonctionnelle. Cette fonction est également constante par morceaux avec deux régions, ce qui est équivalent peu ou prou à un masque binaire. Cette fonctionnelle va reprendre un principe similaire à l'approche *snake* dans sa composition qui contient un terme d'attache aux données, un terme de régularisation sur la longueur du contour de la région segmentée et un second terme de régularisation sur la surface occupée par la région segmentée (pénalisant fortement une segmentation qui occuperait l'image en entier). Plus précisément, le terme d'attache aux données est sous-divisé entre l'intérieur et l'extérieur

de la région segmentée avec une pondération à choisir par l'utilisateur de la méthode. Cette fonctionnelle est également un cas particulier d'une formalisation plus générale due à Mumford-Shah [91].
La minimisation de la fonctionnelle de Chan-Vese passe par la définition d'une fonction $\phi$ dont la ligne de niveau, pour le niveau zéro, est le contour de la région recherchée. On peut alors résoudre le problème par alternance : avec $\phi$ fixé, on trouve la valeur prise par la fonction recherchée à l'intérieur et à l'extérieur du contour. Ensuite pour ces valeurs fixées, on met à jour $\phi$ à l'aide d'une descente de gradient. Un exemple de segmentation par la méthode de Chan-Vese est proposé par les figures 2.17 à 2.20.

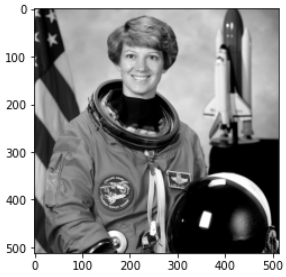


FIGURE 2.17 – Image d'une astronaute

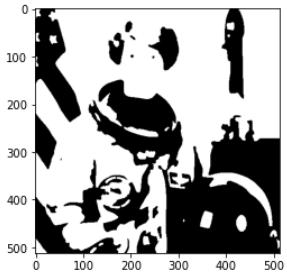


FIGURE 2.18 – Segmentation de Chan-Vese

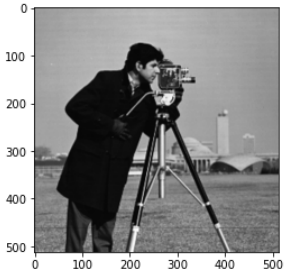


FIGURE 2.19 – Image d'un caméraman.

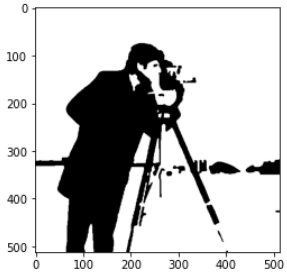


FIGURE 2.20 – Segmentation de Chan-Vese

### 2.3.3 Contours actifs morphologiques

Márquez-Neila et al. [82] expliquent que nombre d'équations aux dérivées partielles peuvent être vue comme l'application d'opérateurs d'érosion ou de dilatation tels que ceux définis en section 2.1.5. Or, la solution des problèmes de contours actifs pouvant précisément elles mêmes être vue comme des équations aux dérivées partielles, on peut donc rechercher un contour actif par applications successives d'opérateurs de morphologie.
Les méthodes de contours actifs morphologiques sont réputées plus stables et rapides que leurs pendants analytiques pour une qualité de segmentation comparable.

## 2.4 Méthodes de clustering

Le *clustering* est une technique d'apprentissage automatique non supervisée permettant de grouper les différents pixels de notre image en fonction d'un critère de similarité ou de distance. Ces algorithmes ont généralement un bon ratio entre la performance et le temps de calcul. Les sections suivantes visent à détailler l'algorithme des K-moyennes [80] ainsi que le *Mean-shift* [37].

### 2.4.1 Méthode des $K$-moyennes

L'algorithme des $K$-moyennes [80] permet de partitionner un ensemble de données en $K$-groupes. La valeur de $K$ étant à choisir par l'utilisateur, il est tout à fait possible de partitionner avec $K$ étant le nombre d'objets présents, ce qui correspond à effectuer une segmentation. L'intuition général qui porte l'algorithme est de considérer qu'un *cluster* de points, ou groupement, doit être un ensemble de points où leurs inter-distances sont supposées être plus faibles comparées à celles calculées avec des points en dehors du *cluster*. Nous allons alors définir des centroïdes nommés $c_0, c_1, ..., c_K$ pour les $K$ différents groupes. L'objectif est alors d'assigner chaque point de notre image à l'un des centroïdes de manière à ce que la somme des distances au carré entre un point et son centroïde soit minimale. Pour mieux formaliser le problème, nous allons définir pour chaque pixel $x_i$ de l'image, un ensemble de $K$ variables binaires nommées $r_{i,k}$ pour $k \in \{1..K\}$. Si la variable $r_{i,k} = 1$, cela signifie que le pixel $i$ est associé au *cluster* $k$. La somme des distances au carré qu'on souhaite minimiser peut alors être définie par :

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} r_{i,k} ||x_i - c_k||^2. \tag{2.15}$$

On rappelle que $N$ est le nombre de pixels de l'image $\mathbf{x}$ selon la convention adoptée en début de chapitre. On cherche alors à trouver la meilleure attribution pour chaque variable $r_{i,k}$ et les meilleurs centroïdes afin de minimiser $J$. Pour ce faire, une procédure itérative basée sur deux étapes est utilisée. Dans un premier temps, on initialise les différents centroïdes, cette étape, bien que souvent faite de manière aléatoire a une influence sur le reste du processus. Ensuite, la 1ère étape consiste à minimiser $J$ sachant les valeurs des centroides fixées, en ne pouvant jouer uniquement que sur les valeurs $r_{i,k}$. La seconde étapes est alors de minimiser $J$ en modifiant cette fois-ci les valeurs $c_k$ car les $r_{i,k}$ sont fixés. On peut mettre en relation ces deux phases avec respectivement les phases d'espérance et maximisation utilisées dans l'algorithme EM[27, 84], généralisation probabiliste des $K$-moyennes que nous ne présenterons pas pour des raisons de concision. De plus, les $K$-moyennes s'inscrivent aussi dans la continuité des algorithmes de type séparation-fusion (section 2.2.2). La phase de modification de l'emplacement des centroïdes pouvant être assimilée à la séparation, alors que la seconde phase qui rattache les points au cluster le plus proche est une phase de fusion.

Minimiser la fonction objectif $J$ par rapport à $r_{i_0,k_0}$ se résume à examiner seulement les termes de la somme où $i = i_0$ car l'affectation à un *cluster* pour le pixel numéro $i_0$ n'affecte pas les autres pixels [2]. En revanche, choisir $r_{i_0,k_0} = 1$ implique que $r_{i_0,k} = 0$ pour tout $k \neq k_0$ puisqu'un pixel ne peut appartenir qu'à un seul *cluster*. Finalement, il est clair que la solution consiste à trouver pour le pixel numéro $i_0$, le centroïde $c_{k_0}$ le plus proche. Cette phase est donc traitée indépendamment pour chaque point de notre image, de manière itérative. On cherche alors à mettre la valeur $r_{n,k}$ à 1 pour le le centroïde $c_k$ qui minimise $||x_i - c_k||^2$. On peut alors en déduire l'expression suivante :

$$r_{i,k} = \begin{cases} 1 & \text{si } k = \arg\min_j ||x_i - c_j||^2 \\ 0 & \text{sinon} \end{cases} \tag{2.16}$$

Pour minimiser les $c_k$, il suffit d'annuler la dérivée de la fonction objectif $J$ par rapport à $c_k$, qu'on peut écrire de la manière suivante :

$$\frac{\partial J}{\partial c_k} = 2 \sum_{i=1}^{N} r_{ik}(x_i - c_k) = 0 \tag{2.17}$$

La résolution de cette équation nous donne :

$$c_k = \frac{\sum_{i=1}^{N} r_{ik} x_i}{\sum_{i=1}^{N} r_{ik}} \tag{2.18}$$

$r_{ik}$ ayant la valeur 1 pour un seul $k$. Le dénominateur est donc le nombre de points appartenant au cluster numéro $k$. Par conséquent, le résultat est en réalité une simple moyenne des points assignés par cluster, c'est de là que vient le nom des $K$-moyennes.
Le critère d'arrêt de l'algorithme est une comparaison de l'ensemble $r_{i,k}$ entre deux itérations. Si aucune modification est faite, alors l'algorithme s'arrête. Les $K$-moyennes sont connues pour être une stratégie simple d'utilisation, efficace pour un temps de calcul faible et une convergence en un temps fini. Malgré tout, c'est également un algorithme dépendant de l'initialisation des centroïdes, qui est généralement aléatoire.
Différentes études [19, 95, 65] montrent qu'il n'est jamais sûr d'obtenir la solution représentée par le minimum global, car on rentre très généralement dans des optimums locaux liés à l'initialisation.

### 2.4.2 $K$-moyennes pour la segmentation

La manière de procéder pour effectuer une segmentation avec l'algorithme des $K$-moyennes la plus trivial est d'utiliser directement le niveau de gris de pixels (ou bien la couleur selon la nature de l'image) comme suggéré dans les paragraphes précédents. En ce sens, un centroïde sera à valeur dans $[0; 1]$. Par conséquent, les pixels d'intensité lumineuse proche vont être regroupés ensemble autour d'un centroïde qui sera la moyenne de ces mêmes pixels.

2. C'est une faiblesse des approches à base d'apprentissage non-supervisé qui ne tiennent pas compte du voisinage entre pixels

Les figures 2.21 et 2.22 donnent un aperçu de ce que donne ce procédé sur une image en niveau de gris. Bien sûr, ce procédé peut être couplé avec un pré-traitement de type *embedding* pour associer à chaque position pixellaire un vecteur de caractéristiques plus informatif que le seul niveau de gris.

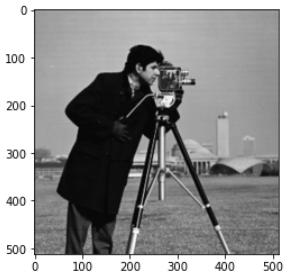


FIGURE 2.21 – Image d'un caméraman

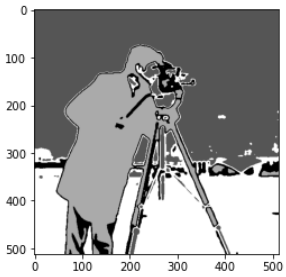


FIGURE 2.22 – k-moyennes avec $K = 4$

### 2.4.3 Méthode de Mean-Shift

La méthode de *Mean shift* proposé par Fukunaga [37, 23, 24] est un algorithme de clustering pouvant facilement être généralisé pour des tâches de segmentation. De la même manière que pour les $K$-moyennes, on se contentera ici de présenter le *Mean shift* dans ce cas d'usage où les points à rassembler en *clusters* sont des niveaux de gris.
Nous partons donc d'un échantillon $\{x_1, .., x_N\}$ de $N$ points en 1D répartis sur l'intervalle $[0; 1]$. A partir de cet échantillon, on peut construire un histogramme ou distribution empirique. Selon le pas de quantification choisi pour l'intervalle $[0; 1]$, nous verrions apparaître différents maxima locaux. Le principe du *Mean shift* est alors le suivant : chaque $x_i$ va être déplacé itérativement vers un maximum local proche. Au bout du compte, après suffisamment d'itérations, les points ne bougent plus et se fixent sur un certain maximum local. On a alors autant de *clusters* que de maxima locaux ayant attiré au moins un point et agissant comme des bassins d'attraction.
Bien évidemment, le bon fonctionnement de cet algorithme repose largement sur la définition de la règle de mise à jour des points pour effectuer les déplacement successifs. L'algorithme du *Mean shift* propose d'utiliser une fonction appelée noyau et notée $K : [0; 1] \mapsto \mathbb{R}^+$ comme pivot de la règle de mise à jour. Cette fonction doit être paire.
Bien des choix sont possibles pour cette fonction. On peut utiliser une gaussienne ou bien plus simplement un noyau plat correspondant à une $\lambda$-boule :

$$K(x) = \begin{cases} 1 & \text{si } |x| \leq \lambda \\ 0 & \text{sinon} \end{cases} \tag{2.19}$$

Pour calculer le déplacement qu'effectuera la valeur $x_j$ associée au $j$ème pixel, on calcule une moyenne pondérée où les points s'obtiennent grâce au noyau :

$$m(x_j) = \frac{\sum_i K(x_i - x_j) x_i}{\sum_i K(x_i - x_j)}. \tag{2.20}$$

On calcule cette moyenne pour tout l'échantillon de point. La différence $m(x_j) - x_j$ est le déplacement, ou *mean-shift*, et $x_j$ sera remplacé par $m\left(x_j\right)$ lors de la prochaine itération.
Cet algorithme a montré son efficacité en pratique mais il n'existe pas encore à notre connaissance de preuve de convergence générale.

### 2.4.4 Quickshift

L'algorithme de Mean-shift souffre de temps de calcul importants, avec une complexité de $O(dN^2 + N^{2.38})$ où $d$ est la dimension de l'espace où vivent nos points ($d = 1$ dans notre cas où ils correspondent à des niveaux de gris). C'est pourquoi Andrea Vedaldi et Al. [127] proposent une version améliorée nommée Quick shift, qui permet d'obtenir des résultats presque identiques, pour un temps de calcul plus faible ($O(dN^2)$).
Quickshift utilise une règle de mise à jour un peu différente. La valeur d'un pixel $x_i$ va être remplacée par la valeur du pixel voisin pour lequel une augmentation du critère $P_j = \frac{1}{N}\sum_i K(x_i - x_j)$ est constatée où $j$ est l'index du voisin en question. S'il y a plusieurs voisins qui satisfont ce critère, alors c'est celui pour lequel le critère est minimal qui est retenu.

## 2.5 Segmentation par utilisation de graphes

De nombreuses approches de segmentation d'images utilisent la théorie des graphes. En effet, ces objets ont l'avantage de pouvoir facilement modéliser les liens entres les pixels de l'image. Les sommets du graphe correspondent donc généralement aux pixels, alors que les arrêtes vont être des caractéristiques de ressemblance entre les pixels (même niveau de gris, éloignement, voisinage). L'exploitation de ces informations permet alors de segmenter l'image. Nous allons étudier dans cette section deux approches très populaires, la segmentation par coupe normalisée [109] ainsi que la méthode de Felzenszwalb [33].

### 2.5.1 Segmentation par coupe normalisée

La segmentation par coupe normalisée proposée par Jianbo Shi et Jitendra Malik [109] est un algorithme qui consiste à partitionner l'image initiale à l'aide de coupures. Pour arriver à la définition de la notion de coupe, il faut déjà associer à l'image à segmenter un graphe (non orienté) pondéré $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ où l'ensemble des pixels/sommets est noté $\mathcal{V}$, l'ensemble des paires de pixels liés entre eux (ou arête) est noté $\mathcal{E}$ et l'ensemble des poids associés à ces liaisons est noté $\mathcal{W}$. Il est alors possible de partitionner un graphe en deux ensembles disjoints $A$ et $B$ tels que :

$$A \cap B = \mathcal{V}, A \cup B = \emptyset \tag{2.21}$$

Pour ce faire, il suffit de supprimer les arrêtes connectant les deux sous-ensembles. On peut alors définir le degré de dissemblance entre les deux sous-ensembles en sommant le poids des arrêtes supprimées. On nomme cette variable la coupe.

$$\text{coupe}(A, B) = \sum_{a \in A, b \in B} w(a, b) \tag{2.22}$$

avec $w(a, b)$ le poids associé à la liaison entre le pixel $a$ et le pixel $b$ (poids nul si la liaison n'existe pas). Trouver la valeur de coupe la plus faible est la manière optimale de diviser notre graphe initial. Cette approche aboutit à une segmentation de l'image si les poids sont construit de sorte à associer un poids élevé à la paire $(a, b)$ quand ces deux pixels sont vraisemblablement dans la même région. Si on se base sur le seul niveau de gris, on pourrait par exemple définir $w(a, b) = \exp\left(-(x_a - x_b)^2\right)$.
En raison du nombre exponentiel de combinaisons, trouver la coupe optimale est un problème très étudié [135]. Shi et Al proposent une nouvelle mesure du coupe nommée la coupe normalisée.

$$\text{coupeNorm}(A, B) = \frac{\text{coupe}(A, B)}{\text{assoc}(A, \mathcal{V})} + \frac{\text{coupe}(A, B)}{\text{assoc}(B, \mathcal{V})} \tag{2.23}$$

avec $\text{assoc}(A, \mathcal{V})$ correspondant au poids total des arêtes liant le sous-ensemble $A$ à l'ensemble $\mathcal{V}$.

$$\text{assoc}(A, \mathcal{V}) = \sum_{a \in A, v \in \mathcal{V}} w(a, v) \tag{2.24}$$

L'avantage de cette nouvelle formule est qu'elle ne désavantage pas les partitionnements de grands ensembles. En effet, la formule de coupe précédente ne prend pas en compte la taille des ensembles $A$ et $B$. Les petites coupures sont alors prioritaires car le nombre d'arêtes supprimées est plus faible. Dans le cas de la coupe normalisée, puisque la valeur de coupe est divisée par le nombre d'arêtes, alors pour minimiser cette valeur, il est plus intéressant de considérer de grandes coupes.
Le calcul de cette nouvelle partition optimale se base sur des propriétés de décomposition matricielle, la preuve entière est détaillée dans l'article original [109].
L'algorithme se résume alors en quatre étapes distinctes.

1. Pour un graphe initial $G$, calculer les poids $w$ pour chaque arrête.
2. Résoudre $(\mathbf{D} - \mathbf{W})\mathbf{x} = \lambda \mathbf{D}\mathbf{x}$ où $\mathbf{W}$ est la matrice contenant les poids du graphe et $\mathbf{D}$ est une matrice diagonale telle que la composante $D_{ii} = \sum_{v \in \mathcal{V}} w(v, v_i)$ (poids total des connections au pixel numéro $i$). Les solutions sont des vecteurs propres généralisés.
3. Sélectionner le vecteur propre associé à la seconde plus petite valeur propre. Modulo un seuillage des composantes de ce vecteur pour les rendre binaires, on obtient un vecteur correspondant à un masque binaire de segmentation.
4. Décider s'il faut partitionner le graphe courant et relancer récursivement sur les deux sous-graphes.

### 2.5.2 Méthode de Felzenszwalb

Les travaux de Felzenszwalb [33] portent sur l'utilisation de graphe afin d'effectuer la segmentation. Basés d'une part sur les études de Zahn [138], pionnier dans la segmentation par graphe par l'intermédiaire du calcul de l'arbre couvrant minimum. La recherche de Felzenszwalb s'articule autour de la notion de régions pour donner la priorité aux pixels proches.
Comme pour la segmentation par couple normalisée, on associe à l'image un graphe pondéré $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ avec les même définitions pour les éléments composant le triplet $G$. Toutefois, ici, le poids noté $w(i,j)$ ne mesure pas la similarité entre le pixel numéro $i$ et le pixel numéro $j$, mais à l'inverse l'écart entre ces derniers. On prendra la différence d'intensité lumineuse entre deux sommets comme définition par défaut pour la fonction $w$ dans le cadre de cette méthode.
Pour partitionner le graphe, Felzenszwalb s'appuie sur la notion d'arbre couvrant de poids minimal d'une région $C \subseteq \mathcal{V}$. Un tel arbre, noté ACM $(C)$, est un sous-graphe de $G$ tel que tous les sommets dans $C$ soient liés sans créer de cycle et dont le poids total des arêtes de ce sous-graphe soit minimal. La stratégie se base sur la création d'une métrique permettant de valider ou non, si deux régions sont biens distinctes. On commence par s'intéresser au maximum d'un arbre couvrant de poids minimal d'une région $C$ :

$$\text{maxACM}(C) = \max_{i,j \in \text{ACM}(C)} w(i,j). \tag{2.25}$$

Pour comparer deux régions $C_1$ et $C_2$, on introduit la différence minimale comme suit :

$$\text{diff}(C_1, C_2) = \min_{i \in C_1, j \in C_2} w(i,j) \tag{2.26}$$

On combine enfin les deux définitions précédentes pour obtenir un critère de décision binaire quant au fait que les régions sont distinctes avec la formule suivante :

$$D(C_1, C_2) = \begin{cases} \text{Vrai} & \text{si diff}(C_1, C_2) > \min\{\text{maxACM}(C_1); \text{maxACM}(C_2)\} \\ \text{Faux} & \text{sinon} \end{cases} \tag{2.27}$$

Quand le cardinal d'une des régions est petit, admettons par exemple la région $C_1$, l'auteur recommande de modifier quelque peu la formule en rajoutant la constante $\frac{k}{|C_1|}$ à maxACM$(C_1)$ pour éviter de favoriser des coupes avec des régions à faibles cardinaux. Le paramètre $k$ est à choisir par l'utilisateur de la méthode et $|C_1|$ désigne le cardinal de la région $C_1$.
C'est grâce à ce même critère $D$ que nous allons pouvoir définir l'algorithme 4.

**Algorithme 4 :** Algorithme de Felzenszwalb

$m = |\mathcal{E}|$, nombre d'arêtes ;
$\mathcal{E}$ = Tri de $\mathcal{E}$ par ordre croissant de poids ;
$S$ = Segmentation initiale, chaque pixel est une région ;
**pour** $q \in \{1, .., m\}$ **faire**
  $(i, j) = e_q$, récupération des pixels de la $q$ème arête ;
  $C_1$ = région de $S$ contenant $i$ ;
  $C_2$ = région de $S$ contenant $j$ ;
  **si** $C_1 \cap C_2 = \emptyset$ **alors**
    **si** $w(i, j) \leq \min\{maxACM(C_1); maxACM(C_2)\}$ **alors**
      Fusionner $C_1$ et $C_2$ et mettre $S$ à jour en conséquence ;
    **fin**
  **fin**
**fin**
Renvoyer $S$

Un filtre gaussien est généralement utilisé en pré-traitement avant l'exécution de l'algorithme 4. Ce type de stratégie est en effet très fortement perturbé par le bruit, il suffit d'une seule valeur extrême pour provoquer une fusion de régions.
Cet algorithme a l'avantage de s'effectuer dans un temps relativement faible. La complexité est égale à $\Omega(m\ log(m))$ pour $m$ étant le nombre d'arêtes [3] du graphe associé à l'image. Les figures 2.23 et 2.24 présente le résultat obtenu par la méthode de Felzenszwalb pour une image test.

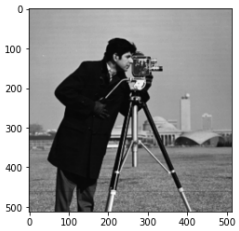


FIGURE 2.23 – Image d'un caméraman

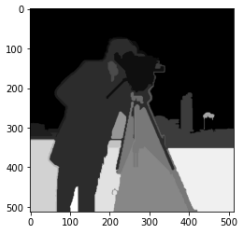


FIGURE 2.24 – Application de la méthodes de Felzenszwalb

## 2.6 Segmentation par champs de Markov

Une dernière grande famille d'approches de segmentation non supervisée et s'appuyant sur une seule image observée $\mathbf{x}$ est présentée succinctement dans les paragraphes qui suivent. Il s'agit des méthodes s'appuyant sur un modèle probabiliste joint de l'image recherchée et inconnue $\mathbf{y}$ et de $\mathbf{x}$. Ce modèle est appelé champ de Markov caché et a été appliqué en traitement d'images suite aux travaux de Geman et Geman [39].

3. Pour un graphe pleinement connecté $m = \frac{N(N-1)}{2}$.

**Remarque** : Habituellement, dans la littérature dédiée, $\mathbf{x}$ est utilisé pour la variable inconnue et $\mathbf{y}$ pour les observations. Pour rester cohérent avec le reste du manuscrit, plus en ligne avec la tradition du *machine learning*, on reste sur la convention inverse ici.

### 2.6.1 Définition du modèle

On commence par définir la loi régissant le masque binaire $\mathbf{y}$. Soit $(Y_i)$ la collection de variables aléatoires binaires et correspondant à la valeur du masque en chaque position pixellaire $i$. Plus précisément, cette collection est appelée champ pour rendre compte de l'existence d'une notion de voisinage caractérisée par un treillis (grille des pixels). On pose alors le modèle de densité suivant

$$p(\mathbf{y}) \propto \lambda \exp\left[-\sum_{i,j \text{ voisins}} \nu_1(y_i, y_j) - \sum_j \nu_2(y_i)\right]. \tag{2.28}$$

Ce modèle est relativement flexible et appartient à la famille des distributions de Gibbs. Les fonctions $\nu_1$ et $\nu_2$ caractérisent respectivement l'interaction entre pixels et l'aléa propre à chaque pixel. Un choix classique pour ces fonctions est le modèle d'Ising ou de Potts, voir [3] pour une définition de ces derniers.
Pour accéder à la loi jointe, il faut compléter ce modèle par celui de la loi des observations conditionnelles aux variables cachées puisque $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y})\, p(\mathbf{y})$. C'est alors qu'on introduit une hypothèse simplificatrice dite de Markov correspondant à l'indépendance conditionnelle de la valeur de niveau de gris observée $x_i$ pour un certain pixel $i$ par rapport à celle observée pour les autres pixels sachant la valeur de $y_i$ :

$$p(\mathbf{x}|\mathbf{y}) = \prod_i p(x_i|y_i). \tag{2.29}$$

En général, la densité de $p(x_i|y_i)$ possède une expression qui ne dépend pas de $i$.

### 2.6.2 Mise en œuvre

Pour déployer le modèle des champs de Markov, on s'heurte à deux difficultés : entraîner ce modèle, puis prédire d'après le modèle entraîné. Pour l'entraînement, il s'agit de déterminer des valeurs adéquates de paramètres qui rentrent dans la définition des fonctions $\nu_1$ et $\nu_2$ ainsi que potentiellement dans la densité $p(x_i|y_i)$. Pour mener à bien l'apprentissage on pourra minimiser la vraisemblance correspondante par descente de gradient. En non-supervisé, étant donné la présence de variables cachées, cette étape peut s'avérer très complexe et on préfère souvent proposer un modèle sans paramètre à apprendre.
Concernant la prédiction, ou inférence, on souhaite retourner

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}),$$

c'est à dire le maximum a posteriori. Une approximation de cet estimé peut être obtenu à l'aide de l'échantillonneur de Gibbs qui est un algorithme de type *Markov Chain Monte Carlo* (MCMC). D'autres algorithmes sont proposés dans [9].

## 2.7 Conclusion

Dans ce chapitre, il a été proposé un tour d'horizon des méthodes de segmentation ne s'appuyant ni sur une forme de supervision (au sens de l'apprentissage machine) ni sur un corpus d'images. Ces méthodes visent à parvenir à une segmentation par la seule analyse de l'image elle même à segmenter, ce qui est un problème difficile. C'est pourquoi les résultats obtenus par ces méthodes ne sont pas toujours d'une précision très élevée. Pour compenser le manque de données à exploiter, les meilleures méthodes relevant de cette catégorie doivent souvent compenser avec un temps de calcul important. Elles font donc toujours l'objet d'une activité de recherche intense pour trouver à la fois la meilleure formulation (attache aux données, régularisation, etc.) et la meilleure implémentation permettant l'optimisation la plus efficace vis à vis de la formulation retenue. Elles ont aussi l'avantage d'être génériques, par opposition aux méthodes basées apprentissage qui généralisent difficilement à de nouvelles images significativement différentes de celles vue pendant l'apprentissage. Cette généricité/flexibilité peut néanmoins parfois être obtenue au détriment de la nécessité du réglage de divers paramètres, et ce pour chaque image à segmenter.
Pendant longtemps, les chercheurs n'ont pas vraiment envisagé d'autres déclinaisons du problème de segmentation, mais avec l'avènement du *big data*, la démultiplication des quantités d'images (annotées ou non) disponibles a changé la donne et ouvert la voie à d'autre méthodes beaucoup plus centrées sur l'apprentissage machine et particulièrement l'*apprentissage profond* depuis une décennie. Ces méthodes sur lesquelles, des contributions seront développées dans les chapitres 4 et 5, sont présentées plus en détails dans le chapitre suivant. Comme nous le verrons, elles permettent de tirer de précieuses informations des jeux de données d'images et constituent aujourd'hui les méthodes les plus performantes de l'état de l'art.

# Chapitre 3

# Apprentissage profond par réseaux de neurones

Ces dernières années ont vu l'émergence de nouvelles stratégies de segmentation basées sur l'apprentissage supervisé. Contrairement au chapitre précédent, ces méthodes vont exploiter, non pas seulement l'information d'une seule paire $(\mathbf{x}, \mathbf{y})$, mais celle fournie par un jeu de données contenant plusieurs paires de la sorte. Parmi les différentes catégories existantes, c'est l'apprentissage profond (par réseaux de neurones) qui est actuellement le plus utilisé, notamment grâce à l'augmentation des ressources de calculs. Bien que d'autres familles de méthodes supervisées soient disponibles dans la littérature pour entraîner un modèle à segmenter des images [8, 78], on se concentrera donc sur celles relevant de l'apprentissage profond. Ce chapitre vise à expliquer en détails les fondements de l'apprentissage profond, avec un focus particulier sur la vision par ordinateur, et dans notre cas d'étude : la segmentation.

## 3.1 Réseaux de neurones artificiels

Les réseaux de neurones artificiels sont la base de l'apprentissage profond. L'objectif est d'approximer une fonction $f$, on considère alors que pour un jeu de données noté $(\mathbf{x}, \mathbf{y})$, avec $\mathbf{x}$ comme image initiale et $\mathbf{y}$ comme masque binaire de segmentation, il existe la relation suivante : $\mathbf{y} = f(\mathbf{x})$. On souhaite alors approcher la fonction $f$ qui permet à partir d'une image, d'obtenir la segmentation souhaitée par une fonction $f_{\boldsymbol{\theta}} \in \mathbb{R}^p$ où $p$ est la taille du vecteur de paramètre $\boldsymbol{\theta}$. En effet, cette fonction dépend de paramètres notés $\boldsymbol{\theta}$ dont on cherche à trouver la meilleure affectation. Il convient de modifier notre relation initiale par $\mathbf{y} \approx f_{\boldsymbol{\theta}}(\mathbf{x})$. La section suivante se concentre principalement sur l'explication détaillée des différents éléments structurant un réseau de neurones artificiels. Pour plus de simplicité, la présentation est dans un premier temps contextualisé à une tâche de classification binaire avant de revenir plus loin dans le chapitre à une tâche de segmentation. La distinction majeure entre les deux cas de figure est que, pour la classification binaire, la sortie du réseau est unidimensionnelle et donc notée $y$ selon notre convention et $y \in \mathcal{Y} = \{0; 1\}$.

### 3.1.1 Le Perceptron

Le modèle du Perceptron [14] est à l'origine des réseaux de neurones. Il se base sur une classification à deux classes, où un vecteur d'entrée $\mathbf{x}$ va directement être multiplié à un vecteur de poids $\mathbf{w}$, avant d'y appliquer en dernier une fonction de classification notée $g_{class}$.

$$y = g_{class}(\mathbf{w}^\intercal \cdot \mathbf{x} + b). \tag{3.1}$$

Dans ce cas, le vecteur de paramètres à apprendre est

$$\boldsymbol{\theta} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ b \end{bmatrix}.$$

La fonction choisie initialement par Rosenblatt consiste à utiliser l'origine pour effectuer la classification.

$$g_{class}(a) = \begin{cases} 1 & si\ a \geq 0 \\ -1 & si\ a < 0 \end{cases} \tag{3.2}$$

Afin de déterminer la valeur des poids $\mathbf{w}$ et du paramètre $b$ (souvent appelé *intercept*), l'algorithme se base sur la minimisation d'une fonction d'erreur par descente de gradient. Bien qu'il semble intéressant de prendre la somme des données d'entrées mal classifiées comme fonction d'erreur, cela peut poser problème car cette fonction est discontinue et propose des gradients à zéro dans la majorité des cas. Ainsi l'algorithme du perceptron va chercher à minimiser la différence entre la valeur de $a$ et l'origine, ce qui permet d'obtenir la fonction d'erreur suivante :

$$E(\boldsymbol{\theta}) = -\sum_{n \in \mathcal{M}} \left(\mathbf{w}^\intercal \cdot \mathbf{x}^{(n)} + b\right) y^{(n)} \tag{3.3}$$

où $\mathcal{M}$ correspond à l'ensemble des indices des données mal classifiées. Cette fonction de coût pénalise une mauvaise valeur du vecteur de paramètres au sens où un exemple $\left(\mathbf{x}^{(n)}, y^{(n)}\right)$ est mal classifié à partir du moment où $\left(\mathbf{w}^\intercal \cdot \mathbf{x}^{(n)} + b\right) y^{(n)} < 0$.
Il suffit alors d'appliquer la descente de gradient à cette fonction d'erreur pour obtenir la mise à jour des poids.

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - \eta \nabla_{\boldsymbol{\theta}} E = \boldsymbol{\theta}^r + \eta \sum_{n \in \mathcal{M}} \mathbf{x}_+^{(n)} y^{(n)}, \tag{3.4}$$

avec $\eta$ un hyperparamètre nommé *learning rate* qui est responsable essentiellement de la rapidité de convergence et $\mathbf{x}_+$ le vecteur $\mathbf{x}$ auquel un 1 a été adjoint en coordonnée $N+1$. Quand les données sont linéairement séparables, Rosenblatt a prouvé la convergence de cet algorithme d'apprentissage.

### 3.1.2 Perceptron multicouche

Un réseau de neurones artificiels ou perceptron multicouche peut être simplement considéré comme une composition de fonctions correspondant au modèle mono-couche. En prenant par exemple 3 fonctions nommées $f_1, f_2, f_3$, alors on peut les combiner ensemble afin de créer une fonction de plus haut niveau (au sens où elle est susceptible de traiter des frontières séparatrices non-linéaires) qu'on note :

$$\hat{y} = f_3(f_2(f_1(\mathbf{x}))). \tag{3.5}$$

Cette chaîne de fonctions est très fréquemment utilisée pour définir des réseaux de neurones, constitués de couches correspondant aux différentes fonctions. Parmi les fonctions ou couches qui composent un réseau de neurones, on peut distinguer la dernière, nommée couche de sortie ainsi que les précédentes appelées couches intermédiaires ou cachées. En général, on substitue à la fonction $g_{class}$ une autre fonction dite d'activation qui est dérivable et finalement $\hat{y}$ devient un réel correspondant (sans perte de généralité) au score de la classe 1. Pour décider quelle classe choisir, il faudra comparer $\hat{y}$ à un certain seuil ultérieurement.
Tout comme le perceptron monocouche, chaque neurone associé à une couche est directement connecté à tous les neurones de la couche précédente, on parle de couches entièrement connectées (*fully connected*). Les connections se font par l'ajout de poids à apprendre, et donc de paramètres à optimiser. Ce nombre de paramètres est donc d'ordre quadratique, ce qui peut entraîner des soucis numériques (mémoire et temps de calcul), mais cela permet également d'augmenter la capacité d'apprentissage du réseau.

L'optimisation d'un réseau de neurones se base également sur la descente de gradient appliquée à une fonction de perte adaptée qui doit donc être définie (c.f. prochaine section). La couche de sortie est alors liée à cette fonction de perte, tandis que les couches intermédiaires sont responsables de la propagation du flux d'informations à travers le réseau.

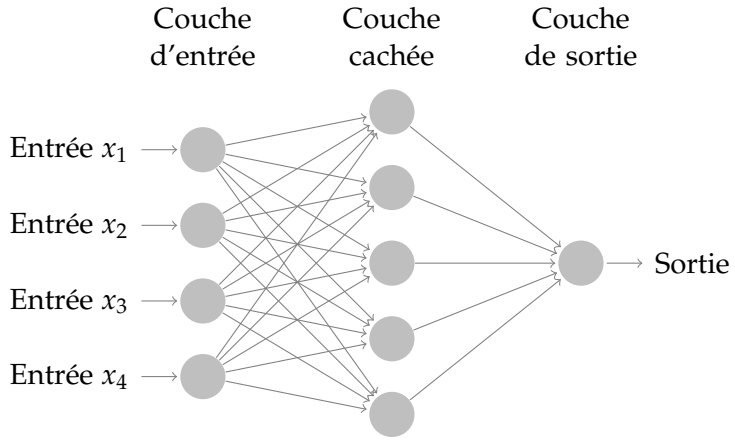


FIGURE 3.1 – Schéma d'un réseau de neurones multicouche.

La figure 3.1 donne la représentation graphique classiquement utilisée pour présenter le fonctionnement d'un réseau de neurones. Chaque noeud représente un neurone et chaque flèche une connexion entre neurones. Les connexions sont associées à un paramètre à apprendre.

### 3.1.3 Fonctions de perte

Le choix de la fonction de perte est déterminant dans la conception d'un réseau de neurones. En considérant qu'on souhaite trouver le meilleur ensemble de paramètres noté $\boldsymbol{\theta}$ qui maximise la probabilité d'obtenir la bonne valeur de sortie sachant notre donnée en entrée, notée $p(y|\mathbf{x};\boldsymbol{\theta})$, on trouve alors la meilleure approximation en passant par le maximum de vraisemblance correspondant au calcul de l'entropie croisée entre les données d'entrées et la sortie du réseau :

$$\mathcal{L}_{\mathrm{EC}}(\boldsymbol{\theta}) = -y \log f_{\boldsymbol{\theta}}(\mathbf{x}) - (1-y) \log\left(1 - f_{\boldsymbol{\theta}}(\mathbf{x})\right). \tag{3.6}$$

Alternativement, on peut aussi utiliser deux autres fonctions de perte très connues, i.e. l'erreur quadratique moyenne (MSE) [132] et l'erreur absolue moyenne (MAE) :

$$\mathcal{L}_{\mathrm{MSE}}(\boldsymbol{\theta}) = \|f_{\boldsymbol{\theta}}(\mathbf{x}) - y\|_2^2, \tag{3.7}$$

$$\mathcal{L}_{\mathrm{MAE}}(\boldsymbol{\theta}) = \|f_{\boldsymbol{\theta}}(\mathbf{x}) - y\|_1. \tag{3.8}$$

Associer une pénalité quadratique permet d'optimiser principalement les données considérées comment étant les plus mauvaises, c'est à dire celles ayant le plus gros écart entre le résultat du modèle et la sortie souhaitée (l'étiquetage $y$). La MAE de son côté correspond à une perte simplement proportionnelle à cet écart.
L'ensemble de ces définitions est donné pour un seul exemple d'apprentissage $(\mathbf{x}, y)$. Pour une certaine perte $\mathcal{L}$ choisie, la fonction de perte globale pour l'entièreté du jeu de données est définie par :

$$L(\boldsymbol{\theta}) = \sum_{(\mathbf{x},y)\in\mathcal{S}_{\mathrm{tr}}} \mathcal{L}\left(\boldsymbol{\theta}\right).$$

Afin d'alléger les notations, la dépendance de $\mathcal{L}$ en $\mathbf{x}$ et $y$ n'est pas rendue explicite mais doit être gardée en mémoire. L'ensemble des données d'apprentissage avec supervision est noté $\mathcal{S}_{\mathrm{tr}}$.

### 3.1.4 Fonctions d'activation du réseau

Comme évoqué précédemment, il existe de nombreuses fonctions d'activations pouvant être utilisées au sein d'un réseau et pouvant remplacer la fonction $g_{class}$. La création du réseau est assez libre en général, seul le choix de la dernière couche d'activation, directement reliée à la fonction de perte possède des contraintes. Nous allons définir ici certaines des fonctions les plus utilisées.

La première, nommée sigmoïde et visible en Figure 3.2 est généralement utilisée pour des tâches de classification binaire. Elle est définie par :

$$\text{sigm}(x) = \frac{1}{1+e^{-x}} \tag{3.9}$$

Elle permet d'obtenir un résultat compris entre 0 et 1, très utile quand on souhaite obtenir une probabilité. Elle est donc compatible avec la perte $\mathcal{L}_{\text{EC}}$. De plus, elle est également très simple à dériver, ce qui facilite les calculs associés à la descente de gradient.

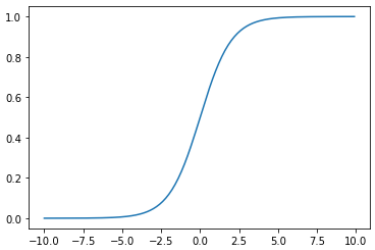


FIGURE 3.2 – Fonction sigmoïde

Cette fonction permet uniquement d'obtenir une probabilité pour deux classes. Afin de généraliser la fonction sigmoïde pour $n_c$ valeurs, on va alors utiliser la fonction softmax. Cette fonction va retourner un vecteur de $n_c$ probabilités pour chaque classe possible associée à notre problème. Chacune des valeurs correspond à une probabilité d'appartenance à la dite classe, et la somme du vecteur est forcément égale à 1. On a :

$$\text{softmax}(\mathbf{x}) = \begin{bmatrix} \frac{\exp(x_1)}{\sum_{j=1}^{n_c}\exp(x_j)} \\ \vdots \\ \frac{\exp(x_{n_c})}{\sum_{j=1}^{n_c}\exp(x_j)} \end{bmatrix}. \tag{3.10}$$

La couche de sortie du réseau doit être adaptée pour fournir $n_c$ sorties à la fonction softmax. Bien que ces fonctions soient utilisées pour générer des probabilités au niveau de la sortie du réseau, il est également parfaitement possible de les utiliser comme couches intermédiaires.

Une autre fonction d'activation nommée tangente hyperbolique est définie par la formule :

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}. \tag{3.11}$$

C'est une fonction très similaire à la sigmoïde, avec un lien entre les deux formules via l'équivalence suivante :

$$\tanh(x) = 2\,\text{sigm}(2x) - 1. \tag{3.12}$$

Il est à noter que $\tanh(0) = 0$, contrairement à la fonction sigmoïde qui retourne 0.5, ce qui rend la fonction tanh plus proche de l'identité.

Une autre fonction dite de correction Linéaire (Relu) [137] est également très utilisée :

$$\text{Relu}\,(x) = \max\{0; x\}\,. \tag{3.13}$$

Comme le montre la Figure 3.3, elle remplace toutes les valeurs négatives par zéro, ce qui permet entre autres une initialisation des poids plus parcimonieuse ainsi qu'une meilleure propagation du gradient. Les extremums de la fonction sigmoïde pouvant saturer très vite sont évités.

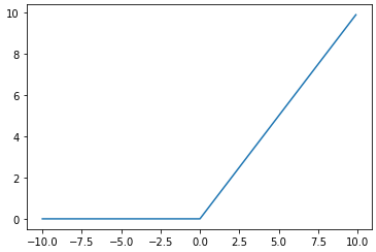


FIGURE 3.3 – Fonction relu

L'un des défauts de la fonction Relu vient du fait que le gradient est complément ignoré quand la valeur de la fonction est égale à zéro. C'est pourquoi il existe de nombreuses variantes de Relu tels que les couches leakyRelu [137] qui permettent de pallier ce défaut.

### 3.1.5 Algorithme d'entraînement du réseau

L'optimisation des paramètres du réseau de neurones correspond à l'entraînement du modèle et se base sur la descente de gradient. Le réseau étant considéré comme une fonction dérivable, on peut, sur la base d'une fonction de perte calculée à la fin du réseau, retrouver l'impact (haussier ou baissier) relatif à chaque paramètre vis à vis de cette dite fonction. Il ne reste plus qu'à modifier automatiquement les poids pour réduire le plus possible la fonction de perte. Un réseau étant généralement constitué de nombreuses couches, le calcul direct de la dérivée de l'erreur est extrêmement long et coûteux. C'est pourquoi Rumelhart et al. [106] proposent un algorithme nommé *Backpropagation* ou rétro-propagation, qui permet d'obtenir les gradients de chaque poids du réseau de manière bien plus efficace. Toute l'ingéniosité de l'algorithme repose sur la décomposition du calcul de la dérivée.

Considérons un exemple similaire et simplifié de celui de la section 3.1.2, dans lequel un réseau est une série de fonctions nommées $f_1, f_2$, directement appliquée à l'entrée : $\hat{y} = f_2(f_1(\mathbf{x}))$. Admettons que l'on ait besoin de la dérivée de la prédiction $\hat{y}$ par rapport au paramètre $w_1$ intervenant dans la définition de $f_1$. On peut alors décomposer le calcul de la dérivée de $\hat{y}$ par rapport à $w_1$ de la manière suivante :

$$\frac{\partial f_2}{\partial w_1} = \sum_i \frac{\partial f_2}{\partial f_{1,i}} \frac{\partial f_{1,i}}{\partial w_1}, \tag{3.14}$$

avec $i$ un indice qui parcourt les composantes du vecteur $f_1$ pour un certain $\mathbf{x}$ appartenant à notre jeu de données.
Il est ainsi possible de calculer la dérivée d'une fonction complexe à partir d'autres fonctions incluses dont la dérivée est déjà connue. L'algorithme de rétro-propagation du gradient va appliquer en chaîne ce procédé en partant de la dernière couche du réseau (fonction de perte comprise). Le calcul du gradient de la dernière couche est directement lié à la fonction de perte, il est très rapide à calculer et permet d'utiliser l'information liée à $y$. C'est le point de départ pour appliquer récursivement la chaîne de calcul des gradients. De plus, les dérivées des fonctions d'activation du réseau sont en général très simples à calculer, ce qui permet d'obtenir un gain de temps de calcul non négligeable. Un exemple : la dérivée de la fonction Relu est donnée par

$$\frac{d\text{Relu}}{dx}(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{sinon} \end{cases} . \tag{3.15}$$

Enfin, comme dérivée et somme commutent, on généralise facilement ce calcul à la somme des pertes pour tous les exemples d'apprentissage du jeu de données. Algorithmiquement, en partant de la couche de perte, la rétro-propagation peut se résumer aux étapes suivantes :

1. Calculer les dérivées de la couche courante par rapport à son ou ses entrée(s) et les stocker en mémoire.
2. Si la couche possède des paramètres, calculer les dérivées de la couche par rapport à ces paramètres
3. À l'aide de la règle de la chaîne, calculer en conséquence les dérivées de la perte par rapport à ces paramètres et les concaténer au vecteur de gradient $\mathbf{g}$.
4. Passer à la couche précédente.

A la fin, le vecteur $\mathbf{g}$ contient toutes les dérivées nécessaires à une étape de descente de gradient :

$$\mathbf{g} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \vdots \\ \frac{\partial L}{\partial \theta_p} \end{bmatrix}, \tag{3.16}$$

où $L$ est la perte globale, i.e. la somme des pertes sur toutes les données d'apprentissage. Une fois ce vecteur en poche grâce à la rétro-propagation, on effectue une étape de descente :

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - \eta \mathbf{g}. \tag{3.17}$$

### 3.1.6 Apprentissage par mini-batch

Bien que l'algorithme de rétro-propagation puisse réduire le temps de calcul, son utilisation à lui seul n'est en général pas suffisant. Une variante très connue se base sur la taille de l'ensemble de données considérée en entrée. On donne par défaut l'intégralité des données aux réseaux au moment d'effectuer un pas de descente du gradient (qui utilise la rétro-propagation pour

réduire le temps de calcul du gradient **g**). Le fait d'utiliser le jeu en entier présente néanmoins quelques défauts :

— Le temps de calcul est très élevé, en raison de la quantité de données en entrée, ainsi que des opérations matricielles effectuées.
— La sensibilité aux optimums locaux : l'algorithme ne pourra converger (de manière déterministe) que vers un seul résultat, si celui-ci est en réalité un minimum local de mauvaise qualité, alors l'optimisation aura échouée.

Pour pallier ces défauts, la stratégie la plus courante consiste à ne pas donner l'entièreté des données en entrée du réseau, mais de les découper en morceaux avant de les injecter. On parle de mini-*batch* pour désigner le nombre de données sur lequel le réseau va s'entraîner à chaque étape de la descente de gradient. Ces méthodes qui ne prennent qu'un nombre réduit de données sont dites "stochastiques" et sont très utilisées dans l'apprentissage profond. Elles permettent d'obtenir les avantages suivants :

— Le temps de calcul est réduit car les opérations matricielles prennent en considération des tenseurs de taille réduite.
— Une forme de perturbation aléatoire dans l'optimisation du réseau est introduite qui, couplée avec une stratégie de sauvegarde du meilleur ensemble de paramètres (grâce à un jeu de données de validation), peut améliorer le résultat final.
— Apprendre un réseau par mini-*batch* peut parfois offrir un effet de régularisation [133] (c.f. section suivante). L'optimisation d'une stratégie stochastique demande en général d'avoir un *learning rate* très faible pour pallier le bruit élevé au sein des données.

Si on choisit un nombre limité de données pour calculer un vecteur **g** de gradient, se pose la question de quelles données choisit-on. En général, on va balayer séquentiellement l'ensemble du jeu données qui aura été séparé préalablement en petits sous-ensembles de taille égale au paramètre mini-*batch* choisi. Quand on a parcouru une fois l'ensemble de toutes les données du jeu, on a complété ce qu'on nomme une *epoch* d'apprentissage. Pour renforcer l'aléa, on peut mélanger les données à la fin de chaque *epoch*.
Quand la taille du mini-*batch* est fixée à 1, on obtient une variante appelée descente de gradient stochastique (SGD).
Pour toutes les variantes, on peut émettre le constat suivant :

1. Au début de l'apprentissage, on souhaite avoir un *learning rate* élevé pour apprendre vite.
2. Vers la fin de l'apprentissage, il faut que la taille des mises à jour du gradient se réduise pour converger efficacement.

En pratique, dans le cadre de l'algorithme d'optimisation classique de descente du gradient, il n'est pas nécessaire de réduire la valeur du *learning rate* car les gradients se réduisent d'eux mêmes au fur et à mesure de l'apprentissage. En effet, la fonction objectif étant globale, les gradients associés vont se réduire au fur et à mesure que le réseau s'optimise. Dans le cadre de la SGD, les gradients sont fortement bruités et les risques de divergences sont bien plus élevés. C'est pourquoi il peut être préférable de réduire la valeur du *learning rate* au fur et à mesure de l'apprentissage comme suit :

$$\eta_{r+1} = (1 - \alpha)\eta_0 + \alpha\eta_\tau, \tag{3.18}$$

avec $\alpha = \frac{r}{\tau}$, $\eta_0$ étant une constante généralement choisie par l'utilisateur, idem pour $\eta_\tau$ (valeur recommandée $\frac{\eta_0}{100}$), et $r$ étant l'*epoch* courante dans l'apprentissage.

Bien que ces stratégies permettent de considérablement réduire le temps de calcul, il reste cependant un défaut majeur qui est directement lié à la descente de gradient : la gestion des optimums locaux. En effet, il n'y a aucune garantie d'obtenir le minimum global de la fonction qu'on minimise car la fonction de perte d'un réseau de neurones est non convexe.

La section suivante s'intéresse en partie aux mécanismes d'optimisation de la descente du gradient, permettant dans certains cas d'aboutir sur un minimum local de meilleur qualité.

## 3.2 Optimisation avancée pour les réseaux de neurones

Plusieurs phénomènes peuvent aboutir à l'obtention d'un modèle $f_\theta$ au pouvoir généralisant très limité. Ceci peut-être causé par des minimas locaux présents dans la fonction de perte, ou bien des effets de plateaux. Des soucis liés à la disparition du gradient peuvent également en être responsable. La présente section vise à décrire dans un premier temps différents optimiseurs avancés, très utilisés et responsables en partie d'une convergence plus efficace des réseaux. Par la suite, une discussion est proposée concernant les effets de plateaux, ainsi que le phénomène de disparition du gradient.

### 3.2.1 Optimiseurs avancés

La section précédente a évoqué l'algorithme mini-*batch* SGD qui permet, en plus de réduire le temps de calcul, d'également pouvoir échapper à certains optimums locaux. On parle d'optimiseurs pour définir les algorithmes qui, comme mini-*batch* SGD, visent à modifier au cours de l'apprentissage les valeurs des gradients utilisés pour effectuer les mises à jour des poids. La littérature est vaste sur ce sujet et la section suivante a pour but de détailler les plus célèbres. Toutes ces approches sont généralement implémentées dans les bibliothèques communes de réseaux de neurones et sont donc utilisées régulièrement sans forcément que l'utilisateur ait à les définir explicitement.

#### 3.2.1.1 SGD avec momentum

L'algorithme de descente de gradient stochastique avec momentum [99] répond à la première contrainte formulée sur le *learning rate*, en réduisant sensiblement le temps d'apprentissage. Basé sur la notion de mémoire des étapes précédentes, le *learning rate* de l'algorithme va donc pouvoir se modifier afin d'accroître la vitesse de convergence. Le principe est le suivant : si un gradient est sensiblement proche de ceux des étapes précédentes, alors on

considère qu'il doit globalement bien correspondre à notre problème d'optimisation et sa valeur sera augmentée via une moyenne glissante. A l'opposé, si le gradient obtenu est complètement différent des autres gradients vu précédemment, alors il est probablement erroné, responsable d'un bruit élevé, et il sera alors réduit toujours grâce à la moyenne glissante qui permet à des gradients opposés de se compenser mutuellement. De plus, la mémoire utilisée se base sur une décroissance exponentielle, ce qui signifie que les dernières itérations (les plus récentes) auront une importance accrue dans le processus. Un terme de vélocité noté $\mathbf{v}$ est alors calculé et mémorise les derniers gradients obtenus :

$$\mathbf{v}_{r+1} = \alpha \mathbf{v}_r - \eta \mathbf{g}_{r+1}, \tag{3.19}$$

pour l'*epoch* courante de l'apprentissage notée $r$, et $\alpha$ étant le paramètre associé au momentum, ayant généralement par défaut la valeur 0,9. Pour la mise à jour, on utilisera la même règle que précédemment en remplaçant $\mathbf{g}$ par $\mathbf{v}$.

#### 3.2.1.2 RMSProp

L'optimiseur RMSProp [124] est l'un des plus utilisé actuellement. Il a la particularité de régulièrement perturber la fonction objectif qu'on souhaite minimiser, ce qui permet entre autres d'éviter certain optimums locaux de mauvaises qualités ainsi que des effets de plateaux. Le principe de l'algorithme va se baser sur deux étapes. La première, initialement proposée par adaGrad [30] consiste à augmenter la valeur des gradients de manière inversement proportionnelle à la somme cumulée des carrées de ceux-ci (composante par composante). Si un paramètre d'un neurone obtient à plusieurs reprises des gradients élevés, alors ceux-ci vont être fortement diminués. A l'opposé, un paramètre ayant des mises à jour trop faibles obtiendra une compensation. Cette mise à jour est définie par la formule :

$$\Delta\boldsymbol{\theta}_r = -\frac{\eta}{\epsilon + \sqrt{\mathbf{s}_r}}\mathbf{g}_r, \tag{3.20}$$

avec $\mathbf{s}_r$ représentant la somme des gradients de l'*epoch* 0 à $r$, $\eta$ un hyperparamètre de type *learning rate* à régler pour choisir une vitesse de convergence et $\epsilon$ une constante de faible valeur permettant d'éviter les divisions par zéro. Les opérations de division et racine carrée dans cette expression sont effectuées composante par composante.
La seconde particularité de RMSProp repose sur cette même somme des gradients. A l'instar d'une approche à base de momentum, une décroissante exponentielle est également effectuée dans la somme des gradients. Les derniers gradients vont alors grandement impacter la valeur associée à la somme.

$$\mathbf{s}_{r+1} = \alpha \mathbf{s}_r + (1-\alpha)\mathbf{g}_r \odot \mathbf{g}_r. \tag{3.21}$$

Le symbole $\odot$ représente la multiplication entre deux vecteurs composante par composante. La valeur par défaut de $\alpha$ est de 0.99, mais elle peut-être généralement modifiée par l'utilisateur dans les bibliothèques standard.

#### 3.2.1.3 Adam

L'optimiseur Adam [66] est également l'un des plus utilisé. En quelques mots, son principe combine les idées de RMSProp avec celles du momentum. Plus précisément, le numérateur de l'équation (3.20)) est à présent calculé à partir d'une moyenne glissante des gradients (selon le principe du momentum) et non plus directement à partir du dernier gradient obtenu, il y a donc un effet mémoire. Il y a de ce fait deux hyperparamètres de type taux d'affaiblissement à régler, généralement à 0.9 et 0.999 respectivement. Les valeurs du numérateur et du dénominateur vont toutes deux subir les modifications vues plus haut. Comparativement à RMSprop, la valeur associée au numérateur étant une moyenne glissante des gradients, ADAM sera donc potentiellement moins sensible aux perturbations récentes des gradients.

Ces optimiseurs ont tendance à fortement réduire le temps d'apprentissage, ainsi que de permettre d'échapper aux optimums locaux. Néanmoins, il ne faut cependant pas négliger le *learning rate* qui reste malgré tout un paramètre essentiel à régler dans l'optimisation du réseau.

### 3.2.2 Gestion des effets de plateaux

Les différents optimiseurs présentés plus haut permettent de fortement accélérer la convergence du réseau. Cependant, il est possible d'obtenir des effets de plateaux très tôt dans l'apprentissage. Afin d'y remédier, les manières les plus efficaces sont d'utiliser sur les données des stratégies de normalisation [59]. Il est également possible de jouer sur les valeurs initiales des poids du réseaux pour débloquer l'apprentissage. On pourra par exemple tirer une dizaine de valeurs initiales de $\boldsymbol{\theta}$, entraîner chaque instance du réseau parallèlement, puis prendre la valeur moyenne des vecteurs $\boldsymbol{\theta}$ ainsi obtenus ou bien retenir celle qui possède l'erreur de la validation la plus faible. On reprend alors l'apprentissage cette fois-ci d'un unique réseau que l'on mène jusqu'à convergence finale.
Si les problèmes persistent, il est également possible de mettre en place ce qui se nomme un *curriculum learning* [5, 32]. En commençant par donner des images dites "simples" au réseau, il est alors capable de mieux démarrer son apprentissage, et donc de plus facilement sortir de différents optimums locaux. C'est une approche qui peut également faire référence au comportement humain, un enfant va apprendre à réaliser des tâches simples et augmenter graduellement son niveau de complexité. Elman et Al. [32] montrent également que ce type d'approches permet d'améliorer l'apprentissage du réseau, notamment par la découverte d'optimums locaux de meilleures qualités. Le seul problème non négligeable vient de la mise en place de l'approche. En effet, il n'y a pas encore de consensus très clair pour définir ce qu'est une donnée "simple". En général, cette définition est à définir au cas par cas pour chaque problème.

### 3.2.3 Problème de la disparition des gradients

La disparition du gradient, aussi nommée "*vanishing gradient*" est un problème rencontré lors de l'apprentissage de réseaux profonds et qui peut empêcher le réseau de mettre à jour ses paramètres. Ce problème provient de la mise à jour des poids, qui est proportionnelle au poids courant, et également basée sur la dérivée partielle de la fonction d'erreur qui tend à se réduire. Avec la rétro-propagation du gradient, les gradients des poids sont décomposés en fonction des couches suivantes, et les dérivées des fonctions sigmoïdes ou tanh sont comprises dans l'intervalle $[0;1]$. Par conséquent, une couche très éloignée de la sortie du réseau va recevoir une mise à jour qui sera un produit entre plusieurs valeurs inférieures à 1. Ce problème entraîne alors des soucis d'apprentissage, pour lequel il n'est plus possible de modifier les poids de certaines couches au point que le communauté a pensé pendant un certain temps qu'apprendre un réseau profond par rétro-propagation n'était pas possible en pratique.

Les solutions pour y remédier consistent en général à réduire l'écart maximal entre une unité neuronale et la sortie du réseau. Pour ce faire, il faut inclure dans le réseau ce qu'on nomme des "*skip connections*", cela s'avère particulièrement efficace dans notre cas d'étude, les images biomédicales[29]. Le but d'une *skip connection* est de créer des liens dans le réseau entre les couches de profondeurs différentes. La profondeur d'une couche est le nombre d'itérations à faire avec l'algorithme de rétro-propagation afin de pouvoir modifier cette même couche. Si les couches les plus profondes sont reliées à la sortie du réseau, alors elles pourront obtenir des gradients ayant des valeurs bien plus élevées. L'un des réseaux les plus connus dans la segmentation d'images biomédicales, nommée U-net[105] utilise ce procédé.
Une autre astuce consiste à initialiser les poids du réseau de sorte à maîtriser la variance de la sortie des couches et éviter de visiter des valeurs pour lesquelles les fonctions d'activation ont des gradients très faibles (ce qui n'est pas le cas autour de zéro pour la sigmoïde par exemple). Glorot and Bengio [40] proposent une telle stratégie pour initialiser les poids de chaque couche en fonction du nombre de neurones des couches voisines.

## 3.3 Gestion du sur-apprentissage

Apprendre parfaitement sur un ensemble d'entraînement n'est pas suffisant pour s'assurer du bon fonctionnement d'un réseau. Il faut également vérifier sa capacité de généralisation. Le réseau doit en effet être capable d'offrir un résultat de même qualité sur des données qu'il n'a jamais vu. Si ce n'est pas le cas, alors on parle de sur-apprentissage (*overfit*), on considère alors que le réseau n'a pas généralisé la tâche souhaitée. Cette section vise à expliquer en détails quelques-unes des stratégies à mettre en œuvre pour remédier à cette situation et se prémunir du sur-apprentissage.

### 3.3.1 Régularisation $L_1$ et $L_2$

L'utilisation d'une stratégie de régularisation n'est pas propre aux réseaux de neurones mais est couramment utilisée en apprentissage, notamment pour les modèles linéaires (régression logistique, linaire etc..). L'objectif global est de limiter la puissance du réseau de sorte qu'il ne puisse pas apprendre de fonctions trop complexes. On considère alors qu'une trop grande liberté pour le réseau augmente le risque qu'il puisse apprendre une fonction trop spécifique propice à un sur-apprentissage. En limitant notre réseau, il sera obligé de capter les éléments principaux associés à la tâche demandée.

Pour effectuer la régularisation, il suffit d'ajouter une pénalité dans la fonction de perte calculée en sortie du réseau. Dans le cadre de la régularisation dite $\mathsf{L}_2$ [49], on souhaite forcer les poids du réseau à se rapprocher le plus possible de l'origine. Le réseau n'a donc plus la liberté de sélectionner des poids élevés en valeur absolue. La pénalité ajoutée sur les poids se base directement sur la valeur de ceux-ci. Pour une certaine fonction de perte $L$ d'attache aux données (c.f. 3.1.3), on utilise la norme du vecteur pour pénaliser le modèle :

$$L_{\text{reg}}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \alpha||\boldsymbol{\theta}||_2^2 \tag{3.22}$$

$\alpha$ étant le paramètre de régularisation, ayant par défaut classiquement une valeur proche de 1e-4. Ce type de régularisation va donc agir en priorité sur les poids les plus extrêmes de réseau. D'un point de vue numérique, le calcul du gradient de la nouvelle fonction de perte ne présente aucune difficulté majeure. En écrivant la pénalité sous la forme suivante : $\frac{\alpha}{2}\boldsymbol{\theta}^\mathsf{T} \cdot \boldsymbol{\theta}$, alors la dérivée n'est autre que : $\alpha\boldsymbol{\theta}$.

La seconde pénalité la plus utilisée s'appuie sur la norme $\mathsf{L}_1$, également nommée Lasso [122] et correspond à la somme des valeurs absolues des poids.

$$L_{\text{reg}}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \alpha||\boldsymbol{\theta}||_1 \tag{3.23}$$

L'avantage de cette régularisation est que les poids du modèle tendent à être plus parcimonieux, ce qui peut se traduire par un grand nombre de poids ayant une valeur après convergence à zéro. La figure 3.4 montrent les courbes de niveaux de la fonction de perte $L$ ainsi que des boules centrées en zéro de type $\mathsf{L}_1$ ou $\mathsf{L}_2$. On voit que pour la norme $\mathsf{L}_1$, l'intersection se produira en un lieu du repère où une coordonnée sera nulle.

### 3.3.2 L'augmentation de données

Afin de pouvoir augmenter la capacité de généralisation d'un modèle, une manière toute simple consiste à augmenter le nombre de données supervisées utilisées. Si ce n'est pas possible, alors il faut les générer artificiellement soit-même, c'est le principe de l'augmentation de données. Le principe consiste à modifier notre ensemble de données initial $\mathcal{S}_{\text{tr}}$ afin d'en obtenir un plus vaste. Imaginons que pour un couple $(\mathbf{x}, \mathbf{y})$ de données, composée d'une image et de son masque de segmentation, on décide d'effectuer une

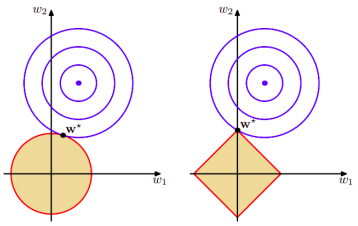


FIGURE 3.4 – Affichages des contours de la fonction d'erreur non régularisée en bleu, ainsi que les contraintes de régularisation en rouge, pour $L_1$ à droite et $L_2$ à gauche.
Source : C. M. Bishop [13]

opération de rotation sur ce couple de données. On obtient alors un nouveau couple d'images noté $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ qui pourra être utilisé dans la boucle d'apprentissage. Il est alors possible dans ce cas de figure de générer des nouvelles données à partir de notre base initiale. Ce procédé n'est malheureusement pas utilisable pour tous les problèmes car il faut connaître une transformation sous laquelle la distribution des données est invariante et une telle connaissance n'est pas toujours disponible.
Les augmentations de données les plus utilisées pour les images sont les rotations (c.f. Fig. 3.5 pour exemple), translations et ajouts de bruits. Le bruitage des données en entrée est très important car il lutte efficacement contre l'un des points faibles des réseaux de neurones : la robustesse au bruit. Cette faiblesse se traduit par le fait qu'une légère modification de l'entrée peut provoquer une sortie complètement différente. Cette faille est d'ailleurs exploitée pour mettre au point des exemples adverses.
Malgré tout, il ne faut pas utiliser inconsciemment cette stratégie, prenons par exemple le jeu de données MNIST contenant des images de chiffres manuscrits. Il est possible d'obtenir le chiffre 6 en retournant un 9 ce qui est contre-productif.

Dans le cadre de la segmentation d'images de cellules, qui est le sujet majeur de nos travaux, l'augmentation qui consiste à effectuer des déformations élastiques sur nos données est relativement importante. En effet, modifier légèrement le contour d'une cellule nous permet d'en obtenir une nouvelle. Il est ainsi possible de générer une quantité presque illimitée de données présentant de nouvelles cellules, ce qui ne signifie pas pour autant qu'il est possible d'atteindre une généralisation parfaite puisque les données générées sont dépendantes de celles de départ. Le chapitre suivant propose une stratégie d'optimisation du choix des types d'augmentation à effectuer et reviendra plus en détail sur ces concepts.

### 3.3.3 Dropout

La méthode de *Dropout* [117] permet d'effectuer efficacement de la régularisation au sein d'un réseau de neurones. Le principe est simple, il consiste à annuler temporairement le poids de différents neurones sélectionnés (choisis

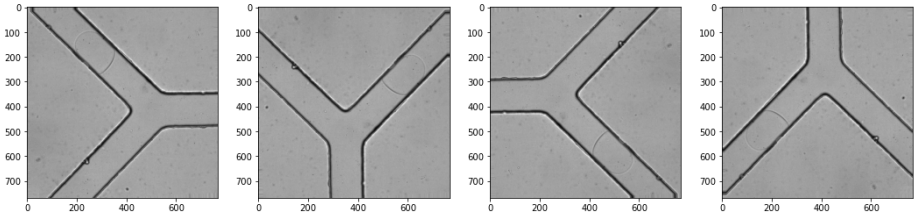

FIGURE 3.5 – Utilisation de rotations (90,180 et 270) sur une image de capsule dans un tube bi-directionnel.

aléatoirement) au cours de l'apprentissage, et de répéter le procédé en changeant les neurones (c.f. Fig. 3.6 pour illustration). Le réseau doit donc être capable de résoudre la tâche demandée pour tous les sous-réseaux possibles, un sous-réseau étant le réseau initial avec la perte des poids choisie aléatoirement. Bien que l'idée de moyenner la sortie de plusieurs sous-réseaux à l'instar d'une stratégie de *bagging* classique permet généralement d'améliorer le résultat final [123], cela n'est pas raisonnable en terme de temps de calcul. L'algorithme de rétropropagation peut-être utilisé directement sur les sous-réseaux, en ne modifiant uniquement que les poids encore valides, ce qui permet un gain de temps non négligeable.

Les travaux de Srivastava et Al. [117] montrent également que le *dropout* permet d'améliorer la robustesse au bruit. En effet, annuler la valeur d'un neurone du réseau peut-être vu comme bruiter fortement les couches adjacentes à celui-ci. Le réseau étant entraîné via cette stratégie est donc plus robuste aux petites perturbations de l'entrée. Il s'agit aussi d'une forme indirecte de régularisation car en l'absence d'un neurone, si des poids trop élevés lui sont attachés, une erreur sera probablement commise par le réseau. Ce dernier est donc contraint de ne pas affecter de poids trop élevé sur un neurone.

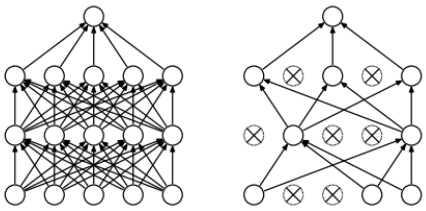

FIGURE 3.6 – Affichage des connections d'un réseau avec et sans *dropout*.
Source : Nitish Srivastava et Al. [117]

### 3.3.4 Early Stopping

L'algorithme d'*early stopping* est très utilisé car il est simple à mettre en œuvre et permet de concurrencer les stratégies de régularisation [34]. Le principe de l'algorithme se base sur le fait qu'un réseau ayant trop de capacité pour une tâche donnée va finir par augmenter son erreur de validation au cours de l'apprentissage. Par erreur de validation, on entend la valeur de la fonction de perte calculée sur un ensemble de données $\mathcal{S}_{\text{val}}$ disjoint de $\mathcal{S}_{\text{tr}}$ et réservé

pour l'occasion. Par conséquent, il est intéressant de sauvegarder les poids du réseau à chaque étape de l'apprentissage, afin de pouvoir récupérer ceux ayant obtenus l'erreur de validation la plus basse. Il est également possible d'arrêter prématurément l'algorithme. Si après de multiples itérations, l'erreur de validation ne baisse plus, alors le réseau est probablement en train de sur-apprendre et l'apprentissage peut s'arrêter. Sur la courbe de droite de la Figure 3.7, on constate que l'erreur de validation finit par remonter, il est alors temps d'arrêter l'apprentissage. A l'inverse, celle de gauche, à savoir l'erreur sur l'ensemble d'apprentissage continue de baisser car le sur-apprentissage commence.

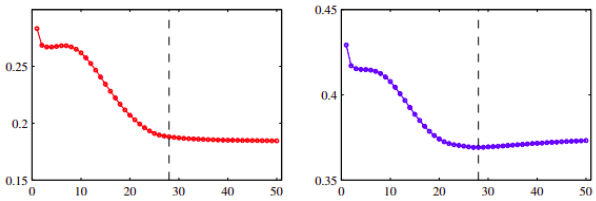


FIGURE 3.7 – Fonction d'erreur pour l'ensemble de d'apprentissage à droite, et de validation à gauche. On souhaite récupérer les poids de notre modèle au moment ou l'erreur de validation augmente, ce qui correspond à l'intersection de l'erreur avec la droite verticale en pointillée.
Source : Cristopher M.Bishop [13]

### 3.3.5 Batch normalisation

La *batch* normalisation [55] est un mécanisme très robuste pour augmenter la généralisation et réduire également le temps d'apprentissage. Il repose sur le fait qu'un phénomène dit de *covariate shift* [110] entre les différentes couches du réseau perturbe généralement l'apprentissage. Le *covariate shift* se traduit en général par le fait que les données d'entraînement sont bien différentes de celles de test, ce qui entraîne des difficultés pour généraliser. Dans notre cas, le *covariate shift* apparaît au sein du réseau entre les entrées et les sorties de chaque couche. En effet, en mettant à jour les poids d'une couche, on modifie la distribution de la sortie de cette dernière. Or la couche suivante a été optimisée pour la distribution avant que cette modification n'intervienne. Par conséquent, normaliser l'entrée de toutes les couches permet de réduire ce phénomène en forçant une homogénéité au sein des valeurs qui parcourent le réseau.

## 3.4 Apprentissage profond pour la vision

L'apprentissage de réseaux de neurones dit classiques (type MLP) présentent des difficultés quand on l'applique sur des images. On peut citer notamment le temps de calcul qui a tendance à exploser car la dimension de l'entrée est très élevée. A savoir qu'une image en haute définition propose plusieurs millions de valeurs en entrée du réseau. Pour faire face à cette difficulté, Lecun

et al. [72, 73] proposent un nouveau type d'architecture : les réseaux convolutifs. La particularité de ces réseaux provient de l'utilisation de nouvelles couches, dont nous allons détailler le fonctionnement dans les sous-sections suivantes.

### 3.4.1 Couche de convolution

La couche la plus importante au sein des réseaux de neurones convolutifs est celle de convolution. Le principe consiste à apprendre un filtre de convolution et l'appliquer à l'image en entrée. Cela permet entre autres d'apprendre uniquement les valeurs optimales du noyau de convolution, ce qui réduit fortement le nombre de paramètres. De plus, ce même nombre de paramètres n'augmente pas avec la taille de l'entrée. L'application de ce noyau sur l'image en entrée est une simple opération de convolution (expliquée en partie 2.1.2). Le réseau va alors enregistrer l'image de sortie de l'opération de convolution, et elle sera donnée à la couche suivante. En général, il est nécessaire d'appliquer plusieurs filtres de convolution en parallèle sur l'entrée et de stocker toutes les images résultantes (tenseur 3D). Le réseau va enregistrer de nombreuses images de sortie qui seront directement envoyées à la couche suivante. La mémoire utilisée par le réseau reste alors importante, bien que le nombre de paramètres à apprendre se trouve fortement réduit.
En résumé, l'utilisation de ce filtre présente les avantages suivants :

1. Le nombre de paramètres à apprendre est fortement réduit, ce qui réduit les risques du sur-apprentissage ainsi que le temps de calcul. En général, il n'est pas nécessaire d'apprendre plusieurs millions de paramètres issus de la multiplication entre l'image initiale (de très grande dimension) et la première couche du réseau. Des éléments de l'image comme des contours précis, des irrégularités de faible taille peuvent permettre de résoudre la tâche souhaitée.
2. Le partage de paramètres est également très utile car les données présentent des corrélations spatiales. L'opération de convolution permet d'utiliser le même noyau sur toute l'image, ce partage de paramètres fait sens car si l'objectif du noyau est de repérer des *features* saillantes de tailles réduites, alors il semble logique de vouloir parcourir toute l'image avec un filtre capable de repérer la dite *feature*.
3. Le filtre de convolution est également invariant à des opérations de translations, ce qui le rend plus robuste.

### 3.4.2 Couche de pooling

La couche de *pooling* est généralement placée entre plusieurs couches de convolution au sein du réseau. Son objectif est de drastiquement réduire la taille des images intermédiaires stockées dans le réseau. Comme vu précédemment, le filtre de convolution va appliquer plusieurs noyaux, et va obtenir de nombreuses images en sortie. Le stockage de ces images peut causer des problèmes de mémoire. C'est pourquoi il faut réduire la taille de ces images, notamment par le biais de l'utilisation d'une couche de *pooling*. Le cadre le plus courant consiste à diviser la taille de l'image en supprimant

simplement des valeurs. La couche de max-*pooling* va par exemple conserver le maximum de chaque voisinage de pixels, là où l'*average-pooling* va calculer la moyenne des valeurs. La taille du voisinage de pixel est le paramètre le plus important à régler pour une telle couche. Un second paramètre fait en sorte que les voisinages constituent une partition complète de l'entrée de la couche, mais il reste tout de même possible d'effectuer un chevauchement entre les voisinages.

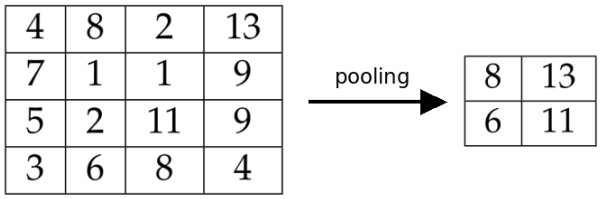


FIGURE 3.8 – Application d 'une couche de max-*pooling* pour des voisinages de taille $2 \times 2$ et sans chevauchement.

### 3.4.3 Réseau de neurones convolutifs

Les réseaux de neurones convolutifs classiques sont constitués d'un assemblage de couches de convolution, couplés à des couches de *pooling*. Pour cette partie du réseau, on peut parler d'architecture encodeur (*encoder*) car la dimension de l'image initiale sera réduite au fur et à mesure qu'elle traversera le réseau. De plus, les informations relatives à la résolution du problème seront alors encodées à travers les filtres les plus avancés du réseau. Enfin, le réseau ne se résume pas uniquement à l'application successive de ces deux types de couches. En effet, une partie de perceptron multicouche va se concaténer à la suite du réseau afin de pouvoir résoudre la tâche de classification directement à l'aide de la représentation simplifiée de faible dimension du problème fournie par l'encodeur.

Ce type d'architecture qui se compose d'une partie encodeur suivie d'une partie de couches entièrement connectées est très utilisé et correspond à celui initialement proposé par Lecun et al. [74], c.f. fig. 3.9 pour illustration. La différence entre ces réseaux, souvent désignés par l'acronyme anglais CNNs, se situe non pas sur la structure, mais sur le nombre de couches, de filtres, de paramètres à apprendre ainsi que la réduction de résolution du flux d'informations qui parcourt le réseau. Enfin, l'ensemble de données en entrée est également à prendre en compte. L'une des plus célèbres architectures se nomme VGG16 [115], bien qu'il en existe de nombreuses autres. Cette architecture est proposée par les bibliothèques connues et peut être directement récupérée moyennant de faibles efforts, ce qui facilite l'utilisation de stratégie à base de *Transfer Learning* [93].

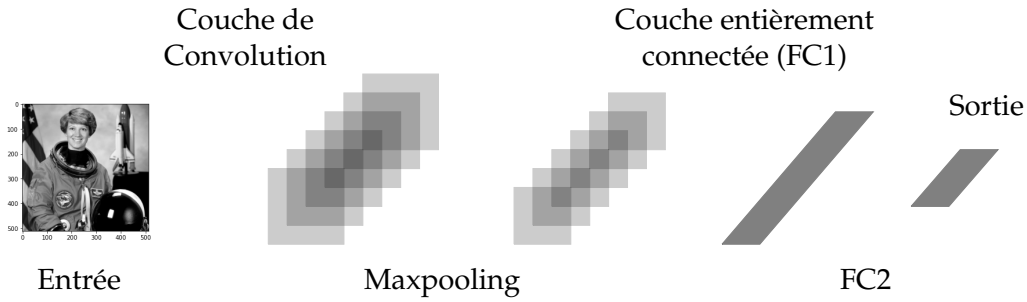


FIGURE 3.9 – Schéma d'un CNN classique ou l'image d'entrée va subir des opérations associées à des couches de convolution et de max-*pooling* avant une partie entièrement connectée, reliée à la sortie du réseau.

## 3.5 Segmentation à base de réseaux de neurones

Les architectures encodeur permettent d'extraire des informations saillantes des données afin de pouvoir résoudre une tâche de classification. Pour passer à la segmentation, il faut cependant obtenir en sortie du réseau une image de la même dimension que celle située à l'entrée. Par conséquent, la phase de réduction de la résolution de l'image doit être compensée par une phase de rehaussement. On parle alors d'architecture encodeur-décodeur pour définir les réseaux encodeur ayant également une phase miroir nommée décodeur qui va naturellement augmenter la résolution du flux d'information au sein du réseau. De cette manière, une partie décodeur va alors être ajoutée et aura pour objectif de résoudre la tâche de segmentation à l'aide du flux d'informations fourni par la partie encodeur de l'architecture.

### 3.5.1 Couches pour la segmentation

La section suivante se consacre à l'explication des différentes couches les plus communes pour la création de la partie décodeur du réseau.

#### 3.5.1.1 Couche de ré-échantillonnage

La couche de ré-échantillonnage a pour effet d'agrandir la dimension d'une image. De nouveaux pixels vont alors être créés à partir d'une méthode prédéfinie nommée interpolation. Il existe deux interpolations très utilisées, celle du plus proche voisin et la bilinéaire.

Dans le cadre de l'interpolation du plus proche voisin, l'algorithme détermine la valeur recherchée comme étant égale à la valeur du pixel le plus proche, sans considérer les autres valeurs connues. C'est une technique simple d'application dans le traitement d'images, mais déconseillée car elle crée un fort effet de crénelage. Visuellement, cette interpolation semble effectuer un grossissement des pixels, comme le montre la figure 3.10.

L'interpolation bilinéaire [17] consiste à calculer la valeur d'une fonction en un point quelconque, à partir de ses deux plus proches voisins dans chaque

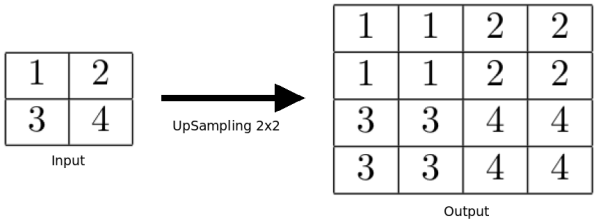


FIGURE 3.10 – UpSampling 2x2 d'une image

direction. C'est une méthode très utilisée qui permet d'obtenir de meilleurs résultats que l'interpolation par plus proche voisin, tout en restant de complexité raisonnable. Supposons que nous cherchons à multiplier par deux la résolution. Les pixels de l'ancienne grille se trouvent séparés d'un cran sur la nouvelle grille. Il faut donc combler les "trous" dans cette nouvelle grille. Intéressons nous aux quatre pixels $x_{11}, x_{13}, x_{31}$ et $x_{33}$ pour le moment connus. Il nous manque 5 pixels dans le voisinage $3 \times 3$ définis par ces 4 pixels connus. Pour déterminer leurs valeurs, la méthode propose de se placer dans un repère 2D continu $(u, v)$ où, le plus souvent les 4 pixels connus correspondent aux coins du carré unitaire. De plus, pour tout point $(u, v)$ dans ce carré unitaire, on prend comme modèle pour toute valeur de pixel situé entre nos 4 points la fonction :

$$f_{\text{bilin}}(u, v) = au + bv + cuv + d,$$

avec $a, b, c$ et $d$ 4 inconnues à déterminer. Pour les trouver, on possède 4 équations :

$$\begin{cases} f_{\text{bilin}}(0,0) = x_{11} & = & d \\ f_{\text{bilin}}(1,0) = x_{31} & = & a + d \\ f_{\text{bilin}}(0,1) = x_{13} & = & b + d \\ f_{\text{bilin}}(1,1) = x_{33} & = & a + b + c + d \end{cases}$$

qui constituent un système facilement soluble.
La Figure 3.11 montre le résultat d'une interpolation bilinéaire de dimension $2 \times 2$ à partir d'une image carrée de 4 pixels.

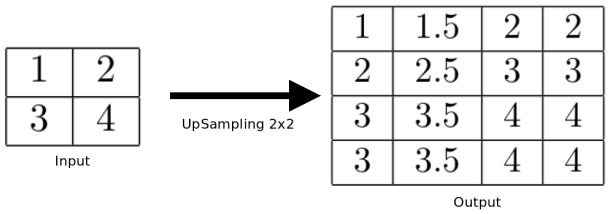


FIGURE 3.11 – UpSampling 2x2 bilinéaire d'une image

#### 3.5.1.2 Couche de déconvolution

Une seconde couche permettant d'augmenter la dimension d'une image se nomme la déconvolution (*Transpose Convolution*).
La procédure de calcul d'une couche de convolution inverse est identique à celle d'une couche de convolution. Au lieu d'utiliser une fonction d'interpolation connue comme dans le cadre de la couche de ré-échantillonnage, l'idée générale de la déconvolution est d'apprendre le meilleur "filtre" permettant d'augmenter la résolution de l'image. La couche de déconvolution va donc ajouter des paramètres à notre modèle, ce qui va rendre plus complexe la phase d'apprentissage. Il existe deux paramètres déterminants pour cette couche, la taille de noyau et la taille des *strides* [1].

La taille du noyau définit le nombre de paramètres que va apprendre notre modèle ainsi que la dimension de sortie de la couche. Pour une image de dimension $(H \times W)$, et pour un noyau de dimension $(k \times k)$, la taille de la sortie sera égale à $(H + k - 1) \times (W + k - 1)$. La figure 3.12 montre l'application d'une couche de déconvolution sur une image. Dans un premier temps, une opération nommée *padding* est appliquée à l'image.

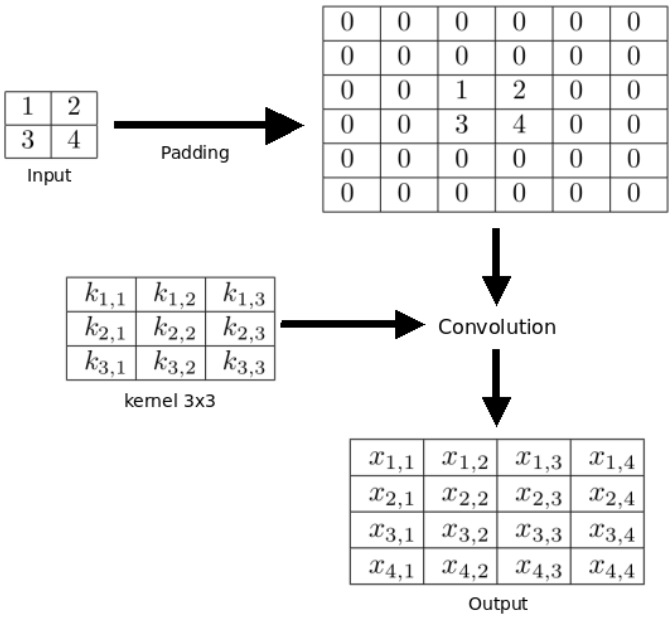


FIGURE 3.12 – Convolution Inverse avec un filtre $3 \times 3$

Le *padding*, également présent dans les couches de convolution, a pour effet d'ajouter un contour la plupart du temps rempli par des valeurs pixéllaires nulles. Le seconde étape de l'opération de déconvolution consiste à effectuer une convolution entre le noyau de convolution et l'image résultante de l'opération de *padding*, ce procédé est identique à celui de la couche de convolution. En définissant un niveau de *padding* de manière à ce que l'image ne

1. Ce paramètre non évoqué précédemment permet de définir un saut entre les étapes de convolution.

change pas de taille, alors la dimension finale de l'image se fait via le paramètre associé aux *strides*.

Les *strides* définissent la portée d'activation du noyau sur l'image. Ils indiquent le nouvel espacement entre chaque pixel de l'image.

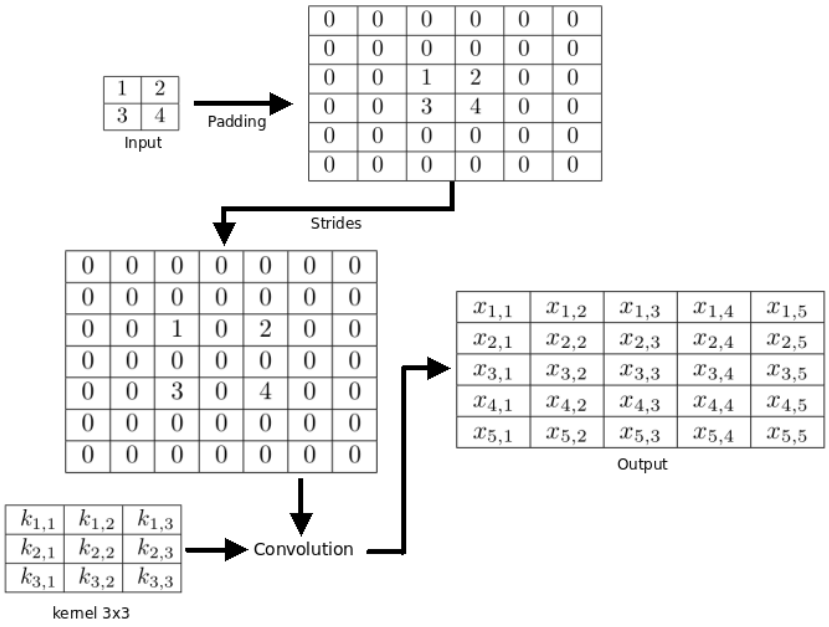


FIGURE 3.13 – Convolution Inverse avec un filtre $3 \times 3$ et des *strides* $2 \times 2$

Dans la figure 3.13, on constate que les paramètres de *strides* ont augmenté la dimension finale de notre image d'une manière différente. Il est également possible de jouer sur ce paramétrage pour viser une résolution particulière en sortie de couche.

### 3.5.2 Architecture de segmentation connues

Cette section vise à présenter trois des plus célèbres architectures ayant pour cible une tâche de segmentation qui est le problème d'intérêt de ces travaux de thèse. Bien que les architectures utilisent des couches spécifiques, cela n'est en aucun cas obligatoire. Il est possible de modifier par exemple certaines couches d'une architecture précise, afin d'en obtenir une nouvelle tout aussi performante. Il faut donc bien garder à l'esprit que ces architectures présentent certaines souplesses quant à leurs modifications.

#### 3.5.2.1 Réseau entièrement convolutif (Fully Convolutional Networks)

L'architecture dite entièrement convolutive [81] est une architecture encodeur-décodeur n'utilisant aucune couche entièrement connectée. Ce type d'architecture est très exploité car elle permet d'avoir un nombre de paramètres à apprendre qui ne dépend pas de la taille de l'entrée. On peut alors réutiliser le réseau sur des données ayant des tailles différentes. Dans ce cas, la

structure du réseau basée sur des couches de convolution doit permettre de garder de bonnes performances, pourvu que la nouvelle tâche de segmentation présente des similarités avec celle ayant servie pour l'apprentissage. La grande majorité des architectures de segmentation sont de ce type.

Dans le papier original de l'architecture proposé par Jonathan Long et al. [81], des couches de déconvolution sont utilisées dans la partie décodeur du réseau. On obtient alors une architecture composée de trois couches : convolution, *pooling* et déconvolution. Grâce aux déconvolutions successives, on obtient une sortie $\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$ qui est une image de même taille que $\mathbf{x}$. On peut alors utiliser une fonction de perte de type MSE entre $\hat{\mathbf{y}}$ et $\mathbf{y}$.

#### 3.5.2.2 U-net

L'architecture U-net [105] revêt une importance particulière dans ce manuscrit car elle est populaire grâce à ses différentes applications sur des images de cellules ou biomédicales plus généralement. C'est donc le cas d'utilisation qui la rend naturellement attrayante. L'architecture complète est décrite dans la figure 5.4, on peut constater que la partie encodeur est une simple série de couches de convolution et de *pooling*. Cependant, l'ajout de la partie décodeur se fait avec deux différences notables.

— Les couches dite de "UpConv" sur le graphe sont en réalité des couches de ré-échantillonnage.
— Des couches de concaténation permettent de relier les flux de l'encodeur et du décodeur respectivement. Cela permet entre autres de lutter efficacement contre le problème de disparition du gradient (voir section 3.2.3). De plus, ces concaténations permettent au réseau d'ignorer les couches les plus profondes si la tâche demandée ne nécessite pas de les utiliser.

L'architecture originale présente près de trente-trois millions de paramètres et se base sur une augmentation de données massive afin de pouvoir obtenir les meilleures performances. En pratique, il n'est pas toujours nécessaire d'avoir autant de paramètres, c'est pourquoi les différents réseaux U-net que nous utiliserons par la suite n'en contiendront qu'un nombre compris entre cent-mille et un million.

#### 3.5.2.3 Segnet

L'architecture Segnet [2] est également très connue dans le cadre de la segmentation. L'originalité de cette architecture provient de deux aspects. L'utilisation d'une normalisation du *batch* présentée en section 3.3.5 ainsi que l'utilisation de couches dites de reverse max-*pooling* pour inverser le processus des couches de *pooling*. L'objectif de ces nouvelles couches est très simple, il consiste à retenir l'emplacement exact des valeurs choisies par les couches de max-*pooling* dans la phase d'encodage du réseau. Ainsi il est possible lors de la phase de décodage, de remettre aux bonnes places respectives les valeurs présentes, et d'ajouter des zéros autour.

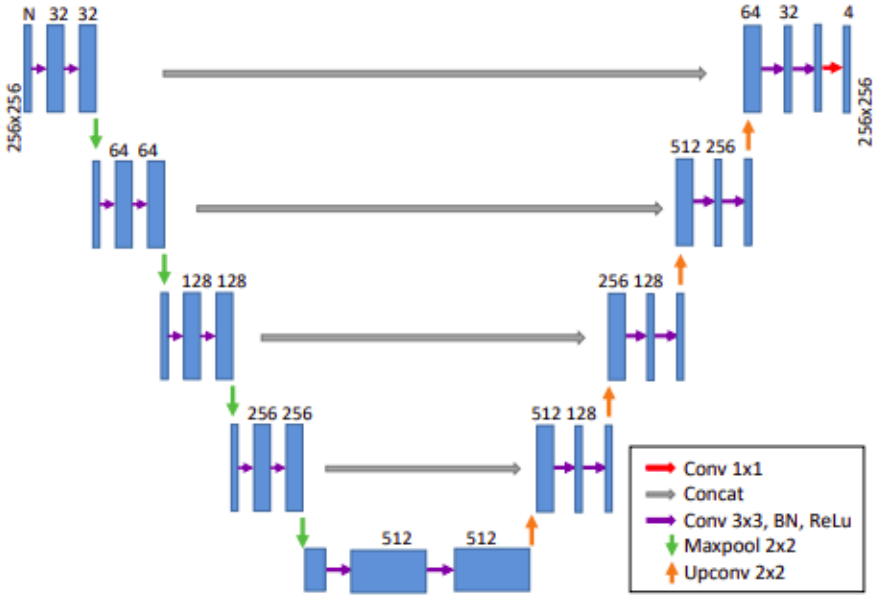


FIGURE 3.14 – Schéma de l'architecture U-net.
Source : Nahian Siddique [94]

Ce type de couches n'est cependant pas géré nativement par les librairies d'apprentissage profond. En effet, chaque couche d'un réseau est généralement considérée comme indépendante afin de pouvoir offrir une grande liberté dans la création de réseaux. Lier cette nouvelle couche à son miroir de la partie encodeur demande alors des efforts d'implémentation supplémentaires.

### 3.5.3 Variantes de l'architecture U-net

Cette section vise à expliquer brièvement deux architectures récentes qui étendent l'architecture U-net. Il ne sera cependant pas possible de détailler un plus grand nombre d'architectures dans ce manuscrit, c'est pourquoi il est conseillé de se référer à Siddique et al. [113] pour obtenir de plus amples informations. Ce focus sur l'architecture U-net n'est pas sans raison, les différents travaux réalisés se basent essentiellement dessus dans les deux chapitres de contribution qui suivront le présent chapitre.

L'architecture U-net++ [141] propose de jouer sur un plus grand nombre de *skip connections*. Dans la variante proposée par Peng et al. [96], une implémentation récursive de l'architecture est également proposée. Les explications vont se baser sur ce second article, dont le schéma de l'architecture est montrée en figure 3.15.
La partie associée à la liaison entre les couches de l'encodeur et du décodeur est complètement modifiée. Au lieu d'utiliser un système de concaténation pour relier les deux parties, U-net++ va directement ajouter des neurones. Ces neurones ajoutés entre les deux parties du réseau impliquent une forte augmentation de *skip connections*. Cela permet également au réseau d'obtenir

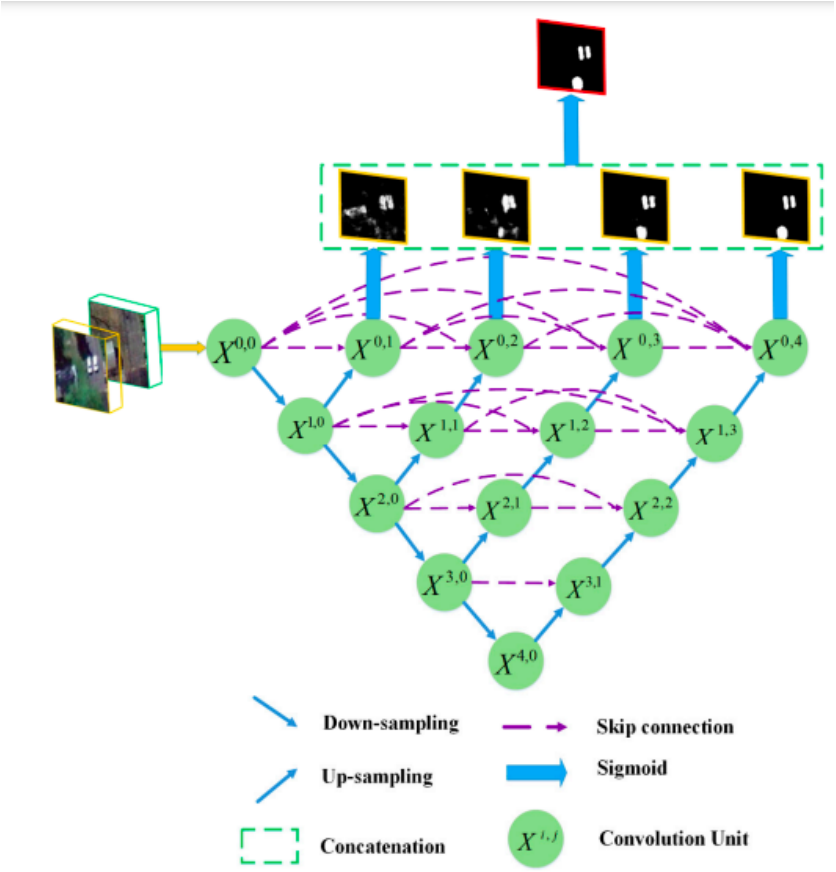


FIGURE 3.15 – Architecture U-net++ avec une fonction de perte cumulée pour chaque sortie des sous-réseaux.
Source : Peng et al. [96]

une plus grande liberté d'encodage. Cette liberté sera cependant contrebalancée par des contraintes ajoutées sur la fonction de perte.

Comme le montre la figure 3.15, le réseau est partitionnée en différents sous-réseaux. La couche de sortie nommée $X^{0,1}$ peut être assimilée au plus petit sous-réseau ne possédant que 3 séries de convolutions et une seule base de *pooling/upsampling*. La couche de sortie nommée $X^{0,2}$ est alors assimilée au second plus petit réseau. Ce procédé récursif intervient jusqu'à la taille finale du réseau. Par conséquent, la fonction de perte à minimiser n'est pas uniquement celle de la toute dernière couche du réseau ($X^{0,4}$), mais prend en compte toutes les couches des sorties précédentes. Dans ce cas, chaque sous-réseau va alors chercher à résoudre le problème de segmentation, ce qui va brider naturellement les réseaux de plus grande taille. Puisque chaque sous-réseau propose un masque de segmentation, une dernière couche de convolution $1 \times 1$ est nécessaire pour agréger ces masques en un seul.

Une seconde architecture basée sur U-net et également sur Resnet [44] se nomme Residual U-net (ResU-net). L'idée générale du réseau est également d'échapper aux soucis liés à la disparition du gradient en ajoutant des *skip*

*connections*. Ce procédé assez répandu est bien différent du cas de U-net++. En effet, cette fois-ci, des concaténations entre les données sont faites à l'échelle des couches de convolution. Les couches de convolution vont alors être encapsulées par ce qu'on appellera un bloc de convolution. Ce bloc va ajouter à la sortie de la dernière couche, l'entrée de la première. Un lien est alors créé entre chaque entrée et sortie des blocs de convolution (c.f. fig. 3.16).

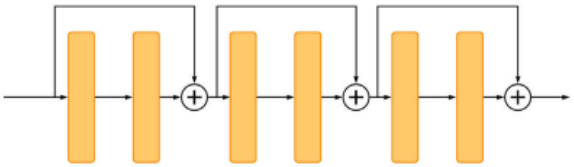

FIGURE 3.16 – RESU-net : Affichage des blocs de convolution, constitués de deux couches en orange, dont l'entrée et la sortie sont additionnées pour former le bloc.
Source : [113]

Cette architecture possède de nombreuses améliorations dont nous n'aurons pas le temps d'évoquer les détails dans ce manuscrit. La plus célèbre reste Dense U-net[52], qui a pour principale spécificité de modifier les blocs. Les entrées et sorties des blocs vont être concaténées, et les couches d'un même bloc vont recevoir des informations des couches précédentes.

De nombreuses architectures ont pour objectif de lutter efficacement contre le problème de la disparition du gradient. Cela n'est pas sans raison, de nombreuses architectures évoluées sont en réalité composées d'une combinaison de nombreux réseaux. Le souci de profondeur de chaque couche est alors un élément crucial. Le chapitre 5 de ce manuscrit traite d'une stratégie basée sur la combinaison de 3 réseaux, qui sans l'utilisation de *skip connections*, ne peut en aucun cas être entraînée correctement.

## 3.6 Autoencodeur

Cette section est dédiée à l'explication d'un nouveau type d'architecture nommé autoencodeur. Bien que ce type d'architecture ne permet pas initialement d'effectuer une segmentation, le principe d'autoencodeur sera réutilisé et étendu dans la stratégie d'apprentissage semi-supervisée longuement détaillée dans le chapitre 5. C'est pourquoi cette partie du manuscrit vise à expliquer brièvement les points clés de ces architectures afin de donner tous les éléments nécessaires à la bonne compréhension des chapitres suivants. Pour obtenir de plus amples précisions, il est préférable de se référer au document de Tschannen et al. [125].

### 3.6.1 Définition générale de l'autoencodeur

Un autoencodeur est un réseau de neurones ayant une structure encodeur-décodeur et qui a la particularité d'avoir la même sortie que son entrée. On

peut alors noter $(\mathbf{x}, \mathbf{x})$ le couple de données en entrée-sortie du réseau. On peut également reconsidérer le problème général d'apprentissage par la formule $\mathbf{x} \approx f_{\theta}(\mathbf{x})$.

Il peut sembler tout à fait normal au premier abord de questionner l'intérêt de ce type d'architecture qui ne semble rien apprendre à part la fonction identité. En effet, récupérer la sortie du réseau n'est généralement pas utile, l'intérêt repose essentiellement sur la récupération des représentations intermédiaires stockées par le réseau. Il est possible de forcer de nombreuses contraintes sur le réseau ce qui l'oblige à apprendre une représentation exploitable (et surtout de faible dimension) par la suite. De plus, ce type d'architecture s'affranchit complètement de l'utilisation de données étiquetées. Nous allons ensuite décrire trois des utilisations d'autoencodeurs les plus connues.

### 3.6.2 Autoencodeur incomplet

L'autoencodeur dit incomplet (*undercomplete autoencoder*) est un autoencodeur qui a la particularité de posséder une représentation intermédiaire de très faible dimension (c.f. fig. 3.18 pour illustration). On parle de représentation intermédiaire pour définir le flux d'informations qui parcourt le réseau. Celui-ci commence à partir de la sortie de la première couche du réseau, et termine à la couche de sortie. L'objectif est de forcer cette représentation, idéalement placée entre la partie encodeur et décodeur, à être de très faible dimension. De cette manière la partie décodeur du réseau devra résoudre la tâche demandée à partir d'un nombre infime d'informations, ce qui rend la tâche bien plus complexe à résoudre. Ce qu'on vise avec ce type d'architecture, c'est d'obtenir une représentation intermédiaire de meilleure qualité. La partie encodeur doit logiquement être capable d'en fournir une excellente pour que le décodeur puisse résoudre la tâche souhaitée. Il peut-être intéressant d'exploiter ce type de représentation pour par exemple, faire de la réduction de dimension.

L'entraînement de ce type d'autoencodeur repose sur de l'apprentissage non supervisé. Les données sans étiquetage sont en général nombreuses, et le temps de calcul est également très élevé avant de converger. Concernant la taille de la représentation intermédiaire, le but est de la réduire drastiquement. Par exemple, si on souhaite reconstruire une image contenant un rectangle (voir figure 3.17), un humain pourrait le faire avec uniquement 4 valeurs, pour encoder les deux points de l'une des diagonales. On souhaite alors que le réseau puisse faire la même chose.
En pratique, nos différentes expériences nous ont montré qu'il faut guider fortement le réseau pour qu'il encode de la même manière le rectangle (on peut notamment ajouter une contrainte dans la fonction de perte).

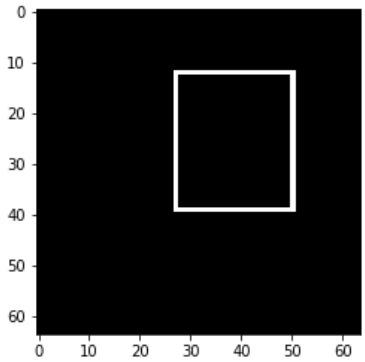


FIGURE 3.17 – Image contenant un rectangle

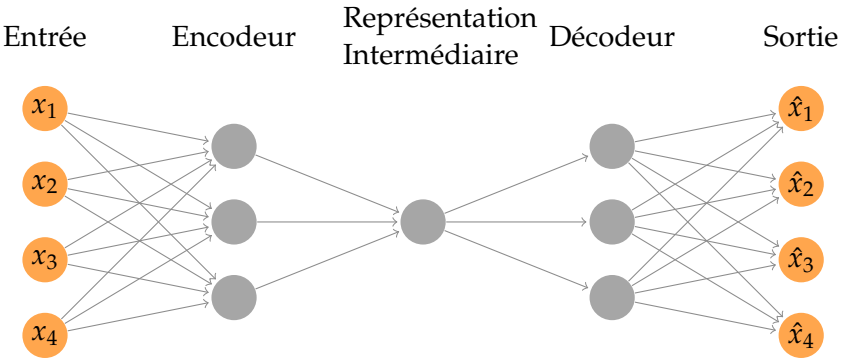


FIGURE 3.18 – Architecture d'un autoencodeur incomplet

### 3.6.3 Autoencodeur variationnel

L'autoencodeur variationnel (VAE) [67] n'est pas bien différent du modèle originel dans sa structure mais l'est sensiblement dans le formalisme aboutissant à sa définition. Sans rentrer dans les détails, un VAE utilise un générateur aléatoire entre l'encodeur et le décodeur. Généralement, on souhaite échantillonner un vecteur aléatoire gaussien $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \text{Diag}(\boldsymbol{\sigma}))$. Les sorties de l'encodeur correspondent aux vecteurs $\boldsymbol{\mu}$ et $\boldsymbol{\sigma}$. Les sorties du décodeur seront aussi de tels paramètres d'une autre loi gaussienne censée être celle des entrées $\mathbf{x}$.
La fonction de perte utilisée pour cet auto-encodeur stochastique est tirée d'une analyse probabiliste faisant appel à l'inférence variationnelle.
Enfin pour le calcul de rétro-propagation du gradient, il n'est pas possible de dériver directement l'expression pour une variable effectuant un échantillonnage, c'est pourquoi il faut re-paramétrer la variable aléatoire selon :

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \mathcal{N}(0, \mathbf{I}), \tag{3.24}$$

avec $\mathbf{I}$ la matrice identité.
La figure 3.19 montre le résultat de l'apprentissage d'un VAE sur le dataset MNIST quand la dimension de $\mathbf{z}$ est fixée à 2. La sous-figure du haut montre quels chiffres manuscrits sont encodés dans différentes positions (espacées

régulièrement) dans l'espace où vivent les $\mathbf{z}$. Celle du bas montre les encodages de différents exemples en fonction de leur classe. On constate que, sans supervision, les classes sont plutôt bien réparties dans des zones distinctes.

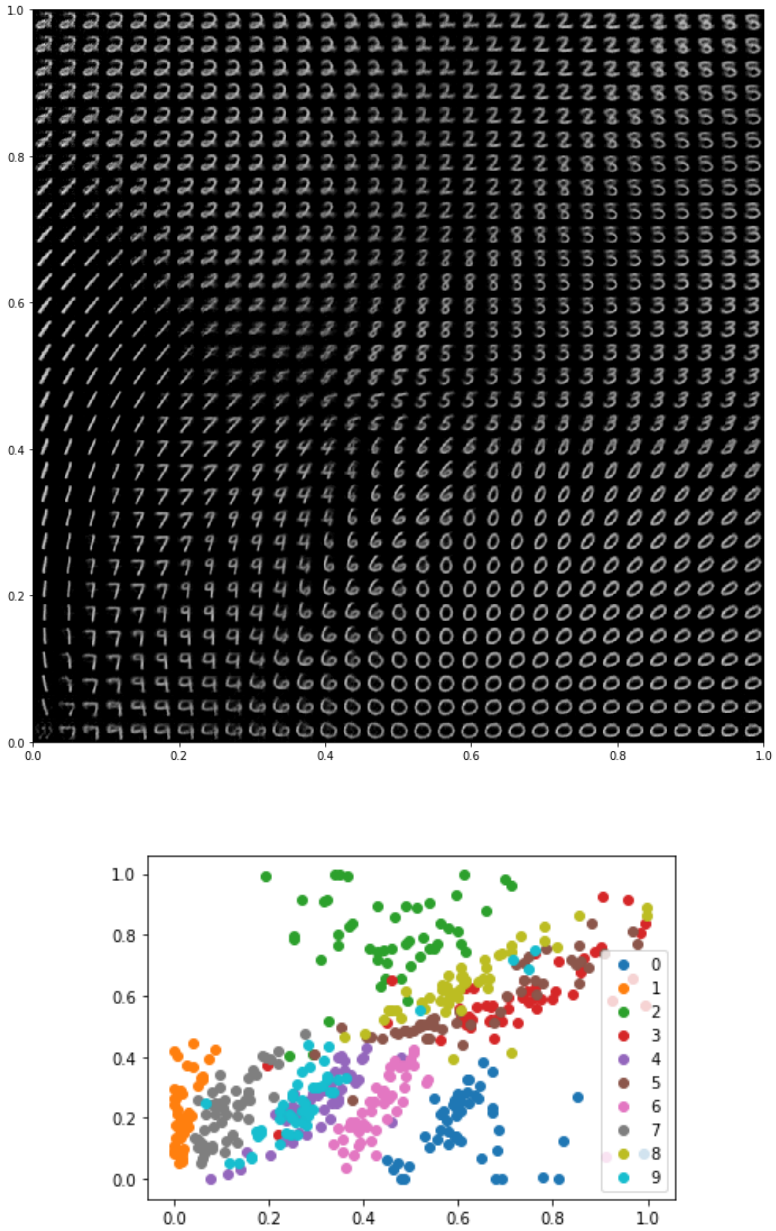


FIGURE 3.19 – Résultat d'un VAE ayant appris sur le dataset MNIST. La figure du dessus montre le résultat du décodeur en fonction des deux coordonnées de la représentation intermédiaire. La figure du dessus est un nuage du point basé sur la même expérience.

### 3.6.4 Autoencodeur de débruitage

L'autoencodeur de débruitage (DAE) [129] est un autoencodeur qui repose essentiellement sur l'utilisation d'une augmentation de données. A partir

d'une fonction noté $g$ qui permet de corrompre une donnée, il est possible de radicalement modifier la nature du problème que doit traiter l'autoencodeur. Dans ce cas précis, il doit être capable de retrouver la donnée avant qu'elle ne soit corrompue, le problème d'apprentissage peut alors être défini par $\mathbf{x} \approx f_{\theta}(g(\mathbf{x}))$. Bengio et al [6] montrent qu'utiliser ce procédé permet de forcer l'autoencodeur à apprendre la structure globale de la distribution des données en entrée, ce qui limite les possibilités de sur-apprentissage.

Il est possible de coupler ce procédé avec des autoencodeur dits incomplets afin de pouvoir ajouter des contraintes supplémentaires sur la représentation intermédiaire à calculer. Tout comme dans le cadre de la segmentation, l'utilisation d'une augmentation de données massive est encore une fois de mise. De nombreuses architectures de ce type tournent alors sur un vaste corpus d'images non étiquetées avec l'utilisation d'une augmentation de données, ce qui demande des ressources de calculs considérables.

## 3.7 Conclusion

Ce chapitre s'est concentré principalement sur les fondements de l'apprentissage profond. Après avoir défini les différentes briques qui compose un réseau de neurones, nous avons également pu discuter de la mise en place d'un modèle permettant la segmentation et faisant partie de ceux offrant aujourd'hui les meilleures performances en la matière. Ces architectures encodeur-décodeur obtiennent souvent de très bonnes performances avec l'aide d'un bon corpus de données supervisées. Certaines des architectures les plus célèbres ont également été décrites, elles se basent essentiellement sur l'utilisation d'une augmentation de données massives pour jouir des meilleurs résultats possibles. Cependant, les stratégies d'augmentation de données reposent sur un ensemble initial qui se doit d'être un minima performant. Disposer de ces données n'est pas toujours simple en pratique, surtout que l'étiquetage est une tâche fastidieuse qui est généralement à la charge de l'utilisateur. Enfin, certaines augmentations de données peuvent présenter des défauts et être contre-productives.

C'est de ce constat que nous avons développé deux axes majeurs dans nos recherches. Le premier se base sur l'optimisation de l'augmentation de données et notamment le fait de pouvoir écarter automatiquement celles étant contre-productives. Le second, plus spécifique à la segmentation d'images de cellules, consiste à apprendre avec un nombre minimal de données afin de pouvoir s'affranchir de le nécessité de fournir de nombreuses données étiquetées. Ces deux axes de recherches seront longuement développés dans les deux prochains chapitres.

# Chapitre 4

# Apprentissage en ligne d'augmentation de données

Les deux chapitres précédents ont dressé un panorama de l'existant en matière d'algorithmes de segmentation d'images. A l'heure où ce manuscrit est rédigé, les meilleures performances sont obtenues par les méthodes d'apprentissage automatique par réseaux de neurones profonds, nécessitant une vaste base d'images annotées. Comme soulevé précédemment, l'enjeu des travaux de thèse présentés dans ce manuscrit est d'obtenir des résultats équivalents avec moins de données et surtout moins d'annotations.

Pour arriver à ce but, une première piste étudiée dans cette thèse est celle de l'augmentation de données. Derrière ce terme, se cache un ensemble de techniques qui permet d'accroître artificiellement la taille du jeu de données qu'un réseau de neurones va exploiter durant sa phase d'entraînement. Il s'agit donc d'une piste plutôt évidente pour espérer apprendre avec succès à partir d'une quantité limitée de données. L'augmentation de données est aujourd'hui un processus bien établi en apprentissage profond et a démontré son efficacité dans de très nombreuses situations. Elle a été en particulier rendue populaire par le réseau AlexNet [69] qui en a fait usage pour accroître la taille de son jeu de données d'un facteur 2048 mais l'idée était aussi déjà présente dans LeNet [74] où Lecun et al. ont utilisé de l'augmentation sur le célèbre jeu de données MNIST.
Les mécanismes d'augmentation de données sont nombreux et variés [111]. Qui plus est, chaque mécanisme peut généralement être déployé de plusieurs manières différentes à travers le réglage de paramètres. Face à cette multitude de choix possibles, le programmeur souhaitant entraîner un modèle peut légitimement se poser les questions suivantes :

- Est ce que tous ces mécanismes fonctionnent sur mes données ?
- Pour ceux qui fonctionnent, ont ils tous la même efficacité ?
- Comment répartir au mieux mon budget en capacité de calcul sur ces différents mécanismes ?

La première contribution de ces travaux de thèse s'inscrit dans une série de travaux récents qui cherchent à automatiser l'allocation d'un budget computationnel sur les différents mécanismes d'augmentation. La solution retenue et exposée ci-après dans ce chapitre, repose essentiellement sur l'apprentissage en ligne. Dans ce paradigme d'apprentissage, l'apprenant est confronté à différentes options et converge vers une allocation en fonction des pertes observées imputables à chaque option.

Ce chapitre commence par un rappel rapide du principe de l'augmentation de données et des mécanismes les plus courants d'augmentation. Il est également proposé un tour d'horizon d'un ensemble de travaux connexes qui vise à automatiser le choix des augmentations. Ensuite, la nouvelle méthode proposée est présentée puis mise à l'épreuve sur des tâches de segmentation. Diverses perspectives d'approfondissement de cette méthode ainsi que ses limites sont discutées avant de conclure le chapitre.

## 4.1 Augmentation de données : principes et généralités

Commençons par formaliser quelque peu le problème à traiter contextualisé à l'apprentissage de réseaux de neurones pour la segmentation. Pour entraîner le réseau de neurones, on dispose d'un jeu de données organisé de la manière suivante :

— $\mathcal{S}_{\text{tr}} = \left\{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right\}_{i=1}^{n_{\text{s-tr}}}$ est le jeu d'apprentissage qui contient $n_{\text{s-tr}}$ paires annotées d'entrées (images à segmenter) et de sortie (masque de segmentation),
— $\mathcal{S}_{\text{val}}$ est le jeu de validation qui contient $n_{\text{s-val}}$ paires annotées,
— $\mathcal{S}_{\text{test}}$ est le jeu de test qui contient $n_{\text{s-test}}$ paires annotées.

De manière classique, $\mathcal{S}_{\text{tr}}$ est utilisé pour la descente de gradient du réseau de neurones tandis que l'ensemble $\mathcal{S}_{\text{val}}$ est utilisé pour le réglage d'hyperparamètres et l'ensemble $\mathcal{S}_{\text{test}}$ est exclusivement réservé à l'évaluation des performances. Notons que dans ce chapitre, le cas où des données supplémentaires non annotées sont disponibles n'est pas traité et fera l'objet de développements dans le chapitre suivant.

Pour attaquer ce problème, on dispose de $K$ mécanismes d'augmentation de données, i.e. un ensemble de fonctions $(a_k)_{k=1}^{K}$ qui prennent une paire $(\mathbf{x}, \mathbf{y})$ en entrée et fournissent une nouvelle paire $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ exploitable pour l'apprentissage. De telles fonctions sont particulièrement faciles à concevoir et utiles quand les données sont structurées comme c'est le cas pour les images. Elles exploitent souvent des transformations géométriques ou un aspect du rendu visuel d'une image. Voici un certain nombre d'exemples de fonctions d'augmentation possibles :

— **Rotations** : Typiquement pour les images de microscopie cellulaire, il n'y a pas spécialement d'orientation prédéfinie au contenu visualisé. On peut donc facilement effectuer une rotation de 90, 180, ou 270 degrés à la fois sur $\mathbf{x}$ et $\mathbf{y}$ pour obtenir trois nouvelles paires à partir d'une seule de départ. La plupart des images étant carrées, on n'a pas de problème de redimensionnement à résoudre. Plus généralement, on peut choisir des angles de rotation différents mais qui imposent la synthèse d'une partie des images ainsi augmentées. Par exemple, pour une rotation de 45 degrés, il faudra déterminer la valeur des pixels de l'image augmentée qui, par rotation inverse, tombent en effet en dehors de la zone du champ visuel couverte par la grille pixellaire de l'image d'origine. Par défaut, on peut utiliser du *padding* pour combler les valeurs manquantes, sinon il faudra faire appel à un modèle génératif préalablement entraîné (voir remarques sur les techniques d'augmentation avancées évoquées un peu plus loin).

— **Symétries** : De manière similaire aux rotations, différentes symétries peuvent produire des paires d'images à peu de frais et sans problème de redimensionnement. Il s'agit des symétries axiales (horizontale et verticale). Notons que la symétrie centrale est équivalente à la rotation de 180 degrés.

— **Ajout de bruit** : Le processus classique d'acquisition d'images s'accompagne d'un bruit qui modifie aléatoirement à la hausse ou à la baisse la valeur d'un pixel. La nature du bruit et son intensité dépendent de la nature du capteur et du temps d'exposition. On peut créer un nouvel exemple d'apprentissage en rajoutant un bruit à $\mathbf{x}$ et en laissant $\mathbf{y}$ intacte. Cette opération peut sembler contre-productive au départ puisque l'ajout du bruit va rendre la détection du *pattern* potentiellement plus difficile. En pratique, cela contraint le réseau à mieux généraliser. Les bruits généralement utilisés en augmentation de données sont de nature gaussienne ou uniforme. L'insertion s'effectue généralement pixel par pixel, i.e. avec indépendance du bruit d'un pixel à l'autre. On a

$$\tilde{x}_i = x_i + b \text{ avec } b \sim \mathcal{N}(0; \sigma_b) \text{ ou } U_{[-e;+e]},$$

pour tout $i$ et avec un écart type $\sigma_b$ ou une demi-largeur d'intervalle $e$ à choisir.

— **Changements d'illumination ou colorimétriques** : Une scène visuelle peut être plus ou moins claire ou sombre. Il est possible de modifier l'illumination d'une scène avec des transformations simples effectuées, le plus souvent, pixel par pixel. La plus classique est la correction gamma, on a

$$\tilde{x}_i = x_i^{\gamma},$$

pour tout $i$ et avec $\gamma$ un réel positif à choisir. Une valeur supérieure à 1 pour $\gamma$ assombrira l'image tandis qu'une valeur inférieure à 1 la rendra plus claire. Plusieurs généralisations de cette transformation existent (avec seuillage et/ou ajout d'une constante additive par exemple). D'autres techniques comme l'égalisation d'histogramme [87] sont un peu dans la même veine et permettent d'accentuer certains contrastes. Enfin, si l'image est en couleur, on peut changer d'espace couleur ou de couleur dominante, de teinte [134]. Ces changements peuvent aussi être appliqués de manière anisotrope, c'est à dire en accentuant progressivement la transformation dans une zone arbitraire de l'image. Comme pour le bruit, cette augmentation n'intervient pas sur le masque $\mathbf{y}$.

— **Déformations élastiques** : À l'image de [114], il est possible de déformer une image à l'aide de petits mouvements appliqués à chaque pixel. Les mouvements peuvent être générés aléatoirement en tirant selon une loi uniforme sur un intervalle centré. Plus l'intervalle est large, plus la déformation est importante. On effectue ce tirage deux fois pour obtenir un déplacement 2D. Le pixel $(i, j)$ se trouve donc déplacé en $(i + u, j + v)$ avec $u, v \sim U_{[-e;+e]}$. Ce pixel ne se trouve donc plus nécessairement sur la grille pixellaire car ses indices ne sont plus

des entiers. L'image augmentée $\tilde{\mathbf{x}}$ est ensuite obtenue par interpolation bilinéaire des pixels déplacés les plus proches d'une position (entière) sur la grille. La même transformation doit être appliquée à $\mathbf{y}$.

— **Effacement partiel** : Il a été constaté par certains auteurs [140], qu'effacer les pixels dans une région de l'image (souvent rectangulaire) crée une nouvelle image $\tilde{\mathbf{x}}$ qu'un réseau peut apprendre à associer à $\mathbf{y}$. On peut y voir une parenté certaine avec la catégorie d'ajouts de bruit et également la technique du dropout [117]. La zone à effacer est souvent tirée aléatoirement mais en imposant des contraintes de surface minimale et maximale.

— **Interpolation** : Cette augmentation consiste à prendre une paire d'exemples disponibles et d'en calculer la moyenne. C'est une technique utilisable dans une très large classe de modèles mais qui fonctionne également avec des réseaux de neurones traitant des images [54].

La liste précédente n'est pas exhaustive et contient seulement les transformations les plus usuelles. Notons que certaines revêtent une forme d'aléa, ce qui signifie que $\tilde{\mathbf{x}}$ est parfois une variable aléatoire. Bien sûr, elles peuvent aussi être combinées, ce qui rend la quantité de choix possibles encore plus grande. Pour clarifier ce point, on souligne que l'entier $K$ dans notre situation désigne l'ensemble des mécanismes possibles (combinaisons incluses).

Plus récemment, des techniques d'augmentation plus élaborées ont été mises en place. Il est en effet possible d'utiliser l'apprentissage automatique pour créer un nouveau mécanisme d'augmentation potentiellement plus puissant que ceux cités plus haut. Par exemple, dans [142, 36], les auteurs entraînent un modèle génératif sur les images d'entrée dans un premier temps, puis utilisent ce modèle pour générer des nouveaux exemples. Ces deux articles sont appliqués sur des tâches de classification où la sortie n'a pas besoin d'être augmentée. Ceci est plus difficilement transposable en segmentation car il faudrait apprendre un modèle joint (entrée, sortie) ce qui revient déjà à résoudre la tâche elle-même. Dans le même ordre d'idée, il est également possible d'utiliser des techniques de type transfert de style [56] afin de générer de nouveaux exemples d'apprentissage. On peut, à partir d'un exemple souche, générer plusieurs exemples en balayant aléatoirement la gamme de style apprise.

Encore plus ambitieux, on peut tenter de laisser libre cours à l'apprentissage pour trouver tout seul le meilleur mécanisme [97] dans un esprit *learning to learn*. Dans de telles approches, plusieurs réseaux de neurones sont nécessaires : un pour résoudre la tâche principale et un autre pour résoudre la tâche d'augmentation. Dans [97], un terme de *loss* spécifique à chaque réseau est mis au point puis, la fonction objectif globale est optimisée.

Indépendamment de la qualité de chacun des mécanismes ou combinaisons de mécanismes, la pléthore de possibilités rend impossible leur usage exhaustif pour entraîner un modèle. Qui plus est, comme certains sont aléatoires, on pourrait tirer indéfiniment de nouveaux exemples d'apprentissage. En lien avec de nombreux efforts et réflexions de la communauté de l'apprentissage artificiel, il convient d'éviter un usage excessif des moyens de calcul. Typiquement, le programmeur devra choisir une certaine quantité fixe d'exemples augmentés à ajouter à $\mathcal{S}_{\text{tr}}$ et $\mathcal{S}_{\text{val}}$ par *epoch*. À qui confier

alors ce budget ? Cette question a inévitablement suscité un intérêt et plusieurs travaux récents y apportent des réponses. Ils sont présentés dans la section suivante. Notre contribution est également une réponse possible à ce questionnement.

## 4.2 Travaux connexes en automatisation de l'augmentation de données

Dans cette section, divers travaux récents qui tentent d'automatiser le choix de mécanismes d'augmentation sont présentés succinctement. Un travail central dans ce domaine est l'approche AutoAugment de Cubuk et al. [25]. Dans cette approche, l'espace de recherche ne se limite pas aux seules fonctions $(a_k)_{k=1}^{K}$ mais comprend aussi les paramètres avec lesquels utiliser ces fonctions ainsi que les probabilités $p_k$ d'être activées ou non. Un levier important d'AutoAugment est que l'espace de recherche est discrétisé. Un total de $K = 16$ fonctions sont utilisées, associées individuellement à un seul paramètre uniformément discrétisé sur 10 valeurs et les probabilités d'activation $p_k$ discrétisées sur 11 valeurs (équiréparties sur $[0; 1]$). Pour l'enchaînement de deux opérations, que les auteurs appellent sous-politique, il y a donc $(16 \times 10 \times 11)^2$ combinaisons possibles. Le but est de trouver un enchaînement de 5 telles sous-politiques. Un tel enchaînement constitue une politique et il y en donc un total de $(16 \times 10 \times 11)\,10$. Il est intéressant de remarquer que cet espace de recherche comprend la possibilité de ne pas augmenter. En effet, si l'algorithme apprend des probabilités $p_k$ nulles, il décide de renoncer à l'augmentation correspondante.
Une fois l'espace de recherche défini, il faut un algorithme pour apprendre quelle politique choisir. AutoAugment a recours à l'apprentissage par renforcement pour ce faire. L'idée de base est qu'un réseau récurrent (appelé contrôleur) va proposer $B$ politiques. Ensuite, le réseau cible réellement souhaité (qui résout la tâche de segmentation par exemple) est entraîné avec les augmentations correspondantes aux $5 \times B$ sous-politiques [1] proposées. Les performances de validation de ce modèle entraîné servent de signal de récompense au contrôleur pour se mettre à jour. Concernant le contrôleur, il s'agit d'un réseau récurrent de type LSTM [47] dont la prédiction est fournie par 30 couches softmax. En effet, pour une politique, on choisit 5 sous-politiques, pour chaque sous-politique, on choisit deux augmentations et chaque augmentation se compose de trois choix (type d'opération, valeur de paramètre et probabilité $p_k$). Ceci donne donc $5 \times 2 \times 3 = 30$ décisions à rendre. L'optimisation proximale de politique [108] est utilisée pour entraîner le contrôleur.
Une limite de l'approche AutoAugment est qu'elle ne permet que d'apprendre des combinaisons de deux opérations, c'est pourquoi Zheng et al. ont proposé DeepAA (deep AutoAugment) [139] qui peut apprendre un enchaînement plus vaste d'augmentations. Pour arriver à cet objectif, les auteurs

1. Pour chaque image d'un batch, une sous-politique est tirée au hasard selon une loi uniforme.

choisissent d'optimiser les probabilités de sélection des opérations d'augmentation de sorte à ce que les gradients des images augmentées soient alignés avec ceux d'images non augmentées de l'ensemble de validation. Pour chaque couche d'augmentation, une distribution multinomiale est donc à identifier. Les auteurs proposent d'apprendre les coefficients de ces lois par descente de gradient sur le cosinus de l'angle formé par les deux vecteurs de gradient. Une solution analytique est fournie pour la première couche. Pour les autres, il faut recourir à une approximation de Monte-Carlo du gradient fourni par l'enchaînement d'opérations d'augmentation.
Toutefois, la critique la plus sévère concernant AutoAugment relève essentiellement du temps de calcul qu'elle implique. Un défaut important de cette méthode est qu'elle fait appel à plusieurs entraînements quasi-complets du réseau cible pour obtenir des signaux de récompense ce qui rend son usage très énergivore et nécessite un temps de calcul considérable. Par exemple, pour AutoAugment, à chaque itération, le réseau cible est entraîné sur 150 *epochs*. Certains auteurs [77] sont parvenus à réduire la procédure à un seul apprentissage pour trouver la politique d'augmentation, suivi d'un second apprentissage pour apprendre le réseau cible sur les données augmentées sélectionnées. L'approche en question s'appuie sur l'optimisation bayésienne, un choix logique puisque le problème du choix des mécanismes d'augmentation émarge aussi dans la problématique plus générale du réglage automatique des hyperparamètres.
Une autre idée est proposée par Faster AutoAugment (Faster AA) [43] sur la base de relaxation différentiable (par exemple en remplaçant la variable aléatoire binaire d'activation d'une augmentation par une généralisation continue [58] ou encore à l'aide de gradients dit *straight through* [4]). Grâce à la différentiabilité, l'apprentissage des paramètres de politique d'augmentation peuvent être appris en rétro-propageant depuis une fonction de perte qui mélange l'objectif du réseau cible ainsi qu'un terme d'adéquation entre la distribution des données augmentées et celle du problème initial. Le terme d'adéquation est une distance de Wasserstein et cet objectif est particulièrement judicieux car toute augmentation de données devrait contribuer à fournir de nouveaux exemples sous la loi du problème de départ et non pas à étudier des lois génératives plus générales ce qui relève de l'apprentissage *Out Of Distribution* (OOD).
La même forme de relaxation différentiable est également exploitée par Differentiable Automatic Data Augmentation (DADA) [76]. DADA optimise en revanche une fonction objective à deux niveaux (un sur l'erreur de *train* du réseau cible et un autre sur l'erreur de validation vis à vis des paramètres de la politique d'augmentation). L'optimisation alterne entre les deux niveaux. L'approche [79] suit la même forme d'optimisation alternée, mais passe par une reformulation du problème qui permet d'éviter d'utiliser des astuces de re-paramétrisation. En pratique, l'optimisation passe par un critère d'alignement de gradients à l'image de [139]. Il en va de même pour [104] avec un déploiement fructueux sur une tâche de classification de signaux EEG et des politiques d'augmentation spécifiquement adaptées à certaines classes. Cette idée est en effet clairement intéressante : si on prend l'exemple de la classification de chiffres manuscrits, la rotation de 180 degré est un choix catastrophique pour la classe "6" qu'elle transforme en "9" tandis qu'elle est

bénéfique pour la classe "8", globalement invariante sous cette transformation.
Dans la contribution qui sera présentée dans la section suivante, nous avons souhaité aller encore plus loin dans la démarche, à savoir apprendre la politique d'augmentation en même temps que le réseau cible, et ce avec un surcoût négligeable (sans gradients supplémentaires à obtenir).

## 4.3 Apprentissage en ligne d'augmentation de données

Cette section présente la première contribution de ces travaux de thèse. Il s'agit d'un mécanisme d'allocation automatique d'un budget computationnel sur un ensemble de $K$ augmentations possibles. Pour simplifier la présentation, on généralise la définition des fonctions $(a_k)_{k=1}^K$ à une notion d'action à choisir et une action peut être une seule opération d'augmentation ou un enchaînement de plusieurs augmentations.
Comme évoqué plus haut, l'intérêt principal de notre méthode par rapport à l'état de l'art est son surcoût calculatoire très modeste. En effet, comme il sera présenté dans les paragraphes qui suivent, notre méthode s'appuie sur les gradients qui sont déjà calculés pour l'apprentissage du réseau cible. Elle ne nécessite donc pas de rétro-propagation supplémentaire. Cette méthode, appelée SODA pour Self-Organizing Data Augmentation, puise son mécanisme d'apprentissage des paramètres d'allocation dans l'apprentissage en ligne qui converge donc au fur et à mesure de la descente effectuée par le réseau cible.
La section commence par une formalisation du problème ainsi que par quelques rappels sur l'apprentissage en ligne et des concepts connexes. Ensuite, le principe de la méthode ainsi que sa mise en œuvre sont exposés.

### 4.3.1 Définition du problème et apprentissage en ligne

On s'intéresse à l'apprentissage d'un réseau de neurones noté $f_{\boldsymbol{\theta}}$ où $\boldsymbol{\theta}$ désigne l'ensemble des paramètres du réseau que l'apprentissage va chercher à déterminer. Pour ce faire, la phase d'apprentissage va consister à régler $\boldsymbol{\theta}$ de sorte à minimiser pour chaque paire $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}_{\text{tr}}$ une perte notée $\mathcal{L}\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right), \mathbf{y}\right)$. Dans notre cas, la fonction $\mathcal{L}$ sera typiquement l'erreur quadratique :

$$\mathcal{L}\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right), \mathbf{y}\right) = \left\| f_{\boldsymbol{\theta}}\left(\mathbf{x}\right) - \mathbf{y} \right\|_2^2 .$$

La fonction objectif est l'erreur quadratique moyenne :

$$L_{\text{mse}}\left(\boldsymbol{\theta}\right) = \frac{1}{n_{\text{s-tr}}} \sum_{i=1}^{n_{\text{s-tr}}} \mathcal{L}\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right), \mathbf{y}\right). \tag{4.1}$$

Pour améliorer l'apprentissage, le programmeur se donne un budget supplémentaire de $n_a$ données à générer depuis les processus d'augmentations $(a_k)_{k=1}^K$. La question qui se pose alors est : quelle fraction de ce budget doit on allouer pour chaque processus ? La solution à ce problème sera fournie sous la forme d'un vecteur $\boldsymbol{\pi}$ de dimension $K$ et qui contient des probabilités $\pi_k$ d'allocation pour les différents processus. Étant donné que la proportion de

données allouées à chaque processus d'augmentation n'est pas toujours la même au fil des *epochs* d'entraînement du réseau, dans la suite de ce chapitre, on indexe également le vecteur de probabilité par $t$ qui désigne le numéro d'*epoch*. En effet, il semble parfaitement possible que certains processus soient très utiles en début d'apprentissage pour rapidement se diriger vers des zones intéressantes de l'espace de paramètres tandis qu'en fin d'apprentissage, plus de diversité dans le choix des processus peut être profitable. Ainsi, on peut légitimement penser que l'approche proposée, de même que celles s'appuyant sur une optimisation alternée [76, 79, 104], offrent une réponse plus fine (sous la forme d'une séquence $(\boldsymbol{\pi}_t)_{t=1}^{T}$) au problème par rapport aux approches qui fournissent une allocation fixe. L'entier $T$ désigne le nombre d'*epochs* choisi (supposé fixe pour plus de simplicité).
Pour un vecteur de probabilités donné, la quantité de données requise auprès du $k$-ième processus d'augmentation est $n_{k,t} = [n_a \times \pi_{k,t}]$ où $[.]$ désigne la fonction arrondi à l'entier le plus proche. On suppose également que $n_a \gg K$ de sorte à ce que l'arrondi ait un impact négligeable sur notre approche. De même, on contraindra l'approche à laisser au moins une donnée à chaque processus : $n_{k,t} \geq 1, \forall k$, et ce tout respectant également $\sum_k n_{k,t} = n_a$. Cette dernière contrainte n'est pas absolument nécessaire mais permet à un processus d'augmentation infructueux en début d'apprentissage de revenir dans la compétition ultérieurement.
La colonne vertébrale de l'approche présentée dans la prochaine sous-section est un algorithme d'apprentissage en ligne appelé HEDGE [35]. Dans le paradigme de l'apprentissage en ligne, l'environnement fournit un vecteur de pertes $\boldsymbol{\ell}_t$ de dimension $K$ qui procure un retour sur le choix de l'ensemble des actions disponibles pour l'apprenant [2]. Dans notre cas, l'environnement est incarné par le processus d'apprentissage du réseau cible $f_{\boldsymbol{\theta}}$ et les pertes correspondront typiquement à une détérioration de l'erreur quadratique moyenne sous une forme qui sera précisée dans les paragraphes qui suivent. La perte au sens du problème d'apprentissage de $\boldsymbol{\pi}_t$ sur une itération est donnée par le produit scalaire $\boldsymbol{\pi}_t^{\intercal} \cdot \boldsymbol{\ell}_t$. Sur la base des pertes constatées, le vecteur d'allocation de ressources $\boldsymbol{\pi}_t$ est mis à jour.

**Algorithme 5 :** HEDGE

Initialisation des poids $w_{k,1} = 1, \forall k$ ;
$t = 1$;
**tant que** $t \leq T$ **faire**
  L'apprenant choisit $\boldsymbol{\pi}_t$ selon $\pi_{k,t} = \frac{w_{k,t}}{\sum_{k=1}^{K} w_{k,t}}, \forall k$ ;
  L'environnement révèle $\boldsymbol{\ell}_t$ ;
  L'apprenant met à jour ses poids selon
  $w_{k,t+1} = w_{k,t} \exp\left(-\eta \ell_{k,t}\right), \forall k$;
  $t = t + 1$;
**fin**

L'algorithme 5 donne le canevas général de HEDGE. Il s'appuie sur des variables intermédiaires $w_{k,t}$, appelées poids, pour mettre à jour le vecteur de

2. Il s'agit d'un formalisme proche de celui dans bandits. Dans le problème des bandits, à chaque itération, l'apprenant choisit une action et obtient donc un retour sur cette unique action.

probabilités d'allocation. On retiendra deux principes essentiels :

— plus $w_{k,t}$ est élevé, plus le budget alloué au $k$ème processus d'augmentation sera important,

— plus $\ell_{k,t}$ est élevée, plus violente sera la diminution de $w_{k,t+1}$ par rapport à $w_{k,t}$.

Le paramètre $\eta$ qui apparaît lors de la dernière étape de HEDGE est le *learning rate* correspondant à l'apprentissage de $\boldsymbol{\pi}_t$.

La littérature en apprentissage en ligne est vaste et d'autres algorithmes pourraient être substitués à HEDGE dans SODA. Concernant HEDGE, on souligne un résultat théorique important disponible dans cette littérature [20] : pour des pertes bornées, il atteint le regret[3] minimal vis à vis de la perte cumulée $\sum_{t=1}^{T} \boldsymbol{\pi}_t^\intercal \cdot \boldsymbol{\ell}_t$. Autrement dit, il appartient à la classe des algorithmes qui découvrent le plus rapidement la meilleure allocation.

S'il est intéressant de mentionner ce résultat, ce dernier n'est pas directement applicable à notre situation puisque l'allocation optimale peut elle-même évoluer avec $t$. On se rapproche des conditions du théorème en choisissant $\eta$ suffisamment petit pour que, d'une itération à l'autre, l'allocation optimale évolue à une vitesse significativement plus faible que celle de la convergence de HEDGE. Par ailleurs, il convient également de préciser que dans le paradigme classique de l'apprentissage en ligne, l'environnement est adversoriel, c'est à dire qu'il choisit les pertes $\ell_{k,t}$ de sorte à maximiser le regret. Dans notre situation, l'environnement a un comportement passif. Étant donné les spécificités de notre cas d'usage, quelques adaptations seront proposées dans SODA pour tenir compte de l'aspect dynamique de l'allocation optimale.

### 4.3.2 SODA : Self-Organizing Data Augmentation

Cette section présente SODA, un algorithme d'allocation de budget de calcul sur différents mécanismes d'augmentation qui peut être couplé à l'optimisation par réseaux de neurones pour un coût supplémentaire très faible. HEDGE est l'épine dorsale de cet algorithme. En s'appuyant sur HEDGE, SODA fournit un algorithme simple à implémenter et à contextualiser pour l'allocation de données augmentées. Il est déployé sur un seul cycle d'entraînement du réseau cible, et contrairement aux approches concurrentes, ne nécessite pas de calculs de gradients supplémentaires.

#### 4.3.2.1 Construction d'une fonction de perte pour l'apprentissage en ligne

Pour que l'apprentissage en ligne fonctionne, il est capital d'avoir accès à des signaux pertinents de perte (ou de récompense selon le point de vue) pour chaque action $a_k$ sélectionnable. Dans notre contexte, derrière chaque action se cache un mécanisme d'augmentation et pour quantifier à quel point ce mécanisme est sans intérêt (voire néfaste) pour entraîner $f_{\boldsymbol{\theta}}$, le critère suivant est retenu :

3. Le regret se définit comme l'écart entre la perte cumulée subie conséquemment aux choix d'actions fait par l'apprenant par rapport à la perte cumulée d'un oracle qui choisirait dès $t = 0$ l'allocation optimale.

$$c_{k,t} = \frac{1}{n_{k,t}} \sum_{i=1}^{n_{k,t}} \frac{1}{n_{\text{s-tr}}} \sum_{(\mathbf{x},y)\in\mathcal{S}_{\text{tr}}} \mathcal{L}\left(f_{\theta_t+\Delta\theta_{t,i}}(\mathbf{x}), y\right) - \mathcal{L}\left(f_{\theta_t}(\mathbf{x}), y\right), \quad (4.2)$$

$$= \frac{1}{n_{k,t}} \sum_{i=1}^{n_{k,t}} L_{\text{mse}}\left(\boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_{t,i}\right) - L_{\text{mse}}\left(\boldsymbol{\theta}_t\right) \quad (4.3)$$

avec $\Delta\boldsymbol{\theta}_{t,i}$ la mise à jour du vecteur de paramètres obtenu à partir de la $i$ème donnée générée par le $k$ème processus d'augmentation lors de l'*epoch* courante $t$. Cette mise à jour est donnée par l'optimiseur de type descente de gradient choisi pour entraîner $f_{\boldsymbol{\theta}}$ à cette étape de l'apprentissage, c'est pourquoi le vecteur de paramètres est lui aussi indexé par $t$. Notons qu'en toute rigueur l'indice $k$ devrait aussi apparaître mais il sera omis pour alléger quelque peu les notations. Ce critère s'interprète de la manière suivante : il s'agit de l'écart moyen en erreur quadratique moyenne obtenu en utilisant des données issues du $k$ème mécanisme d'augmentation lors de l'*epoch* $t$.
Si le $k$ème mécanisme d'augmentation aide l'apprentissage du réseau cible, les données qu'il fournit contribuent à faire baisser l'erreur quadratique moyenne et $c_{k,t}$ est négatif. Dans le cas contraire, l'erreur quadratique moyenne augmente ou stagne et $c_{k,t}$ est positif ou nul. En ce sens, $c_{k,t}$ encode bien une notion de perte relative à l'action $a_k$.
Même si le critère $c_{k,t}$ est facilement interprétable et constitue l'objectif primordial associé à l'augmentation de données, il présente quelques défauts. Tout d'abord, en vertu de la seconde étape de HEDGE, à l'issue d'une *epoch*, il faudrait calculer l'ensemble des gradients correspondant à toutes les données augmentées. Lors de la prochaine *epoch*, ces données ne seront pas toutes dans le premier *batch*. Soit il faut abandonner les gradients qui n'y sont pas, soit il faut passer à un optimiseur hybride *full batch* pour une itération suivi de *mini-batch* pour les autres itérations à l'intérieur d'une *epoch*. Deuxièmement, l'évaluation des valeurs d'erreur quadratique moyenne (MSE) pour l'ensemble des $\boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_{t,i}$ nécessite $n_a \times n_{\text{s-tr}}$ passes *forward* dans le réseau cible, ce qui n'est pas négligeable.
Pour alléger cette charge de calcul, il est possible de passer par un développement de Taylor à l'ordre 1 de l'erreur quadratique :

$$L_{\text{mse}}\left(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{t,i}\right) \approx L_{\text{mse}}\left(\boldsymbol{\theta}\right) + \nabla L_{\text{mse}}\left(\boldsymbol{\theta}\right)^{\intercal} \cdot \Delta\boldsymbol{\theta}_{t,i} \quad (4.4)$$

En substituant dans (4.3), il vient

$$c_{k,t} \approx \frac{1}{n_{k,t}} \sum_{i=1}^{n_{k,t}} \nabla L_{\text{mse}}\left(\boldsymbol{\theta}\right)^{\intercal} \cdot \Delta\boldsymbol{\theta}_{t,i}, \quad (4.5)$$

$$\approx \nabla L_{\text{mse}}\left(\boldsymbol{\theta}\right)^{\intercal} \cdot \left(\frac{1}{n_{k,t}} \sum_{i=1}^{n_{k,t}} \Delta\boldsymbol{\theta}_{t,i}\right). \quad (4.6)$$

En supposant par exemple que l'optimiseur retenu pour le réseau cible est

l'algorithme du gradient stochastique, on a $\Delta\boldsymbol{\theta}_{t,i} = -\alpha \mathbf{g}_{t,i}^{(k)}$ où $\mathbf{g}_{t,i}^{(k)}$ est le gradient calculé sur la $i$-ième donnée générée par le $k$-ième processus d'augmentation et $\alpha$ est le *learning rate* du gradient stochastique. Notons $\mathbf{g}_t^{(k)} = \frac{1}{n_{k,t}} \sum_i \mathbf{g}_{t,i}^{(k)}$ le gradient moyen fourni par la $k$ème augmentation. Pour rendre les notations plus homogènes, on note aussi $\mathbf{g}_t^{(0)} = \nabla L_{\text{mse}}(\boldsymbol{\theta})$ le gradient moyen issu des données sans augmentation. Finalement, en première approximation, on constate que notre critère revient à vérifier l'alignement du gradient moyen issu des données augmentées par le $k$-ième processus et du gradient moyen fourni par les données originelles ce qui rejoint les travaux [79, 104, 139].
Pour éviter de multiples évaluations de l'erreur quadratique moyenne, il est donc proposé la définition suivante pour le signal de perte de l'apprentissage en ligne :

$$\ell_{k,t} = \frac{1}{2}\left(1 - \frac{\mathbf{g}_t^{(0)\,\mathsf{T}} \cdot \mathbf{g}_t^{(k)}}{\left\|\mathbf{g}_t^{(0)}\right\|_2 \left\|\mathbf{g}_t^{(k)}\right\|_2}\right). \tag{4.7}$$

Cette expression ne fait pas apparaître $\alpha$ qui est indépendant de $k$. On préfère également étudier le cosinus de l'angle formé par les vecteurs de gradients moyens afin d'obtenir des pertes bornées : $\ell_{k,t} \in [0;1]$.
Si on applique strictement l'équation (4.7), il n'en reste pas moins qu'il faut obtenir les gradients pour l'ensemble des données (augmentées et non-augmentées). En pratique, on se contentera d'une version "stochastique", c'est à dire qu'on calculera les gradients moyens sur la base de ceux obtenus pendant l'*epoch* qui vient de s'achever. Si le *learning rate* $\alpha$ est suffisamment petit [4], on ne dévie pas trop de (4.7) mais surtout l'apprentissage en ligne devient extrêmement peu coûteux : il faut calculer et mémoriser $K+1$ vecteur de gradients moyens puis faire $K$ produits scalaires. Le coût de ces opérations est négligeable devant celui là même de l'entraînement du réseau cible. Il conviendra également de panacher la composition d'un *batch* de sorte à visiter régulièrement des données non-augmentées et des données issues de divers processus d'augmentation.

#### 4.3.2.2 Apprentissage en ligne avec facteur d'oubli

Comme l'allocation recherchée est susceptible d'évoluer avec $t$, on se propose d'utiliser une généralisation de HEDGE auquel un paramètre $\beta \in [0;1]$ de type facteur d'oubli est adjoint. Ce paramètre intervient dans la troisième étape de la boucle de HEDGE, à savoir la mise à jour des poids de sorte que celle-ci devient

$$w_{k,t+1} = w_{k,t}^{\beta} \exp\left(-\eta \ell_{k,t}\right), \forall k. \tag{4.8}$$

Quand $\beta = 1$, on retombe bel et bien sur HEDGE tandis que quand $\beta = 0$, on obtient un processus sans mémoire. Cette simple modification permet donc à notre approche d'oublier certaines pertes $\boldsymbol{\ell}_t$ observées dans un passé lointain, et ce, afin de s'adapter plus rapidement à un changement possible de

4. Ceci est déjà requis pour que l'approximation de Taylor soit bonne.

l'efficacité des processus d'augmentation au cours l'apprentissage du réseau cible.
Par ailleurs, si on déroule la formule entre la première *epoch* et la courante, on obtient

$$w_{k,t} = \exp\left(-\eta \sum_{t'=1}^{t} \beta^{t-t'} \ell_{k,t'}\right). \tag{4.9}$$

Cette autre expression laisse apparaître la perte cumulée pondérée dans laquelle les termes les plus récents ont un poids plus important que les termes éloignés dans le passé. Cela revient à appliquer HEDGE en vue de minimiser la perte cumulée pondérée $\sum_{t=1}^{T} \beta^{T-t} \boldsymbol{\pi}_t^\intercal \cdot \boldsymbol{\ell}_t$.

#### 4.3.2.3 Algorithme retenu

Nous avons à présent introduit toutes les étapes majeures qui vont constituer SODA. Avant de résumer la procédure algorithme correspondante, on donne quelques précisions supplémentaires.
Premièrement, l'équation (4.7) a été bâtie en supposant que le réseau cible suit une descente de gradient orchestrée par l'algorithme du gradient stochastique. Il ne faut pas y voir une limitation, mais plus une hypothèse facilitant la présentation de SODA. Si un optimiseur différent est utilisé (RMSprop ou ADAM par exemple) alors il faut simplement redéfinir $\mathbf{g}_t^{(k)}$ comme la moyenne des $\Delta\boldsymbol{\theta}_{t,i}$ correspondant au $k$-ième processus d'augmentation.
En outre, on peut légitimement s'attendre à ce que les gradients calculés lors des premières *epochs* soient plus volatiles. Pour stabiliser l'algorithme, on va donc substituer dans (4.7) les gradients moyens par une somme glissante à poids exponentiel de ces derniers. Pour cela, on utilise des vecteurs de momentum comme suit :

$$\mathbf{m}_t^{(k)} = \rho \mathbf{m}_{t-1}^{(k)} + (1-\rho)\, \mathbf{g}_t^{(k)}, \tag{4.10}$$

$$\tilde{\mathbf{g}}_t^{(k)} = \frac{\mathbf{m}_t^{(k)}}{1-\rho^t}, \tag{4.11}$$

avec $\mathbf{m}_t^{(k)}$ le vecteur de momentum pour le $k$-ième processus d'augmentation initialisé par $\mathbf{m}_0^{(k)} = \mathbf{0}$. La constante $\rho$ est un hyperparamètre de SODA qui régule la portée du moyennage glissant. Finalement, le signal de perte devient $\ell_{k,t} = \frac{1}{2}\left(1 - \frac{\tilde{\mathbf{g}}_t^{(0)\intercal} \cdot \tilde{\mathbf{g}}_t^{(k)}}{\left\|\tilde{\mathbf{g}}_t^{(0)}\right\|_2 \left\|\tilde{\mathbf{g}}_t^{(k)}\right\|_2}\right)$. Il est également possible de démarrer la mise à jour des poids de HEDGE avec un retard de $\tau$ *epochs* afin de constituer une phase de *burn-in* et d'obtenir des valeurs de gradients moyens plus fiables. Toutefois, cette possibilité ne sera pas actionnée dans les expériences

numériques présentées dans la prochaine section.

**Algorithme 6 :** SODA (Self-Organizing Data Augmentation)

Entrées : $n_a$, $\eta$, $\rho$, $\beta$, $\tau = 0$, réseau $f_{\boldsymbol{\theta}}$, $\boldsymbol{\theta}_0$, données non-augmentées $\mathcal{S}_{\text{tr}}$, fonctions $(a_k)_{k=1}^K$;
Initialisation des poids $w_{k,1} = 1, \forall k$ ;
Initialisation des vecteurs de momentum $\mathbf{m}_0^{(k)} = \mathbf{0}, \forall k$ ;
$t = 1$;
**tant que** $t \leq T$ **faire**
  $\pi_{k,t} = \frac{w_{k,t}}{\sum_{k=1}^K w_{k,t}}, \forall k$ ;
  $n_{k,t} = \max\{1; [n_a \times \pi_{k,t}]\}$;
  **tant que** $\sum_k n_{k,t} > n_a$ **faire**
    $k_* = \arg\max_k \; n_{k,t}$;
    $n_{k_*,t} = n_{k_*,t} - 1$;
  **fin**
  Obtenir les données augmentées $\text{DA}_k = \left\{ \left( \mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) \right\}_{i=1}^{n_{k,t}}, \forall k$ grâce aux fonctions $(a_k)_{k=1}^K$ et l'ensemble $\mathcal{S}_{\text{tr}}$.;
  Effectuer une *epoch* de descente de gradient pour $f_{\boldsymbol{\theta}}$ sur les données $\mathcal{S}_{\text{tr}} \cup \bigcup_{k=1}^K \text{DA}_k$ ;
  Obtenir $\boldsymbol{\theta}_{t+1}$ ainsi que les gradients moyens $\left( \mathbf{g}_t^{(k)} \right)_{k=0}^K$;
  $\mathbf{m}_t^{(k)} = \rho \mathbf{m}_{t-1}^{(k)} + (1-\rho)\, \mathbf{g}_t^{(k)}$;
  $\tilde{\mathbf{g}}_t^{(k)} = \frac{\mathbf{m}_t^{(k)}}{1-\rho^t}$;
  $\ell_{k,t} = \frac{1}{2} \left( 1 - \frac{\tilde{\mathbf{g}}_t^{(0)\,\intercal} \cdot \tilde{\mathbf{g}}_t^{(k)}}{\left\| \tilde{\mathbf{g}}_t^{(0)} \right\|_2 \left\| \tilde{\mathbf{g}}_t^{(k)} \right\|_2} \right)$ ;
  **si** $t > \tau$ **alors**
    $w_{k,t+1} = w_{k,t}^{\beta} \exp\left(-\eta \ell_{k,t}\right), \forall k$;
  **fin**
  $t = t + 1$
**fin**
Retourner $\boldsymbol{\theta}_T$ et $(\boldsymbol{\pi}_t)_{t=1}^T$.

La procédure correspondant à SODA est résumée par l'algorithme 6. Dans cette procédure on suppose que chaque processus d'augmentation est capable de fournir le budget $n_{k,t}$ alloué. Dans le cas contraire, l'excédent de budget serait reversé sur d'autres processus selon des modalités à définir. Dans nos expériences, ce cas ne se présente jamais. On peut d'ailleurs choisir $n_a$ de sorte à se prémunir de ce cas de figure. Une possibilité serait toutefois de reverser l'excédent au processus avec la probabilité d'allocation maximale et qui ne soit pas celui pour lequel la limite est atteinte. Notons de plus, qu'un bon nombre de processus sont aléatoires. De manière générale, les augmentations fournies d'une *epoch* à l'autre ne sont pas forcément les même paires $\left( \mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right)$.
Pour ce qui est de l'obtention des moyennes des gradients, les bibliothèques classiques de programmation de réseaux de neurones profonds proposent

des outils qui mémorisent les gradients avant application. Dans ces travaux, on utilise Keras avec Tensorflow en arrière plan. L'outil en question est une instance de `tf.GradientTape`.

## 4.4 Validation expérimentale en segmentation

Dans cette section, plusieurs expériences numériques sont menées afin de tester la validité de SODA en terme de capacité à distinguer les mauvais processus d'augmentation des bons. On commence par décrire les jeux de données utilisés et le protocole expérimental avant de tester SODA dans différentes situations.

### 4.4.1 Jeux de données et prétraitements

Étant donné l'objectif de la thèse, SODA est éprouvé dans ce chapitre sur des données d'imagerie de micro-biologie cellulaire associées à des tâches de segmentation bien qu'il soit applicable dans tout contexte d'apprentissage supervisé de réseaux de neurones. Les jeux de données utilisés sont

— DRIVE [118] qui contient des images d'ophtalmologie où on cherche à segmenter des vaisseaux sanguins de rétine. L'analyse de ces vaisseaux (branchements, longueurs, tortuosité, etc.) permet d'établir des diagnostics pour différentes pathologies cardiovasculaires ou ophtalmologiques. Le jeu se compose de 40 images couleur (avec annotations) de résolution $768 \times 584$ pixels.

— CAPSULE [107] qui est un jeu de données composé d'images de micro-capsules artificielles traversant un canal dont la section est de $100 \times 100\mu m^2$. Elles ont été acquises par l'équipe du Pr. Anne-Virginie Salsac de l'UTC de Compiègne à l'aide de caméras haute-fréquence. Les micro-capsules ont des propriétés mécaniques comparables à celles des globules rouges et leurs déformations lors de l'écoulement dans le canal permettent de mieux comprendre le flux sanguin. Ces travaux de thèse ont été l'occasion de collaborer avec cette équipe, notamment en procurant une vérité terrain de précision grâce à un utilitaire de contourage s'appuyant sur des courbes de Bézier et des fonctionnalités d'Inkscape. Le jeu de données contient 353 images annotées en niveau de gris et de résolution $768 \times 768$ pixels. Il est disponible sur un dépôt Github public.

— DIC-C2DH-HeLa qui contient des images de cellules de type HeLa, c'est à dire appartenant à une lignée de culture de cellules cancéreuses initialement obtenue via un prélèvement de type biopsie sur métastases. Le jeu se compose de 65 images en niveaux de gris (avec annotations) de résolution $628 \times 690$ pixels.

— PhC-C2DH-U373 qui contient des images de microscopie de type contraste de phase montrant des cellules de type glioblastome astrocytome U373 (cancer du cerveau). Le jeu se compose de 48 images en niveaux de gris (avec annotations) de résolution $992 \times 832$ pixels.

Les deux derniers jeux de données sont mis à disposition par l'ISBI [83] dans le cadre de challenges de reconnaissance dans des applications biomédicales.
Indépendamment du jeu de données, les images sont toutes pré-traitées de sorte à être passées à la même architecture U-net. Elles seront en niveaux de gris et normalisées (moyenne nulle et variance unitaire). Enfin, pour chaque dataset, nous utiliserons $n_{\text{s-tr}} = 20$ images pour l'entraînement de U-net ainsi que 20 données supplémentaires pour former l'ensemble de validation couplé à un Early Stopping. Toutes les images restantes dans un dataset sont employées pour constituer l'ensemble de test $\mathcal{S}_{\text{test}}$.

### 4.4.2 Cadre expérimental

Le but des expériences menées et présentées dans cette section est d'entraîner un réseau de neurones de type U-net. Pour cette série d'expériences, l'architecture retenue est constituée de 8 couches (une moitié pour la partie encodeur et l'autre moitié pour la partie décodeur). Le nombre de filtres par couche va en croissant pour l'encodeur et en décroissant pour le décodeur de sorte que deux couches symétriquement opposées dans l'architecture (et reliées par une *skip connection*) possèdent le même nombre de filtres. Tous les filtres sont de taille $3 \times 3$ avec une option de *zero-padding* et utilisent ReLU comme fonction d'activation. La première couche possède 10 filtres. Ensuite, le nombre de filtres est multiplié par deux pour chaque nouvelle couche jusqu'à la dernière couche de l'encodeur. Les couches de *maxpool* utilisent un voisinage de taille 2x2, sans intervention de strides.
Côté environnement d'entraînement, l'optimiseur retenu est RMSProp avec un *learning rate* $\alpha = 1e-4$. Les autres paramètres sont pris à leurs valeurs par défaut (implémentation en Keras avec Tensorflow en arrière-plan). Une régularisation en norme euclidienne est utilisée pour tous les paramètres à apprendre sauf ceux de la dernière couche. La constante devant le terme de perte correspondant est fixée à $1e-4$.

Concernant les processus d'augmentation de données, on s'accorde sur un budget $n_a = 60$ données augmentées et on laisse à SODA $K = 3$ choix possibles :

— ajout de bruit : on tire $\sigma$ uniformément au hasard dans $\{0.01, 0.02, \ldots, 0.05\}$ et on multiplie l'image originelle par un bruit dont les pixels sont tirés d'une loi normale $\mathcal{N}(0, \sigma^2)$,
— rotation : on tire $a$ uniformément au hasard dans $\{1, 2, \ldots, 8\}$ et on effectue une rotation d'un angle $a\frac{\pi}{4}$ sur l'image originelle,
— "junk" : on remplace l'image d'entrée $\mathbf{x}$ par une image aléatoire où chaque pixel est tiré selon une loi uniforme dans l'intervalle $[0, 1]$.

On s'attend à ce que les deux premiers mécanismes permettent d'entraîner U-net malgré un jeu de données de taille modeste tandis que le dernier mécanisme devrait perturber l'apprentissage ou à minima gâcher du temps de calcul. C'est pourquoi, SODA sera comparé à une allocation "cible" dont le vecteur de probabilité correspond à

$$\boldsymbol{\pi}_t = \begin{bmatrix} 0,5 \\ 0,5 \\ 0 \end{bmatrix}.$$

L'autre allocation de base à laquelle SODA sera comparé est l'allocation uniforme :

$$\boldsymbol{\pi}_t = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}.$$

Enfin, le critère d'évaluation retenu est l'indice de Jaccard. Il s'agit d'un choix classique en segmentation d'images. Pour un exemple $(\mathbf{x}, \mathbf{y}) \in \mathcal{S}_{\text{test}}$, cet indice se définit par

$$\text{JI}(\%) = 100 \times \frac{\|\mathbf{y} \odot \hat{\mathbf{y}}\|_1}{\|\mathbf{y} + \hat{\mathbf{y}}\|_1 - \|\mathbf{y} \odot \hat{\mathbf{y}}\|_1}, \tag{4.12}$$

où le masque prédit $\hat{\mathbf{y}}$ est obtenu en comparant $f_{\boldsymbol{\theta}}(\mathbf{x})$ au seuil de 0.5 et $\odot$ est le produit élément par élément entre le masque prédit et la vérité terrain. Dans ces expériences, on exécute l'apprentissage 10 fois et on présente les indices de Jaccard moyen sur l'ensemble de ces exécutions.

### 4.4.3 Résultats

On donne ici plusieurs résultats pour SODA et deux autres stratégies de référence pour les quatre jeux de données cités plus haut. Ces résultats sont obtenus avec les choix de paramétrisation donnés en Table 4.1. Les paramètres $\rho$ et $\beta$ ne changent pas d'un jeu de données à l'autre ce qui montre une sensibilité faible de SODA à l'égard de ces paramètres. A l'inverse, le *learning rate* $\eta$ doit être plus finement réglé pour que SODA soit efficace.

| Dataset | $\eta$ | $\rho$ | $\beta$ |
|---|---|---|---|
| DRIVE | 6 | 0.99 | 0.5 |
| Capsule | 4 | 0.99 | 0.5 |
| ISBI : DIC-C2DH-HeLa | 3 | 0.99 | 0.5 |
| ISBI : PhC-C2DH-U373 | 7 | 0.99 | 0.5 |

TABLE 4.1 – Hyparamètres de SODA pour chaque jeu de données.

On rappelle aussi que comme évoqué précédemment, l'hyperparamètre $\tau$ est laissé à sa valeur par défaut ($\tau = 0$) dans toutes ces expériences.
On constate sur les graphes présentés en Figure 4.1, que SODA fonctionne mieux que l'allocation uniforme sur l'ensemble des jeux de données. SODA parvient aussi à tangenter ou égaler la stratégie cible. Pour autant, SODA ne converge par nécessairement vers une allocation semblable à la stratégie cible mais parvient essentiellement à rejeter l'augmentation "junk". Cela peut-être vérifié en Figure 4.2 qui montre les allocations déterminées par SODA au cours de l'apprentissage. L'élimination de l'augmentation inutile

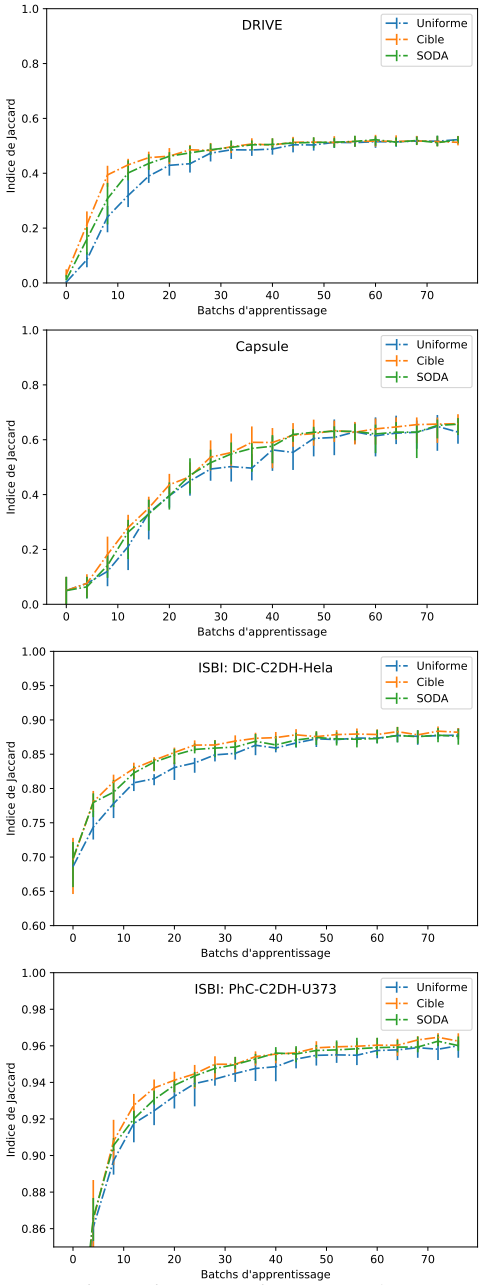


FIGURE 4.1 – Indices de Jaccard moyens (sur 10 exécutions) pour les 4 jeux de données et les trois stratégies d'allocation au cours de l'apprentissage, divisé en batchs de taille 80.

est presque totale pour deux jeux de données. Pour Capsule, l'élimination est plus progressive tandis que pour PhC-C2DH-U373, un résidu de quelques 10% du budget ne parvient pas à être éliminé mais reste minoritaire en comparaison du budget accordé aux autres mécanismes. Dans nos expériences, et contrairement à nos attentes initiales, nous avons pu constater la robustesse du processus d'apprentissage par rétro-propagation face à des données sans intérêt comme celles produites par le mécanisme "junk". Il semble que les gradients produits par de telles données ajoutent un bruit dans la trajectoire effectuée par $\boldsymbol{\theta}_t$ mais que ce bruit soit d'une amplitude faible devant les

gradients informatifs et n'empêche donc pas la convergence in fine.

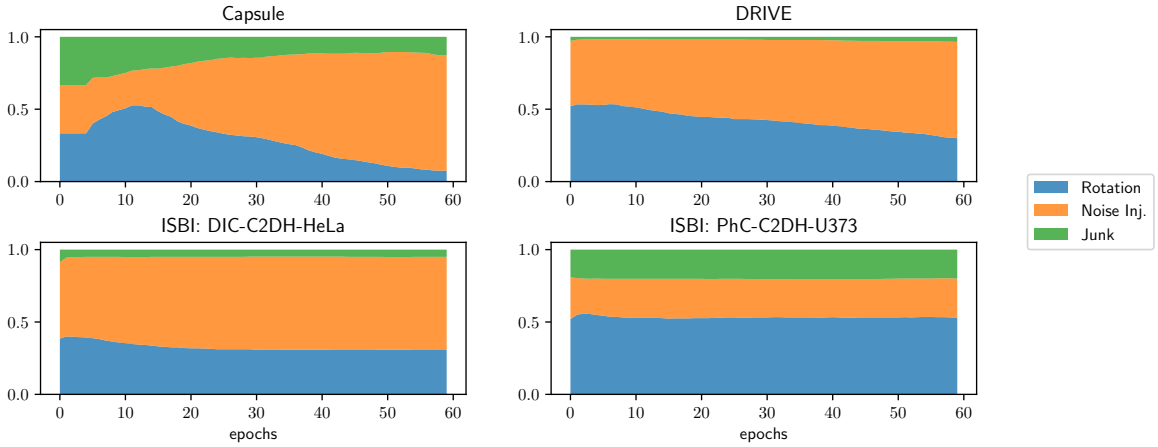


FIGURE 4.2 – Séquences d'allocations $\pi_t$ déterminées par SODA pour les 4 jeux de données.

Pour rebondir sur la dernière remarque, il est aussi important de remarquer qu'au bout d'un certain nombre d'itérations, un niveau similaire de performance est atteint par toute les stratégies. Ceci tend à montrer que SODA trouve son utilité plutôt dans l'accélération de la convergence de l'apprentissage plutôt qu'en termes d'amélioration pure des performances.

## 4.5 Discussion et évolutions possibles

L'approche SODA présente plusieurs qualités qui ont été mises en lumière par les expériences de la section précédente. En particulier, SODA, pour un surcoût très modique, permet d'éliminer un processus d'augmentation contre-productif et d'accélérer la convergence. Toutefois, pour atteindre cet objectif quelques choix simplificateurs ont été nécessaires et il n'en demeure pas moins que SODA possède aussi plusieurs limites, qui feront l'objet de discussions dans les paragraphes suivants. Différentes perspectives de travail permettant de pallier ces limitations seront également proposées.

Une première faiblesse de SODA est que l'alignement entre gradients qui sert à alimenter HEDGE en signaux de perte/récompense est calculé à partir de données de l'ensemble d'entraînement $\mathcal{S}_{\text{tr}}$. En ce sens, SODA peut surapprendre une allocation. Une solution possible pour éviter ce désagrément serait de calculer les signaux (4.7) sur des gradients issus de données de validation. Le prix à payer est alors que ces gradients ne peuvent être exploités pour la descente effectuée vis à vis de $\theta$ ce qui accroît le surcoût calculatoire liée à la méthode proposée. Toutefois, si l'ensemble de validation $\mathcal{S}_{\text{val}}$ n'est pas trop grand (ce qui est souvent le cas), ce surcoût reste toute de même limité.

Il convient également de revenir sur la difficile question de la convergence. Comme évoqué précédemment, pour se rapprocher des conditions de convergence de HEDGE, il faudrait faire plusieurs mises à jour de $\pi_t$ avant d'en

effectuer une sur $\boldsymbol{\theta}_t$ ce qui n'est pas la solution retenue dans la forme actuelle de SODA [5] et ce, afin de garder le surcoût calculatoire très faible. Il serait néanmoins possible de passer sur ce mode de fonctionnement en mettant à jour $\boldsymbol{\pi}_t$ non pas à la fin d'une *epoch* mais après le traitement de chaque *mini-batch*. Cela pose alors un autre problème : il n'est plus raisonnable de supposer pouvoir attribuer au moins une donnée à chaque processus d'augmentation à l'intérieur d'un *mini-batch*. Pour palier cet obstacle, on peut penser à des évolutions de SODA où l'apprentissage en ligne serait remplacé par des bandits. Dans le cadre d'un algorithme de bandits, un seul mécanisme serait sélectionné pour participer au prochain *mini-batch* et nous n'aurions un retour que sur ce seul mécanisme. Pour en revenir à la question de la convergence, notons que cette solution nous aiderait à converger pendant la durée d'une *epoch*. En revanche, d'une *epoch* à l'autre, le problème de bandits change, puisque $f_{\boldsymbol{\theta}}$ change aussi. On se retrouve un peu alors sur une forme d'optimisation alternée similaire aux approches [76, 79].
Enfin, on ne peut éluder le fait que, dans sa forme actuelle, SODA n'est sans doute pas adapté à un passage à de grandes valeurs de $K$, or, l'éventail de mécanismes d'augmentation possibles est extrêmement large, surtout si il comprend la combinaison de plusieurs dispositifs de base. Pour ces cas d'usage, il conviendrait de passer à un mode de fonctionnement similaire aux approches qui sont dans la lignée d'AutoAugment [25], à savoir avec une optimisation à faire pour chaque maillon du mécanisme d'augmentation à plusieurs étages.

## 4.6 Conclusion

Dans ce chapitre, un nouvel algorithme d'allocation d'un budget en données augmentées est introduit. Cet algorithme, appelé SODA pour Self-Organizing Data Augmentation, s'appuie sur l'apprentissage en ligne pour déterminer à la volée (au cours de la descente de gradient du réseau de neurones compagnon) les probabilités d'allocation du budget sur chaque mécanisme d'augmentation disponible. En comparaison aux approches concurrentes de l'état de l'art, SODA se distingue par un surcoût calculatoire très modeste puisqu'il exploite les gradients déjà calculés pour effectuer l'apprentissage du réseau.
SODA a montré son efficacité en terme d'élimination d'un processus d'augmentation parasite. Il a aussi été constaté empiriquement que SODA permet globalement d'accélérer la convergence de la descente de gradient. Par ailleurs, plusieurs évolutions ont été aussi présentées en vue de rendre SODA plus flexible et plus performant.
Dans la perspective plus globale de ces travaux de thèse, on rappelle que l'objectif primordial est d'apprendre à segmenter avec peu de données. Les expériences menées dans ce chapitre montrent que des taux d'indice de Jaccard satisfaisants peuvent être atteints sur certains jeux de données par le seul truchement de l'augmentation de données (sous réserve qu'elle soit bien choisie) et en partant de seulement 20 exemples d'apprentissage annotés. En

5. Notons qu'actuellement, les signaux de perte $\ell_{k,t}$ sont obtenus à partir de moyennes de gradients et sont donc sans doute moins bruités.

ce sens, ce travail est une première forme de réponse à l'objectif affiché de la thèse. Toutefois, il n'est pas une réponse complètement générale à notre problématique puisque si le nombre de données annotées est plus faible, la marge de manœuvre des mécanismes d'augmentation devient très réduite. En outre, sur certains jeux de données, on constate que les performances sont encore très largement perfectibles. Pour aller plus loin dans la réponse à ces différents besoins, le prochain chapitre présentera une seconde contribution faisant appel à des principes d'auto-apprentissage et qui pourra être couplée à l'augmentation de données pour apprendre à partir d'un nombre très limité d'exemples annotés.

# Chapitre 5

# Apprentissage auto-supervisé

## 5.1 Introduction

L'état de l'art en segmentation repose principalement sur des architectures de réseaux de neurones profonds, apprenant sur un vaste corpus d'images, idéalement ayant subit une augmentation de données massive. Le chapitre précédent a mentionné une stratégie permettant d'optimiser et de superviser au mieux l'augmentation de données. En ce sens, cela répond en partie aux problématiques évoquées dans le cadre de ces travaux de thèse. Cependant, un problème persiste quand le nombre d'images initial est trop infime pour permettre à l'augmentation de données d'améliorer suffisamment les performances du réseau. Contrairement au chapitre précédent, où la méthode proposée est générique et ne fait pas d'hypothèses sur le contenu des images, celle développée dans ce chapitre est spécifique aux images de microscopie cellulaire ou similaires.

La première réflexion toute légitime pour comprendre au mieux ce problème consiste à visualiser les résultats d'un réseau U-net quand il apprend sur un nombre très faible de données. Le protocole est trivial, il consiste à réduire itérativement le nombre de données présentées au réseau jusqu'à observer un décrochage au niveau de la métrique d'évaluation calculée sur un ensemble de validation. Afin de donner un ordre de grandeur, les résultats sur le jeu de données Capsule sont médiocres quand le nombre de données annotées passe en dessous de 8. L'objectif de ce chapitre est d'apprendre sur cette base de données étiquetées, puis de la pousser dans ses retranchements en réduisant progressivement au minimum la part de données annotées.

La figure 5.1 est une première réponse à ce questionnement. On cherche dans cette expérience à apprendre au réseau à trouver le contour d'un rectangle situé dans une image fortement bruitée. Les données sont générées artificiellement et on suppose qu'effectuer la segmentation du rectangle en ayant connaissance de son contour fermé n'est pas un problème. La figure 5.2 de son côté illustre le problème avec une tâche de segmentation réutilisant le jeu de donnes Capsule (présenté en 4.4.1).
On peut remarquer que dans ces deux cas de figure, le réseau est très proche de résoudre la tâche demandée. Il ne reste d'une part, qu'à relier deux points pour finir le contour, et d'autre part, qu'à remplir l'intérieur de la cellule. Ces exemples sont très nombreux et ne se limitent pas à ces ensembles de données.

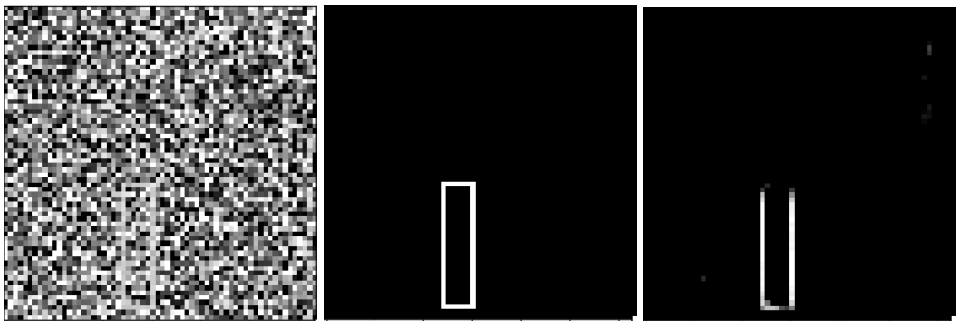

FIGURE 5.1 – Jeu d'images contenant des rectangles. L'image initiale, l'étiquetage et la prédiction du réseau sont répertoriés de gauche à droite respectivement.

FIGURE 5.2 – Triplet image-étiquetage-prédiction sur le jeu données des Capsules.

Il semble alors légitime de se poser la question suivante : comment est-il possible de corriger ce défaut ?
Une réponse à cette question sera longuement détaillée dans la suite de ce chapitre, elle repose sur l'utilisation d'un second réseau de neurones qui aura pour but d'apprendre à "fermer" des contours ou bien à "remplir des formes". Il est important de constater que la tâche restante est relativement basique, par conséquent, il est tout à fait possible d'entraîner un réseau capable de la résoudre. Malgré tout, certaines contraintes persistent : on souhaite être capable de le faire avec un faible nombre de données étiquetées. La piste explorée dans nos travaux consiste à rendre ce réseau correcteur générique, en apprenant à partir d'un *dataset* artificiel en amont, et d'être ensuite utilisé directement sans apprentissage supplémentaire, et ce sur des jeux de données différents correspondant à des images de cellules. Entraîner un réseau générique peut être considéré comme une mission ardue, mais dans notre cas la tâche visée pour ce dernier est d'une complexité raisonnable.

Cependant, l'utilisation d'un réseau correcteur tiers est insuffisante, en effet si le problème consiste uniquement à remplir l'intérieur d'une cellule, alors un algorithme automatique de remplissage suffirait pour parvenir à nos fins. La vraie difficulté du problème ne consiste alors pas à remplir, mais à également effectuer une tâche plus générale de complétion (une cellule pouvant tout de même avoir des formes relativement différentes et le réseau étudiant pouvant produire une sortie trop mauvaise pour être corrigée). La figure 5.3

montre un exemple dans lequel une grande partie de la cellule doit être complétée.

FIGURE 5.3 – Triplet image-étiquetage-prédiction sur le dataset des Capsules. Exemple où la prédiction omet une grande partie de la cellule.

Cet exemple est bien plus difficile à corriger car compléter la forme de la cellule consiste en partie à résoudre la tâche de segmentation. L'utilisation d'un simple réseau correcteur en plus du réseau initial n'est alors pas suffisant. De plus, obtenir de telles performances en ne tenant uniquement compte du faible nombre de données étiquetées est bien trop ambitieux. C'est pourquoi la stratégie employée consiste à nous aider d'un vaste corpus d'images non annotées pour aider notre réseau à mieux segmenter.
La stratégie décrite dans ce chapitre ne consistera pas à uniquement concaténer deux réseaux, elle devra également prendre en compte une chaîne de traitements adaptée aux images non supervisées. L'un des objectifs majeurs de notre stratégie est de perturber positivement les solutions proposées par notre réseau initial de manière à améliorer naturellement ses performances, mais surtout à éviter les optimums locaux fortement présents quand le nombre de données étiquetées est très faible. L'utilisation des données non annotées aide sur ce point, mais il faut que notre stratégie prenne également en considération des "attracteurs"[1] qui peuvent perturber l'apprentissage des poids du réseau et le faire converger vers une solution de qualité non-satisfaisante. Il faut en quelque sorte être capable de guider le réseau afin qu'il puisse parcourir dans son processus d'optimisation des solutions plus performantes et éviter de tomber dans la facilité. Dans l'idéal, ces solutions doivent remplir davantage la forme de l'objet à segmenter. Incorporer ce type de procédé n'est pas une tâche facile, nous utilisons dans notre cas un troisième réseau qui, une fois réuni avec les deux précédents, correspond à un autoencodeur. Ce dernier va alors comparer l'image initiale avec une image reconstruite par le troisième réseau dont l'objectif sera justement de reproduire une image initiale à partir d'un masque de segmentation. De ce fait, il sera possible d'utiliser un vaste corpus d'images non étiquetées au sein de notre algorithme, et l'utilisation du réseau professeur va forcer l'étudiant à

1. On entend par attracteur un minimum local dans l'espace de paramètres du réseau qui correspond à une fonction triviale et généralise très mal. Par exemple, la fonction $f_\theta(\mathbf{x}) = \mathbf{0}$ qui renvoie systématiquement une image noire.

tendre pour une solution de meilleure qualité. Nos travaux montrent qu'utiliser cet autoencodeur permet de contraindre efficacement les poids du réseau étudiant en le forçant à remplir et compléter les formes de cellules, sans quoi elles n'apparaîtraient pas entièrement dans l'image reconstruite.

Pour conclure sur ces prémisses de notre stratégie, il est à noter que la tâche de segmentation présentée en figure 5.3 est parfaitement résolue grâce au protocole proposé, et ce pour un nombre de données identiques et généralement très faible (3 données étiquetées dans le cas le plus poussé). Le chapitre va s'articuler autour de l'explication détaillée de cette stratégie. Dans un premier temps, un panorama exhaustif de la littérature sur les sujets connexes sera dressé, avant de passer à l'explication détaillée du protocole. Les résultats ainsi que les discussions qu'ils engendrent permettront de conclure le chapitre.

## 5.2 Panorama des sujets connexes

Notre stratégie s'inscrit naturellement dans l'état de l'art associé à la segmentation. Les différentes stratégies proposées dans le chapitre 2 sont donc parfaitement de mise mais présentent le défaut de pouvoir créer un effet de sur-segmentation, qu'on nomme superpixel [102]. Plus récemment, les réseaux de neurones profonds dont le fonctionnement est longuement détaillé dans le chapitre 3 ont fait leurs apparitions. Les architectures comme U-net [105], Segnet [2] ou tout autre FCN [81] brillent par leurs efficacités dans la résolution des tâches de segmentation. Cependant ces performances sont également à mettre au compte d'une quantité massive de données annotées disponibles. Dans le cadre de nos travaux, on souhaite apprendre sur un nombre infime de données, et ne pas se limiter uniquement à l'utilisation de l'augmentation de données [112]. Il n'est ainsi pas pertinent de comparer nos travaux à ceux utilisant ce type de procédé pour accroître le nombre de données étiquetées. On parle de *Few Shot Learning* [71, 88] pour définir les algorithmes d'apprentissage profond qui apprennent en ne tenant compte que d'un nombre très faible de données. Notre stratégie s'inscrit alors parfaitement dans cette catégorie, dont nous allons détailler les principales avancées dans la section suivante.

### 5.2.1 Few Shot Learning

Vinyals et al. [130] proposent un réseau nommé *Matching network* ayant pour but de résoudre une tâche de classification multi-classes à partir d'un seul étiquetage par classe (ce qui se nomme le *One Shot Learning*). La stratégie repose sur un mécanisme d'attention afin d'effectuer la prédiction sur une donnée de test. Ce mécanisme consiste à comparer la nouvelle donnée avec chaque donnée de l'ensemble d'entraînement (on rappelle qu'il n'y a qu'un seul exemple par classe). La fonction d'attention se base sur une fonction softmax sur la distance cosinus. Concernant l'apprentissage, un système d'*epochs* nommé "épisode" est utilisé et chaque épisode est conçu pour imiter la tâche

de *few-shot* en sous-échantillonnant les classes ainsi que les points de données. Ce système permet de rendre la phase d'apprentissage plus fidèle à celle de test, et donc d'augmenter la généralisation. Ravi et Larochelle [100] proposent une variante de ce modèle, également basée sur le système d'épisodes. Si le réseau voit les données toujours dans un ordre précis (et défini par les épisodes), alors l'utilisation d'un LSTM [48] permet d'ajouter une mémoire au réseau qui apprendra naturellement à partir de l'agencement des *batchs*.

Toujours dans cette idée d'utiliser la notion de distance pour aider à la classification, Snell et al. [116] définissent les *Prototypal networks*. L'objectif consiste à utiliser un réseau de neurones afin d'être capable d'apprendre une fonction d'intégration $h(x)$ dont le but est de ramener dans un unique espace les données initiales. On souhaite que dans ce nouvel espace, les données de la même classe soient proches, et celles de classes différentes soient alors éloignées. L'affectation d'une donnée de test se fait par un calcul de distance entre chaque élément d'une même classe. Mengye Ren et al. [101] proposent une version semi-supervisée de cet algorithme. Les données non étiquetées sont directement ajoutées dans le *batch*. Le réseau va alors ramener ces nouvelles images non étiquetées dans son espace, par lequel une variante de l'algorithme des $K$-moyennes (voir section 2.4.2) sera utilisée pour l'attribution des classes respectives.

Utiliser de l'apprentissage non supervisé afin d'extraire des informations capables de guider la partie supervisée est un concept assez répandu. Pour ce faire, les autoencodeurs (section 3.6), ainsi que les réseaux siamois [15] ont un rôle non négligeable dans le domaine du *Few Shot Learning*. Gregory Koch [68] a étendu le principe du réseau siamois pour créer les réseaux de neurones siamois profonds. L'idée principale consiste à remplacer les deux réseaux concurrents de l'approche classique par des réseaux convolutifs. La stratégie utilise également de l'augmentation de données, plus spécifiquement des opérations de distorsion. Les autoencodeurs de débruitage (section 3.6.4) se basent également sur l'augmentation de données, couplée à de l'apprentissage non supervisé. Les représentations d'encodeurs bidirectionnels à base de transformateurs (BERT) [28] utilisent également de l'apprentissage non supervisé. Le réseau va apprendre dans un premier temps sur des données non étiquetées, avant d'ajouter la partie supervisée. La partie bidirectionnelle du réseau le rend normalement capable d'effectuer la tâche souhaitée dans les deux sens, ce qui agit comme une forte contrainte au sein de l'apprentissage.

### 5.2.2 Self-learning

La stratégie décrite dans ce chapitre s'articule autour d'un champ plus spécifique de l'apprentissage non supervisé nommé le *self-learning* [136]. Dans cette catégorie, une méthode nommée distillation de connaissances [46] présente de fortes similarités avec nos travaux. A partir de deux réseaux dits

'étudiant' et 'professeur', un mécanisme d'étiquetage automatique sera effectué. Le réseau professeur va apprendre dans un premier temps sur les données étiquetées, pour ensuite produire des annotations sur toutes les données non étiquetées. Le réseau étudiant va alors apprendre à partir d'un corpus d'images plus vaste, regroupant également les données fournies par le professeur. L'une des motivations principale de cette stratégie consiste à sensiblement réduire la taille du réseau étudiant, comparativement à son professeur tout en gardant des performances similaires.

Pham et al. [98] proposent une variante consistant à réapprendre après un certain temps le réseau professeur puis d'alterner l'apprentissage des deux réseaux. Le professeur doit donc être capable de modifier les pseudo-étiquetages qu'il propose afin d'améliorer la fonction de perte globale. Ce même procédé est également utilisé dans les *Stochastic dropout network* [70] dans lequel les deux réseaux font partie d'un même ensemble, et dont la fonction de perte minimisera également la différence entre les sorties des deux réseaux. Ces travaux montrent qu'utiliser des étiquetages provenant d'une partie non supervisée permet d'augmenter la robustesse du réseau face aux mauvais étiquetages. Tarvainen et al. [121] ont également suggéré de modifier les poids du professeur sur la base d'une moyenne mobile exponentielle directement utilisée sur les poids du réseau étudiant. À la fin de l'apprentissage, le réseau professeur est plus susceptible d'obtenir de meilleures performances.

### 5.2.3 Originalité de notre stratégie

La stratégie mise en place et détaillée dans ce chapitre se place naturellement dans le champ d'étude du *self-learning*. Avant d'en expliquer les moindres détails, il semble primordial de pouvoir s'interroger sur ce qui la rend originale par rapport à l'état de l'art actuel. On peut distinguer un ensemble de différents points qui, assemblés, rendent nos travaux uniques :

1. Notre stratégie permet de segmenter des images à partir d'un faible nombre de données étiquetées.
2. Elle repose sur du *self-learning* et nécessite un corpus d'images non étiquetées. De plus, la structure des cellules est exploitée afin de faire plus efficacement de la distillation de connaissances entre nos différents réseaux.
3. Elle n'utilise aucune information extérieure telle que du *transfer learning*.
4. Bien que la stratégie repose sur de nombreux réseaux, son utilisation finale pour la segmentation ne requiert que le réseau nommé étudiant, qui est composé d'un nombre restreint de paramètres.

Il est également important de noter que nos travaux ne se situent pas dans le cadre des approches entièrement auto-supervisées [131] qui ne nécessitent aucune donnée étiquetée, mais dont le niveau de performances reste à ce stade significativement plus bas.

## 5.3 Méthodologie

### 5.3.1 Notation

Avant d'expliquer les différentes parties de notre protocole d'auto-apprentissage, il convient de définir en amont une notation claire pour les jeux de données. On note $S_{\text{tr}}$ l'ensemble des données d'entraînement, composé de $n_{\text{s-tr}}$ paires de données $(x^i, y^i)$. Un second ensemble de ce type nommé $S_{\text{val}}$ s'occupera de la validation des réseaux, notamment via l'utilisation d'un *early stopping* au sein de la boucle d'apprentissage. L'ensemble de validation est généralement constitué d'une seule paire. Notre stratégie, basée sur du *self-learning* repose également sur une partie non supervisée. On note $U_{\text{tr}}$ l'ensemble de données non étiquetées d'entraînement et $U_{\text{val}}$ son homologue associé à la validation. Ces deux ensembles contiennent un nombre de données, qu'on notera $n_{\text{u-tr}}$ et qui sera de toute évidence bien plus important que $n_{\text{s-tr}}$. Le protocole expérimental proposé repose sur l'utilisation jointe de 3 réseaux. Le réseau étudiant, ainsi que le réseau professeur de notre approche *self-learning* seront notés $f_{\boldsymbol{\theta}}^{(\text{etu})}$ et $f_{\boldsymbol{\phi}}^{(\text{prof})}$ respectivement. Enfin, un dernier réseau nommé $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ fera son apparition et aura pour objectif de produire le résultat inverse du réseau étudiant. Par conséquent, il recevra en entrée un masque de segmentation et retournera une image proche de l'image initiale. Chaque réseau optimise un vecteur de paramètres différent, notés respectivement $\boldsymbol{\theta}$ pour l'étudiant, $\boldsymbol{\phi}$ pour le professeur et $\boldsymbol{\varphi}$ pour le réseau inverse. Ces réseaux sont tous des architectures U-net (section 3.5.2.2) dont la structure générale est rappelée en figure 5.4. L'application successive de ces trois réseaux permettra de former un dernier réseau de type autoencodeur qu'on notera $f^{(AE)}$.

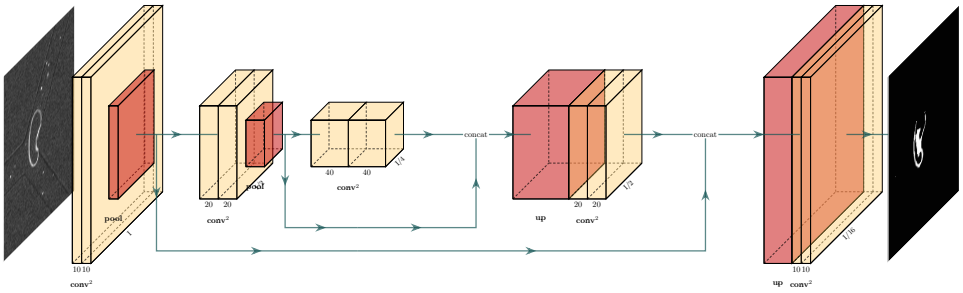


FIGURE 5.4 – Structure du réseau U-net, composé d'une partie encodeur et décodeur.

### 5.3.2 Vision globale du protocole

La stratégie de *self-learning* proposée peut se décomposer en quatre phases distinctes.

1. Entraîner le réseau $f_{\boldsymbol{\phi}}^{(\text{prof})}$ à partir de données artificielles.
2. Entraîner le réseau $f_{\boldsymbol{\theta}}^{(\text{etu})}$ en utilisant $S_{\text{tr}}$ et $S_{\text{val}}$

3. Entraîner le réseau $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ en utilisant également $S_{\text{tr}}$ et $S_{\text{val}}$
4. Améliorer les performances de $f^{(\text{etu})}$ à partir de tous les ensembles $S_{\text{tr}}$,$S_{\text{val}}$ , $U_{\text{tr}}$ et $U_{\text{val}}$, utilisés dans l'apprentissage du réseau $f^{(AE)}$.

### 5.3.3 Entraînement du réseau professeur

Le réseau professeur a pour objectif de corriger la prédiction proposée par l'étudiant. Plus précisément, l'étudiant apprenant sur un faible corpus de données va se tromper principalement sur des erreurs associées au contour, ou le remplissage de la forme (section 5.1). Pour ce faire, on souhaite posséder un réseau qui puisse compenser le réseau étudiant sur ces défauts, mais qui peut également être générique. On ne souhaite pas re-entraîner le réseau professeur sur chaque jeu de données, et ce pour plusieurs raisons :

— Le réseau professeur possède un nombre de poids bien plus important que les autres réseaux, son entraînement est également plus coûteux.
— L'un des objectifs de notre stratégie est de se substituer à l'augmentation de données. Même si la combinaison des deux n'est en rien incompatible, on souhaite éviter de baser notre approche sur une augmentation massive utilisée directement sur $S_{\text{tr}}$.

Bien que le concept de réseau générique soit lié à la réalisation d'une tâche souvent complexe, nous allons tout de même pouvoir jouer sur plusieurs facteurs dont le principal n'est autre que la difficulté de la tâche à résoudre. En effet, la tâche à réaliser est relativement simple puisqu'elle consiste à remplir des objets dont le contour est déjà identifié. De plus, la forme des cellules sur laquelle se base nos travaux est généralement simple et se rapproche d'une ellipse. Par conséquent, nous allons créer un jeu de données artificiel composé de couples qu'on notera $(\tilde{\mathbf{m}}, \mathbf{m})$ pour entraîner ce réseau. Les principes qui motivent ce jeu de données se basent essentiellement sur les autoencodeurs de débruitage (section 3.6.4).
L'étiquetage noté $\mathbf{m}$ correspond à une donnée d'ellipse générée aléatoirement (position centrale et grandeur des axes) dont l'intérieur est rempli. Elle a également subi une déformation élastique aléatoire. Plus précisément :

1. La position du centre de l'ellipse est tirée selon une loi uniforme sur la grille pixellaire.
2. Les longueurs des axes sont tirées selon une loi uniforme dont le support est $[0.03N; 0.6N]$ (on rappelle que $N$ désigne le nombre de pixels d'une image).
3. On applique une déformation élastique aléatoire à l'aide de la fonction `random_distortion` du paquet Python Augmentor avec les paramètres laissés à leurs valeurs par défaut.

L'image $\mathbf{m}$ correspond alors à une pseudo-ellipse correctement segmentée ou parfaite. L'image d'entrée notée $\tilde{\mathbf{m}}$ sera une version corrompue de la sortie $\mathbf{m}$. Les fonctions utilisées pour corrompre les données se concentrent principalement sur les défauts de U-net et consistent à éroder l'intérieur de l'ellipse, afin de créer des défauts de remplissage. Un ajout de bruit est également effectué pour améliorer la généralisation du réseau. Un triplet d'images représentant respectivement $\tilde{\mathbf{m}}$, $f_{\boldsymbol{\phi}}^{(\text{prof})}(\tilde{\mathbf{m}})$ et $\mathbf{m}$ est montré en figure 5.11.

(A) $\tilde{\mathbf{m}}$ (B) $f_{\boldsymbol{\phi}}^{(\text{ref})}(\tilde{\mathbf{m}})$ (C) $\mathbf{m}$

FIGURE 5.5 – Triplet d'images données au réseau professeur, ainsi qu'un exemple de sa prédiction.

Cette même figure montre également que la prédiction proposée par le réseau est "meilleure" que l'étiquetage donné. En effet le réseau a tendance à mieux lisser les contours de l'ellipse, alors que l'image initiale, obtenue par déformation élastique possède des bords biens plus rigides. Un tel niveau de performance est obtenu à l'aide d'un réseau possédant un nombre de poids très important, ainsi qu'avec l'utilisation du générateur de paire $(\tilde{\mathbf{m}}, \mathbf{m})$ permettant d'avoir accès à une grande masse de données synthétiques. Pour donner un ordre de grandeur, le réseau professeur est entraîné sur un corpus d'images différentes dont la taille est de l'ordre du million.
La fonction de perte à minimiser par le réseau professeur est une erreur quadratique moyenne sur l'ensemble des données synthétiques, qu'on notera $\mathcal{M}$. On souhaite résoudre

$$\arg\min_{\boldsymbol{\phi}} L_{\text{prof}}(\boldsymbol{\phi}; \mathcal{M}) = \sum_{(\tilde{\mathbf{m}},\mathbf{m})\in\mathcal{M}} \left\| \mathbf{m} - f_{\boldsymbol{\phi}}^{(\text{prof})}(\tilde{\mathbf{m}}) \right\|_2^2. \tag{5.1}$$

### 5.3.4 Apprentissage du réseau étudiant

Le seconde étape de notre protocole d'apprentissage consiste à apprendre le réseau étudiant sur les données étiquetées notées $S_{\text{tr}}$ et $S_{\text{val}}$ respectivement. A l'instar du réseau professeur, une MSE est également utilisée pour entraîner le réseau. On cherche ici à résoudre :

$$\arg\min_{\boldsymbol{\theta}} L_{\text{etu}}(\boldsymbol{\theta}; \mathcal{S}_{\text{tr}}) = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{S}_{\text{tr}}} \left\| \mathbf{y} - f_{\boldsymbol{\theta}}^{(\text{etu})}(\mathbf{x}) \right\|_2^2. \tag{5.2}$$

Il est à noter que le résultat de cet apprentissage est déterminant. L'objectif global de la stratégie mise en œuvre consiste à améliorer les performances de cet unique réseau. A l'évidence si cet entraînement réussit, le protocole proposé dans ce chapitre devient superflu, toutefois, quand $n_{\text{s-tr}}$ est très petit, ce dernier n'a quasiment aucune chance d'apprendre un bon modèle. C'est pourquoi on s'intéresse ici à une taille de l'ensemble $S_{\text{tr}}$ petite, en général inférieure à 10 données, à moduler en fonction du jeu de données.
Enfin, les scores de ce réseau doivent être soigneusement sauvegardés afin de pouvoir effectuer une comparaison des performances du réseau avant et

après l'utilisation de notre protocole auto-supervisé. De ce fait, la fin de l'apprentissage du réseau étudiant nous donne les performances de base qu'on souhaite surpasser ensuite.

### 5.3.5 Apprentissage du réseau inverse

Comme son nom l'indique, le but du réseau inverse est d'être capable d'obtenir une image proche de $\mathbf{x}$ à partir de son homologue étiquetée $\mathbf{y}$. Il peut sembler particulier d'avoir à utiliser ce type de réseau, mais en réalité cela permet d'être capable de définir un autoencodeur (section 3.6) au sein de notre protocole. En effet, en appliquant le réseau inverse directement sur la sortie du professeur, une image proche de $\mathbf{x}$ va alors être créée, et celle-ci pourra être comparée avec $\mathbf{x}$ lui-même dans une fonction de perte identique à ce qu'utilisent habituellement les autoencodeurs. On cherche donc à résoudre ici :

$$\arg\min_{\boldsymbol{\theta}} L_{\text{inv}}\left(\boldsymbol{\varphi}; \mathcal{S}_{\text{tr}}\right) = \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{S}_{\text{tr}}} \left\| \mathbf{x} - f_{\boldsymbol{\varphi}}^{(\text{inv})}\left(\mathbf{y}\right) \right\|_2^2. \tag{5.3}$$

L'apprentissage de ce réseau est une tâche bien plus difficile que l'apprentissage de l'étudiant. C'est lui qui généralement définit la limitation basse de la taille de $S_{\text{tr}}$. En effet, avec un nombre trop faible de données, le réseau devient incapable d'effectuer la reconstruction de l'image initiale. De plus, cet apprentissage ajoute des contraintes sur les jeux de données à exploiter. Il faut que le fond des différentes images puisse présenter une faible variabilité pour que le réseau soit capable de moyenner le fond des images. Ce postulat est généralement respecté car les séquences des jeux de données cellulaires, produites en laboratoire reposent sur un protocole strict qui vise à limiter au maximum les changements de luminosité, ou d'environnement. Nous discuterons plus en détails de ces limitations dans la partie discussion qui y sera en partie consacrée.

### 5.3.6 Combinaisons des réseaux et ajustement des poids du réseau étudiant

Une fois les trois premières étapes terminées, la suite consiste à combiner les différents réseaux afin de créer un autoencodeur. Les trois réseaux vont alors être concaténés dans l'ordre suivant : $f^{(\text{etu})}$, $f^{(\text{prof})}$ puis $f^{(\text{inv})}$. On parle de combinaison pour dire que la sortie d'un réseau sera directement mise en relation avec l'entrée du réseau suivant. À savoir que l'objectif de ce protocole est d'améliorer les performances de $f^{(\text{etu})}$, ainsi les poids $\boldsymbol{\phi}$ et $\boldsymbol{\varphi}$ des réseaux $f_{\boldsymbol{\phi}}^{(\text{prof})}$ et $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ vont rester fixes dans cette phase du protocole. De cette façon, toutes les modifications visant à minimiser la fonction de perte devront être effectuées dans $f_{\boldsymbol{\theta}}^{(\text{etu})}$. Cela bride la capacité de notre réseau et nous assure du bon fonctionnement de $f_{\boldsymbol{\phi}}^{(\text{prof})}$ et de $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ tout au long de la boucle d'apprentissage.

Concernant la fonction de perte, deux nouveaux termes vont être ajoutés à $L_{etu}$, et vont utiliser les ensembles $U_{\text{tr}}$ et $U_{\text{val}}$. Deux flux vont donc parcourir

notre architecture, et seront conjointement responsables de la fonction de perte globale. Les termes adjoints sont :

$$L_{\text{cons}}\left(\boldsymbol{\theta};\mathcal{U}_{\text{tr}}\right)=\sum_{\mathbf{x}\in\mathcal{U}_{\text{tr}}}\left\|f_{\boldsymbol{\phi}}^{(\text{prof})}\left(f_{\boldsymbol{\theta}}^{(\text{etu})}\left(\mathbf{x}\right)\right)-f_{\boldsymbol{\theta}}^{(\text{etu})}\left(\mathbf{x}\right)\right\|_2^2, \tag{5.4}$$

$$L_{\text{ae}}\left(\boldsymbol{\theta};\mathcal{U}_{\text{tr}}\right)=\sum_{\mathbf{x}\in\mathcal{U}_{\text{tr}}}\left\|f_{\boldsymbol{\varphi}}^{(\text{inv})}\left(f_{\boldsymbol{\phi}}^{(\text{prof})}\left(f_{\boldsymbol{\theta}}^{(\text{etu})}\left(\mathbf{x}\right)\right)\right)-\mathbf{x}\right\|_2^2. \tag{5.5}$$

Les objectifs de ces deux nouvelles fonctions de pertes sont déterminants dans le bon fonctionnement de notre stratégie. $L_{\text{cons}}$ minimise la différence entre la sortie du réseau étudiant, et la sortie du professeur. L'effet souhaité est de forcer l'étudiant à rapprocher sa sortie de celle du professeur. On considère le postulat naturel que le réseau professeur propose une version plus aboutie de la segmentation, notamment en forçant le remplissage de l'objet. L'étudiant doit donc parvenir à effectuer la même tâche que son professeur afin d'améliorer ses performances et pour qu'au final, la tâche de rectification du professeur devienne inexistante.

La seconde fonction $L_{\text{ae}}$ a de multiples usages. On pourrait penser dans un premier temps que l'unique objectif est d'insérer notre corpus d'images non étiquetées dans la boucle grâce à l'autoencodeur issu de la combinaison de nos différents réseaux. Même si cela est exact, cela n'est pas le seul avantage. En effet, ce terme permet également de compenser les "attracteurs négatifs" afin d'assurer une convergence globale et satisfaisante du protocole. Par attracteur négatif, on parle d'un certain nombre de cas particuliers qui provoquent un effet de plateaux généralement impossible à éviter en faisant confiance à la descente de gradient ainsi qu'aux optimiseurs associés (RMSprop, ADAM). Prenons un exemple simple et imaginons que le réseau étudiant retourne une image vide, le professeur ne voyant aucune cellule à remplir va naturellement retourner une image vide. De ce fait, $L_{cons}$ est parfaitement minimisée. En pratique, ce cas arrive très souvent car notre corpus d'images annotées $S_{\text{tr}}$ est de faible dimension, $f_{\boldsymbol{\theta}}^{(\text{etu})}$ a donc assez de capacité pour répondre correctement sur les données d'entraînement, et ensuite renvoyer une image entièrement vide pour n'importe quelle autre entrée. Une compétition va alors s'installer entre $L_{\text{cons}}$ et $L_{\text{ae}}$. En effet, une image noire à la sortie du professeur va être traduite par un fond sans cellule par $f^{(\text{inv})}$, et $L_{\text{ae}}$ va alors être impactée négativement. Pour mieux gérer cette compétition, dont l'objectif principal est d'éviter les minima locaux et effets de plateaux indésirables, un nouvel hyperparamètre noté $\lambda_{\text{ae}}$ à régler par l'utilisateur va être ajouté à $L_{\text{cons}}$. On en vient alors à la fonction de perte globale à minimiser dans cette 4ème et dernière phase du protocole, qu'on note $L_{\text{glob}}$.

$$L_{\text{glob}}\left(\boldsymbol{\theta}\right)=\arg\min_{\boldsymbol{\theta}}\ L_{\text{etu}}\left(\boldsymbol{\theta};\mathcal{S}_{\text{tr}}\right)+L_{\text{cons}}\left(\boldsymbol{\theta};\mathcal{U}_{\text{tr}}\right)+\lambda_{\text{ae}}L_{\text{ae}}\left(\boldsymbol{\theta};\mathcal{U}_{\text{tr}}\right) \tag{5.6}$$

Cette phase d'apprentissage est assez ambitieuse et utilise les ensembles de validation $S_{\text{val}}$ et $U_{\text{val}}$ pour effectuer de l'*early stopping*. L'architecture globale, ainsi que l'implication des différentes fonctions de perte est résumée en figure 5.6.

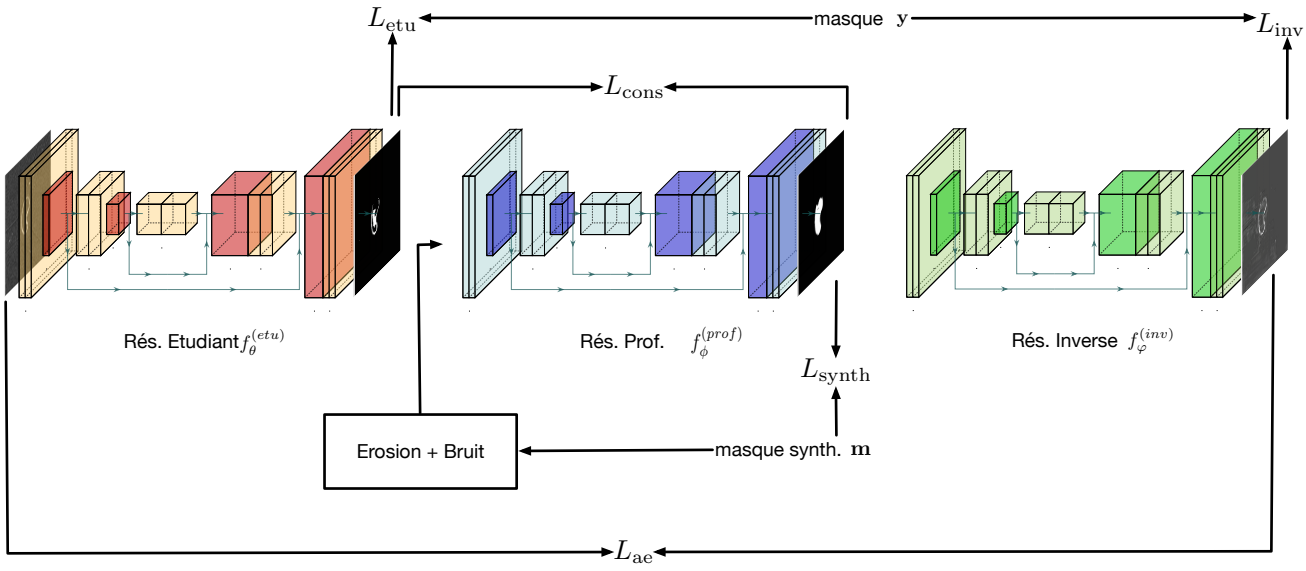


FIGURE 5.6 – Combinaison des trois réseaux pour former un autoencodeur, bénéficiant également des données non étiquetées.

### 5.3.7 Intégration d'un Curriculum Learning

La minimisation de $L_{\text{glob}}$ n'est pas une tâche facile. Au delà de l'ajout du terme $L_{\text{ae}}$ permettant d'éviter en partie certains attracteurs négatifs, il existe tout de même des risques non négligeables que l'apprentissage bloque. L'apprentissage est en réalité fortement dépendant de l'ordre dans lequel les données parcourent le réseau. Si la partie non supervisée de l'apprentissage propose des données dans lesquelles la cellule est trop petite ou peu présente, alors la fonction de perte $L_{\text{cons}}$ va forcer le réseau à renvoyer une image vide. Par conséquent, il est primordial de s'intéresser à cet ordre pour faire tourner notre boucle d'apprentissage, ce qui implique l'utilisation d'un Curriculum Learning (CL, c.f. section 3.2.2). Il est à noter que le CL n'est utilisé que sur l'ensemble $U_{\text{tr}}$. Dans ce type d'approche, la problématique majeure consiste à distinguer ce qu'est une donnée facile, d'une complexe.
Après plusieurs tests, il semblerait qu'une bonne heuristique doit prendre en considération la taille de la cellule. Plus une cellule est grande, mieux notre protocole fonctionne. Le souci provient du fait qu'il n'est possible de connaître la taille de la cellule sans avoir accès à l'étiquetage, et l'heuristique fonctionne uniquement sur la partie non supervisée de l'approche. Par conséquent, une stratégie de retrait du fond est appliquée, l'objectif est de s'assurer que les pixels du fond soient plus foncés que ceux correspondant au contour de la cellule [2]. Si le fond des images a une variance faible et des valeurs proches de zéro, alors une image contenant une grosse cellule a naturellement une norme $\mathsf{L}_1$ plus élevée. Le deuxième paramètre (également très important) est la différence entre l'image initiale $\mathbf{x}$ et la sortie du réseau

2. Dans le cas inverse, il suffirait de travailler avec $1 - \mathbf{x}$.

inverse qu'on note $f^{(\text{inv})}(f^{(\text{prof})}(f^{(\text{etu})}(\mathbf{x})))$. On souhaite minimiser cette distance puisque cela signifie que le réseau professeur est probablement performant sur cette donnée, ce qui la place assez logiquement dans la catégorie des images faciles à segmenter. Finalement, le critère permettant de trier chaque donnée est défini par la formule suivante :

$$s(\mathbf{x}) = -\frac{\left\| f_{\boldsymbol{\varphi}}^{(\text{inv})}\left(f_{\boldsymbol{\phi}}^{(\text{prof})}\left(f_{\boldsymbol{\theta}}^{(\text{etu})}(\mathbf{x})\right)\right) - \mathbf{x}\right\|_1}{\|\mathbf{x}\|_1}. \tag{5.7}$$

Plus il est élevé, plus la donnée est facile à traiter et doit être présentée au réseau rapidement. Afin de procéder à l'apprentissage de notre modèle, nous allons initialement utiliser 30% des données non supervisées obtenant le meilleur score $s(\mathbf{x})$.
Après une première phase d'apprentissage, nous allons réutiliser l'heuristique présentée ci-dessus pour re-calculer le critère sur chacune des données non étiquetées restantes (parmi les 70%). Ensuite, nous allons ajouter les 7% les plus éligibles, et ce processus sera itéré 10 fois afin de pouvoir utiliser l'intégralité de nos données lors d'un dernier apprentissage. Nous allons également sauvegarder les poids des réseaux à la fin de chaque apprentissage. Comme $f_{\boldsymbol{\phi}}^{(\text{prof})}$ et $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ ont des poids gelés, il ne suffit alors que d'enregistrer ceux de l'étudiant.

À la fin de la boucle, il faut retenir un modèle parmi les 11 entraînés, et ce, sans utiliser davantage de données étiquetées. Cette étape de sélection est très importante car le dernier réseau ayant appris sur toutes les données n'est généralement pas le meilleur, les dernières données ajoutées sont souvent de mauvaises qualités (ce sont les 7% les pires) et peuvent détériorer l'apprentissage. Le modèle retenu sera celui qui minimise la fonction de perte de validation notée $L_{\text{val}}$ qui est définie comme suit :

$$L_{\text{val}} = L_{\text{etu}}(\boldsymbol{\theta}; \mathcal{S}_{\text{val}}) + L_{\text{cons}}(\boldsymbol{\theta}; \mathcal{U}_{\text{val}}) + L_{\text{ae}}(\boldsymbol{\theta}; \mathcal{U}_{\text{val}}). \tag{5.8}$$

Les ensembles de validation sont donc utilisés pour les *early stoppings* à l'échelle des différentes fonctions de perte, mais ils servent également pour la sélection du meilleur modèle via la minimisation de $L_{\text{val}}$.

## 5.4 Expériences

### 5.4.1 Description des ensembles de données et pré-traitements.

Nous avons testé notre stratégie sur deux jeux de données respectivement. Le premier n'est autre que le jeu de données Capsule [107] déjà détaillé en section 4.4.1. Ce jeu de données est particulièrement intéressant car il est la base d'une collaboration avec l'UTC de Compiègne. Le protocole auto-supervisé présenté dans ce chapitre a été essentiellement pensé pour travailler avec ces données. Notre approche est sensible au nombre de cellules que possède l'image, elle ne peut s'appliquer en l'état qu'à des jeux de données ne contenant qu'une seule cellule par image. Cette limitation forte fera l'objet d'une discussion dans la section dédiée. Le jeu de données Capsule est

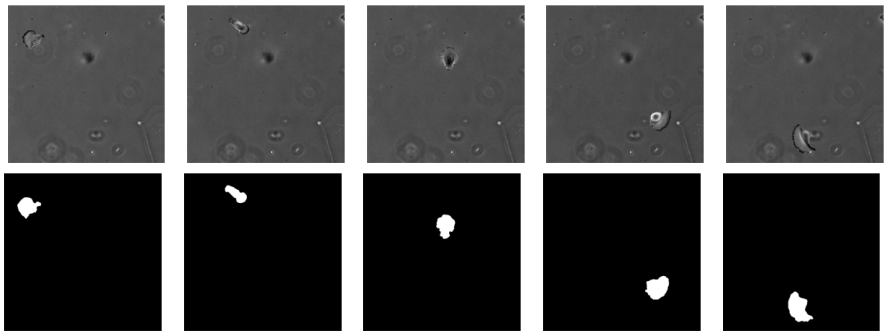

FIGURE 5.7 – Quintuplet de couples (image-étiquetage) isolant chaque cellule d'une donnée initiale provenant du jeu de données PhC-C2DH-U373.

donc naturellement attrayant car il ne possède qu'un seul objet à segmenter, une micro-capsule qui parcourt un tube bi-directionnel.
Un second jeu de données, semi-artificiel est utilisé pour évaluer les performances de notre protocole. On entend par semi-artificiel le fait qu'il se base sur le jeu de données PhC-C2DH-U373 qui est également présenté en section 4.4.1. Les modifications apportées à ce dernier ont pour objectif de forcer chaque image à n'avoir qu'une seule cellule à segmenter, toujours en relation avec les limitations de notre modèle. Pour ce faire, le calcul des pixels médians qu'on notera $\mathbf{b}$ est effectué à partir des 115 données fournies et correspond approximativement au fond des images. Chaque image contient au minimum 5 cellules, que nous allons isoler via l'utilisation de masques binaires comme suit.

$$\mathbf{x} = \mathbf{y} \odot \mathbf{o} + (1 - \mathbf{y}) \odot \mathbf{b} \tag{5.9}$$

Avec $\mathbf{y}$ le masque binaire de la cellule à isoler. L'effet de cette opération est trivial, on garde les valeurs de l'image initiale à l'endroit où la cellule est présente, et on remplace les autres pixels par la valeur médiane $\mathbf{b}$. Un dataset de 575 (5 x 115) paires ($\mathbf{x}$,$\mathbf{y}$) est alors généré, dont 5 couples provenant d'une même donnée sont montrés en figure 5.7.

Par ailleurs, un pré-traitement est utilisé sur toutes les données faisant l'objet d'expériences dans ce chapitre. Il consiste à retirer le fond des images, afin de pouvoir permettre notamment à notre heuristique à la base du *Curriculum Learning* de fonctionner correctement. Le calcul des nouvelles images se fait par la soustraction entre une image et la médiane calculée sur l'ensemble du jeu de données. En notant $\bar{\mathbf{x}}$ une image ayant subit le pré-traitement, des triplets ($\mathbf{x}$,$\bar{\mathbf{x}}$,$\mathbf{y}$) basés sur le dataset Capsule sont montrés en figure 5.8.

### 5.4.2 Modalités des expériences

Avant de parler plus en détails des expériences réalisées, il est nécessaire de définir en amont un certain nombre de métriques, et de paramètres généralement communs aux différents tests.
La métrique utilisée pour évaluer les performances et un indice de Jaccard

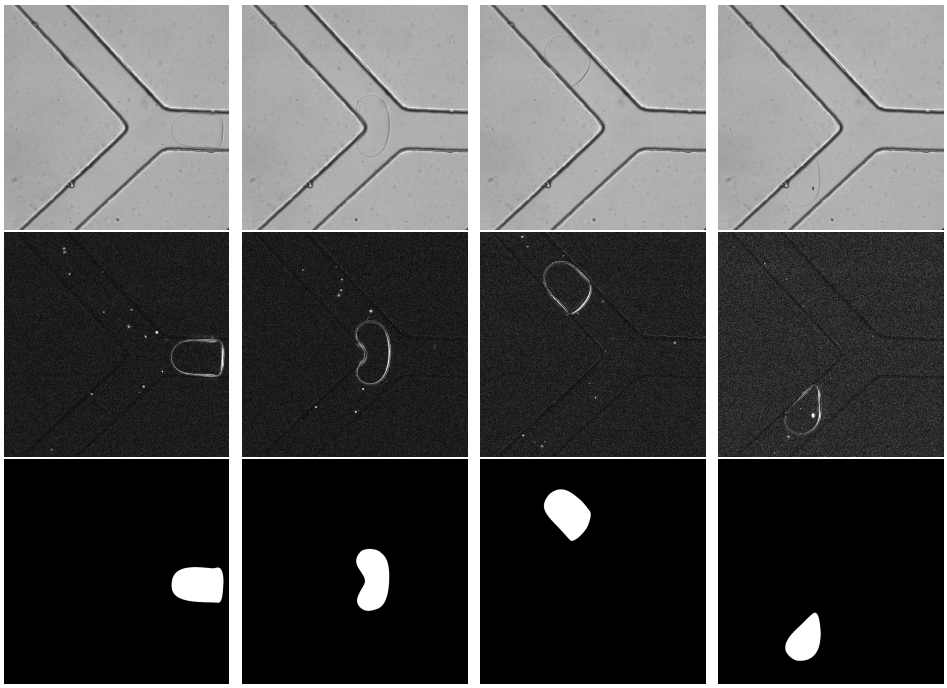

FIGURE 5.8 – Images du jeu de données Capsule, affichées dans l'ordre suivant (de haut en bas) : Image initiale, Image pré-traitée et étiquetage.

(voir section 4.4.2), également utilisé pour l'évaluation de la stratégie proposée au chapitre précédent.

Les réseaux utilisés sont tous des U-nets, avec la même logique de répartition des filtres au sein des couches. Après l'application d'une couche de *pooling*, les quantités de filtres des couches de convolution vont doubler. Une symétrie est établie pour le nombre de filtres des couches de convolution entre la partie encodeur et décodeur respectivement. Une application de 3 couches de *poolings*, et de ré-échantillonnage est effectuée au sein des réseaux. Les couches de convolution utilisent également un *padding* afin de ne pas réduire la taille de l'image, avec un noyau pour chaque filtre de dimension $3 \times 3$. Un paramètre qu'on notera $F$ défini le nombre de filtres de la première couche de convolution du réseau. Ce paramètre est égal à 5 pour les réseaux $f_{\boldsymbol{\theta}}^{(\text{etu})}$ et $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ et à 30 pour $f_{\boldsymbol{\phi}}^{(\text{prof})}$. Si une modification de ce paramètre a lieu, elle sera spécifiée directement dans la section dédiée à l'expérience. Afin de donner un ordre de grandeur, $f^{(\text{etu})}$ et $f^{(\text{inv})}$ possèdent 192.026 paramètres, tandis que $f^{(\text{prof})}$ est à 6.896.401.
Les deux couches de *dropout* situées au milieu du modèle et incluses dans U-net "vanille" sont retirées. Le *pooling* utilisé est de dimension $4 \times 4$ contrairement aux expériences du chapitre précédent.
Pour l'entraînement des réseaux, le *learning rate* $\eta$ est fixé à $1e^{-4}$. L'optimiseur utilisé est RMSprop (c.f. section 3.2.1.2) avec un paramètre $\alpha$ à 0.9. Un *early stopping* basé sur les différents jeux de validation $\mathcal{S}_{\text{val}}$ et $\mathcal{U}_{\text{val}}$ est de mise. Une taille de 100 est fixée pour la dimension des *batchs* de $f_{\boldsymbol{\theta}}^{(etu)}$ et $f_{\boldsymbol{\varphi}}^{(\text{inv})}$.

Le réseau $f_{\boldsymbol{\phi}}^{(\text{prof})}$ étant générique, les spécifications de son entraînement ne dépendent pas d'une expérience précise et sont indiquées ci-dessous. Pour l'apprentissage du professeur, la patience de l'*early stopping* est de 500, ce qui force la généralisation au détriment d'un temps de calcul assez intensif (2500 à 3000 *epochs*). Puisque ce réseau est générique, il est raisonnable d'investir des ressources pour s'assurer de son bon fonctionnement. Son ensemble de validation, également obtenu grâce au générateur d'images artificielles d'ellipse est de taille 300. Enfin la quantité d'images vue pendant une *epoch* est également paramétrée à 300. On fait appel au générateur à chaque nouvelle *epoch* pour renouveler ces données et améliorer l'apprentissage qui se rapproche alors d'une minimisation du risque réel et non pas du seul risque empirique.

Une stratégie de *self-learning* [136] (section 5.2.2) est mise en place afin de comparer les résultats avec une stratégie concurrente. Cette stratégie consiste à apprendre un réseau professeur sur l'ensemble $\mathcal{S}_{\text{tr}}$ selon les mêmes modalités que pour l'apprentissage de $f_{\boldsymbol{\theta}}^{(\text{etu})}$. Ensuite, ce réseau va alors effectuer une prédiction sur l'ensemble $\mathcal{U}_{\text{tr}}$. Ces nouveaux étiquetages vont permettre d'obtenir un corpus d'images annotées bien plus important, qui sera ensuite utilisé pour apprendre le réseau étudiant. Ce dernier apprentissage se passe dans les mêmes conditions que celui effectif dans notre protocole (même architecture, même *early stopping* etc...). Cette approche sera dénommée *self-training* dans le reste du manuscrit.

### 5.4.3 Validation du protocole sur les données de Capsule

Les tests effectués sur les données de Capsule prennent en considération un ensemble $\mathcal{S}_{\text{tr}}$ constitué respectivement de 2, 3, 4, 5 et 6 données. L'ensemble de validation $\mathcal{S}_{\text{val}}$ ne possède qu'une seule image. Pour la partie non supervisée, $\mathcal{U}_{\text{tr}}$ contient 281 images, et la validation associée $\mathcal{U}_{\text{val}}$ est de taille 20. Les réseaux $f_{\boldsymbol{\theta}}^{(\text{etu})}$ et $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ apprennent avec pour condition d'arrêt un ES de patience 20. Dans la dernière boucle d'apprentissage, $f^{(\text{AE})}$ va utiliser un *early stopping* de patience 40, cela est nécessaire car l'amélioration des performances du réseau par l'ajout de données non étiquetées demande généralement plus de temps que lors d'un apprentissage entièrement supervisé.

Afin de visualiser les interactions des différents réseaux, la figure 5.9 montre la sortie proposée pour chaque réseau, en fonction d'une image d'entrée affichée en premier lieu.
Les résultats pour ce dataset sont montrés dans la table 5.1. Ils proviennent d'une moyenne sur 5 exécution de l'algorithme. On peut remarquer que la stratégie concurrente basée sur le *self-learning* n'améliore pas ou peu les résultats de U-net. Ceci peut s'expliquer par le fait que lorsque le nombre d'images annotées est trop restreint, les prédictions du réseau professeur sont fortement erronées. L'ajout de ces nouvelles annotations au sein du jeu de données initial peut alors être considéré comme du bruit.
Les performances de notre stratégie sont remarquables, allant de +27.09%

(A) $\mathbf{x}$ (B) $f_\theta^{(\text{etu})}(\mathbf{x})$ (C) $f_\phi^{(\text{prof})}\left(f_\theta^{(\text{etu})}(\mathbf{x})\right)$ (D) $f_\varphi^{(\text{inv})}\left(f_\phi^{(\text{prof})}\left(f_\theta^{(\text{etu})}(\mathbf{x})\right)\right)$

FIGURE 5.9 – Exemple de sortie des 3 réseaux en fonction d'une image initiale de Capsule.

à +36.07% de gain (en absolu). Ces résultats significatifs prouvent en partie la validité de notre protocole. Il est à noter qu'en réduisant le nombre de données annotées à une seule, notre stratégie est fortement impactée négativement par des soucis de convergence.

| $n_{\text{s-tr}} + n_{\text{s-val}}$ | $\lambda_{\text{ae}}$ | U-net | Self-trained U-net | Auto-encadré U-net (notre solution) |
|---|---|---|---|---|
| 3 | 20 | 27.05% | 23.63% | 54.86% |
| 4 | 5 | 41.67% | 41.01% | 78.64% |
| 5 | 100 | 52.67% | 51.18% | 80.77% |
| 6 | 20 | 56.77% | 56.09% | 82.23% |
| 7 | 5 | 55.14% | 55.38% | 82.43% |

TABLE 5.1 – Indice de Jaccard pour la segmentation du jeu de données Capsule pour 3 stratégies différentes. U-net fait référence au score du réseau étudiant après la première phase d'apprentissage. Self-training U-net est le score de la stratégie concurrente. Auto-encadré U-net correspond à la stratégie proposée dans ce chapitre.

### 5.4.4 Augmentation du covariate shift au sein du dataset Capsule

Le *covariate shift* est l'une des situations les plus complexes en apprentissage. Il se traduit par un fort décalage entre l'ensemble d'entraînement et celui de test. On souhaite vérifier la robustesse de notre protocole face à ce type de situation. Notre ensemble de données Capsule représente la traversée d'une micro-capsule dans un canal bi-directionnel. La micro-capsule est originaire de la partie droite de l'image, elle avance jusqu'à être confrontée à un obstacle qui va la faire dévier soit vers la partie haute de l'image, soit vers la partie basse. Par conséquent, il est possible de diviser complètement notre jeu de données initial. En considérant que toutes les données avant la collision sont des données d'entraînement, que le déplacement vers le haut de la micro-capsule est associé à l'ensemble validation et que les dernières données sont consacrées à l'ensemble de test, on obtient alors un dataset ayant un niveau conséquent de *covariate shift*. A noter que l'ensemble de test est alors composé de 55 données alors que $\mathcal{S}_{\text{tr}}$ dispose de 2, 4, 5, 6, ou 7 images

et que $\mathcal{U}_{\text{tr}}$ passe à 198 données. Enfin, $\mathcal{S}_{\text{val}}$ et $\mathcal{U}_{\text{val}}$ sont toujours fixés à 1 et 20 données respectivement. $\mathcal{U}_{\text{tr}}$ contient des images dont la capsule est située dans la partie haute, ainsi que la partie centrale de l'image.

La table 5.2 détaille les résultats de cette nouvelle expérience. On constate que la stratégie concurrente semble légèrement mieux fonctionner, mais ne parvient pas égaler les scores obtenus par l'approche proposée dans ce chapitre. Les gains sur le réseau $f_{\theta}^{(\text{etu})}$ sont compris entre 17.59% et 28.83%. On peut également vérifier que le problème de segmentation est plus complexe dans ce cas de figure, les nouveaux résultats proposés ont des valeurs sensiblement réduites.

| $n_{\text{s-tr}} + n_{\text{s-val}}$ | $\lambda_{\text{ae}}$ | U-net | Self-trained U-net | Auto-encadré U-net (notre solution) |
|---|---|---|---|---|
| 3 | 10 | 14.31% | 16.11% | 36.75% |
| 5 | 5 | 29.36% | 30.35% | 46.95% |
| 6 | 10 | 28.33% | 37.28% | 55.61% |
| 7 | 5 | 34.49% | 34.18% | 58.69% |
| 8 | 5 | 23.53% | 30.88% | 52.36% |

TABLE 5.2 – Indice de Jaccard pour la segmentation du jeu de données Capsule, avec utilisation d'un partage des données visant à maximiser le *covariate shift*.

### 5.4.5 Validation du protocole sur les données artificielles provenant de PhC-C2DH-U373

Changer de jeu de données permet de conforter la validation expérimentale du notre protocole, mais cela permet également de vérifier la transférabilité du réseau professeur, étant normalement capable de fonctionner peu importe le jeu de données. Un effet de *covariate shift* est également ajouté au dataset artificiel. Il se base sur le fait que les données provenant du Cell Tracking Challenge de l'ISBI permettent le traçage des cellules au sein des images. Sur les 575 images disponibles, il est possible de retrouver les 115 images contenant la première cellule, puis les 115 contenant la seconde, etc. De cette manière, il est possible de forcer les données d'entraînement à ne contenir uniquement des images contenant la cellule numérotée 1, la validation va prendre en compte celles contenants les cellules 2 et 3, alors que l'ensemble de test va récupérer celles associées aux deux dernières. Une permutation des cellules permettra d'obtenir une moyenne sur 5 exécutions, avec un pseudo-mélange des cellules associées aux différents ensembles.

Ce problème d'apprentissage est bien plus complexe que celui des Capsules, ceci peut s'expliquer car certaines cellules, comme la première, vont se déplacer dans les parties hautes de l'image sans jamais descendre. De ce fait, elle ne pourra jamais aller dans un espace ou est présent la cellule numéroté 4, située dans un coin isolé en bas de l'image. C'est pourquoi le nombre de

données de $\mathcal{S}_{\text{tr}}$ passe à 10. $\mathcal{S}_{\text{val}}$ contient 5 données et $\mathcal{U}_{\text{tr}}$ possède au total l'équivalent de 3 cellules, soit 325 données (il faut retirer la partir de validation). $\mathcal{U}_{\text{val}}$ va alors prendre les 20 données restantes. L'ensemble de test comprend les 230 données associées aux deux dernières cellules. De plus, le nombre de filtres des couches noté $F$ des réseaux $f_{\boldsymbol{\theta}}^{(\text{etu})}$ et $f_{\boldsymbol{\varphi}}(\text{inv})$ va passer de 5 à 10, ce qui implique une forte augmentation du nombre de paramètres, de l'ordre du quadratique. Les résultats ne sont disponibles que pour un seul nombre précis de données supervisées utilisées et la table ci-dessous les indiquent.

| $n_{\text{s-tr}} + n_{\text{s-val}}$ | $\lambda_{\text{ae}}$ | U-net | Auto-encadré U-net (notre solution) |
|---|---|---|---|
| 15 | 10 | 26.45% | 81.16% |

TABLE 5.3 – Indice de Jaccard pour la segmentation du jeu de données PhC-C2DH-U373.

Un écart significatif de 54.69% est constaté sur l'indice de Jaccard. Ceci prouve encore une fois l'efficacité de notre stratégie dite auto-encadrée, et ce pour l'utilisation d'un réseau professeur identique.

## 5.5 Discussions

La stratégie proposée semble être convaincante de part ses résultats significatifs. Cependant, elle peut faire l'objet de nombreux questionnements dont une partie est listée ci-dessous.

— Cette stratégie permet d'apprendre avec un nombre très faible de données, actuellement c'est l'augmentation de données massive qui prime. Si cette approche n'est pas compatible avec la DA, les utilisateurs vont toujours préférer utiliser la DA qui est plus simple à mettre en œuvre.

— La stratégie proposée repose sur un réseau $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ qui nécessite de reconstruire l'image de cellule. Si le signal associé à cette cellule ou au fond varie fortement dans le jeu de données initial, la sortie générée par le réseau pour la reconstruire ne va-t-elle pas générer des blocages dans l'apprentissage ? Dans ce cas, le cadre d'utilisation de l'approche doit-il se restreinte à des données dont l'objet à segmenter ne subit que des modifications visuellement légères ?

— Le protocole expliqué doit utiliser des images ne contenant qu'une seule cellule. Le problème est que ce type de jeu de données est assez rare, notamment car la stratégie cible le biomédical. Une version multi-cellules de l'approche est-elle envisageable ?

Cette section vise à répondre en partie à ces questionnements par le biais de quelques expériences supplémentaires ou réflexions.

### 5.5.1 Couplage du protocole avec l'augmentation de données

Comme le questionnement plus haut l'a suggéré, si notre stratégie n'est pas compatible avec l'augmentation de données, il est fort probable qu'elle soit

peu considérée par les potentiels utilisateurs. C'est pourquoi différentes expériences d'augmentation de données ont été produites pour démontrer la compatibilité des deux approches.

Les tests utilisent le jeux de données Capsule avec ajout d'un *covariate shift* (c.f. 5.4.4 pour la composition détaillée de ce jeu). La différence majeure va se situer sur le fait que $f_{\boldsymbol{\theta}}^{(\text{etu})}$ va apprendre sur une version augmentée de $\mathcal{S}_{\text{tr}}$. L'augmentation de données se base sur une partie rotation, ainsi qu'une de bruitage :

— La rotation a une chance sur deux de s'effectuer. Dans le cas suivant, la rotation cible des angles à 0, 90, 180 et 270 degrés. Enfin, une symétrie horizontale est également possible avec probabilité un sur deux. On distingue 8 combinaisons dont la sélection se fait via une loi uniforme.
— L'ajout de bruit a également une chance sur deux d'être présent. Une loi uniforme permet de sélectionner le bruit parmi : poivre et sel sur 5% des pixels, multiplicatif centré gaussien avec 0.15 d'écart-type ou additif uniforme compris entre [-0.5,0.5].

À savoir que l'utilisation de cette augmentation de données se fait également dans l'entraînement de $f^{(\text{AE})}$. Une question assez ouverte demeure sur l'utilisation de l'augmentation de données sur la partie non supervisée de l'approche. Un ajout de bruit peut sembler contre-productif pour la fonction de perte $L_{\text{ae}}$ qui va chercher à reconstruire l'image initiale. Nous avons donc essayé les deux approches, avec et sans augmentation de données sur l'ensemble $\mathcal{U}_{\text{tr}}$. Concernant cette augmentation, seule la partie sur les rotations a été conservée afin d'éviter d'inclure une augmentation de données pouvant être destructrice.

Enfin, la taille de l'ensemble $\mathcal{S}_{\text{tr}}$ est réduite à 2, alors que $\mathcal{S}_{\text{val}}$ reste à une image. Cette réduction vient directement compenser l'amélioration des performances du réseau étudiant grâce à l'augmentation de données. Les valeurs concernant les ensembles non étiquetés ne changent pas. Les paramètres associés à l'*early stopping* se voient également modifiés. Pour que l'augmentation de données puisse avoir le temps de générer assez d'images différentes, la patience a été augmentée pour atteindre 40, 40 et 100 respectivement pour $f_{\boldsymbol{\theta}}^{(\text{etu})}$, $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ et $f^{(\text{AE})}$.

| U-net | U-net + aug. | Auto-encadré U-net + aug. sur $\mathcal{S}_{\text{tr}}$ | Auto-encadré U-net + aug. complète |
|---|---|---|---|
| 14.31% | 40.27% | 67.39% ($\lambda_{\text{ae}} = 20$) | 65.55% ($\lambda_{\text{ae}} = 5$) |

TABLE 5.4 – Indice de Jaccard sur le *dataset* Capsule avec augmentation du *covariate shift*. La table montre respectivement les performances du réseau seul, puis avec ajout d'une augmentation de données sur la partie supervisée, ainsi que le couplage avec l'augmentation de données utilisée sur la partie non supervisée. Les expériences considèrent $n_{\text{s-tr}} + n_{\text{s-val}} = 3$ données étiquetées.

Les résultats sont présentés dans la table 5.4. A noter que notre stratégie initiale sur 3 données obtient le score de 36.75% (voir table 5.2). On peut constater que notre stratégie améliore les performances d'un réseau ayant appris initialement à l'aide d'une augmentation de données. De plus, le gain de performance de 27.12% est significatif.

Ces résultats suscitent également de nombreuses questions, notamment sur la performance de notre protocole avec utilisation de l'augmentation sur l'ensemble $\mathcal{U}_{\text{tr}}$ qui semble moins bien fonctionner. Ce faible écart est-il inclut au sein de la variance générée par les tests ? Une autre hypothèse, plus intéressante se base sur la forte sensibilité de l'autoencodeur global face à la corruption de données. Il est possible que la plupart des augmentations de données soient contre-productives dans ce cas précis. Par conséquent, il pourrait être envisageable d'utiliser une stratégie de sélection en ligne d'augmentations de données (telle que celle introduite au chapitre précédent) pour améliorer les performances du protocole.

### 5.5.2 Robustesse du réseau inverse

Il est tout à fait normal de questionner le réseau inverse sur sa robustesse aux fortes modifications liées à l'objet à segmenter. Ce réseau va générer une image dans lequel l'objet à segmenter sera très souvent remplacé par une représentation moyenne. Par conséquent, si cette représentation est trop éloignée de l'image initiale, la fonction de perte $L_{\text{AE}}$ va forcément être caduque et elle ne pourra pas compenser efficacement $L_{\text{cons}}$. Pour le jeu de données Capsule, le problème n'existe pas car les contours des micro-capsules sont très proches et l'intérieur de ces mêmes capsules est transparent. Pour le dataset provenant de PhC-C2DH-U373, l'intérieur ainsi que le contour de chaque cellule possèdent des différences.

La figure 5.10 présente une visualisation complète de l'image d'entrée, de l'étiquetage ainsi que de la sortie du réseau inverse sur ce même étiquetage, et ce pour 4 données distinctes. La première est issue de l'ensemble d'entraînement alors que les trois autres proviennent de l'ensemble de validation. On remarque, pour une donnée d'entraînement, que le réseau $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ s'est contenté de retourner parfaitement l'image d'entrée. On distingue cependant une nette différence sur les données de validation. Le réseau $f_{\boldsymbol{\varphi}}^{(\text{inv})}$ semble proposer une cellule dont la nuance de gris est assez éloignée de l'image initiale. Cependant, on sait de part la section 5.4.5 que les performances obtenues sur ce jeu de données sont favorables au bon fonctionnement du protocole. Une explication possible provient du fait que, à partir du moment où le fond moyenné de l'objet reconstruit est plus proche de l'objet initial que du fond de l'image, alors $L_{\text{AE}}$ devrait avoir un impact positif dans le protocole. Par conséquent, des jeux de données présentant des cellules différentes mais dont la moyenne est extrêmement proche des pixels du fond de l'image risque de poser problème. Dans ce cas précis, il semble nécessaire d'utiliser des stratagèmes de pré-traitements adaptés permettant de contourner au mieux le problème.

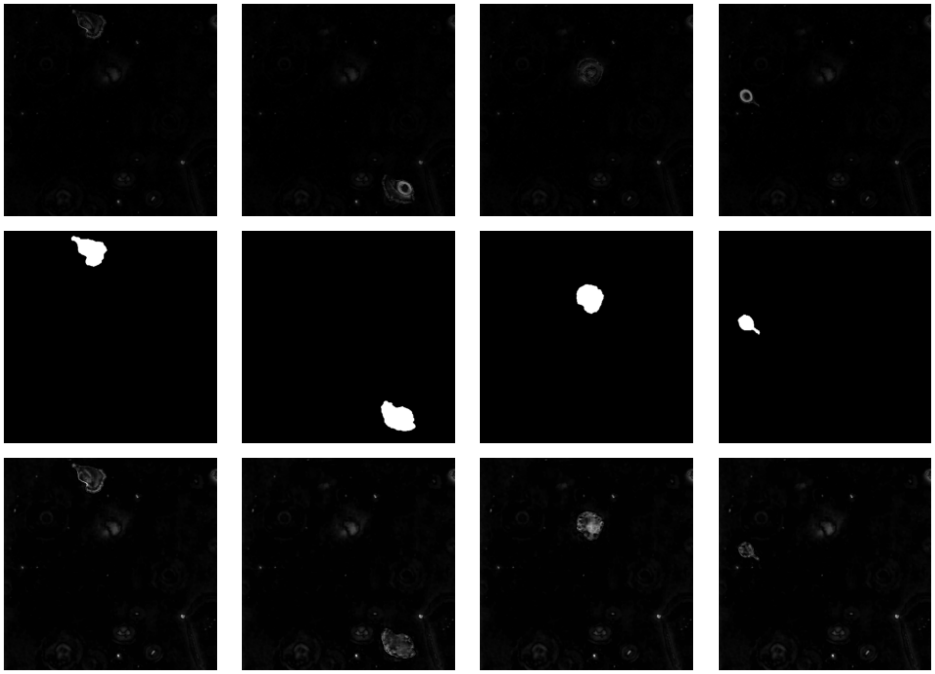

FIGURE 5.10 – Images du jeu de données PhC-C2DH-U373 avec isolement des cellules. L'affichage proposé est dans l'ordre suivant (de haut en bas) : Image initiale, Image étiquetage et sortie du réseau $f_{\varphi}^{(\text{inv})}$ sur l'étiquetage.

### 5.5.3 Utilisation de données multicellulaires

Le protocole présenté actuellement ne peut s'exécuter que sur des données monocellulaires. La contrainte majeure se situe au niveau de l'apprentissage du réseau professeur reposant sur une génération massive de données d'ellipses déformées. Un test primordial consiste à apprendre le réseau $f_{\phi}^{(\text{prof})}$ sur des données synthétiques présentant plusieurs ellipses par image, et d'incorporer ce nouveau professeur dans notre stratégie.

Le jeu de données choisi est PhC-C2DH-U373 (sans aucune modification). Un réseau étudiant apprend sur le jeu de données à partir de 5 paires d'images annotées. La figure montre entre autres l'image initiale, l'étiquetage ainsi qu'une prédiction proposée par le réseau étudiant après son apprentissage. On constate que la prédiction contient de nombreux trous ainsi que des cellules à compléter et donc que le postulat initial sur lequel repose notre stratégie semble encore valide.
L'entraînement du réseau professeur utilise la même augmentation de données que pour la partie monocellulaire. L'unique différence provient de la génération aléatoire de l'ellipse dans l'image, qui peut varier entre une et cinq. Dans le cas ou plusieurs ellipses sont générées, les déformations élastiques vont toutes les cibler. Sur les 115 données initiales, 6 sont utilisées pour la partie supervisée (5+1) et 20 sont mises de côté pour l'ensemble de test. Les 89 données restantes sont incluses dans l'ensemble non étiqueté avec une répartition (69,20) pour $\mathcal{U}_{\text{tr}}$ et $\mathcal{U}_{\text{val}}$ respectivement.

(A) $x$ (B) $y$ (C) $f_{\phi}^{(\text{etu})}(x)$

FIGURE 5.11 – Triplet d'images représentant le *dataset* PhC-C2DH-U373, avec une image d'entrée et son annotation. La dernière image représente la sortie du réseau étudiant à partir de l'image d'entrée.

Après l'application du protocole entier, les résultats obtenus ne permettent pas d'améliorer la performance du réseau étudiant, et ce malgré le fait qu'il présente tous les défauts devant logiquement être compensés par notre stratégie. Le problème majeur qui invalide le protocole actuel pour ce cas d'usage provient du réseau professeur. Si par exemple le professeur doit corriger une image contenant deux cellules proches, comment peut-il savoir si ce sont deux cellules distinctes ou bien si c'est une seule cellule dont l'intérieur n'est pas bien rempli ? Ce problème pratique ne semble même pas solvable pour un humain. Le réseau professeur ne peut donc pas généraliser la tâche de remplissage ce qui engendre des segmentations erronées qui bloquent l'entièreté du protocole.

Une idée permettant de résoudre le problème pourrait être d'utiliser un réseau Mask R-CNN [45] en lieu et place de U-net. Ce nouveau réseau a pour avantage d'être capable de pouvoir identifier et encapsuler les différents objets d'une image (segmentation d'instances). Remplacer le réseau étudiant actuel par un Mask R-CNN permettrait alors de pouvoir isoler les différentes cellules de notre image avant de pouvoir exploiter la correction du professeur. Une telle approche n'est cependant pas immédiate à mettre en place, et n'a pas pu être testée lors de la rédaction de ce manuscrit. Elle reste cependant en ligne principale des futurs travaux à réaliser.

## 5.6 Conclusion

Ce chapitre a introduit un protocole, dit d'apprentissage auto-encadré, émargeant dans la thématique du *self-learning* afin d'améliorer les performances d'un réseau U-net. Cet algorithme s'utilise dans un cadre de *few shot learning* sévère où les performances des réseaux profonds sont fortement amoindries. La stratégie décrite utilise un réseau dit professeur ayant pour objectif de corriger les défauts de U-net. Un troisième réseau est introduit afin d'inverser la sortie du professeur, et de créer un autoencodeur résultant de l'application successive des trois réseaux. Cet autoencodeur sera la base d'intégration

d'un vaste corpus de données non étiquetées. Ces données, classées dans un ordre précis grâce à une heuristique de *curriculum learning* permettent notamment avec l'aide du réseau professeur, de corriger positivement les poids du réseau étudiant afin d'atteindre de meilleures performances.

Ce protocole propose de multiples avantages. Le gain de performances est significatif, et le réseau professeur est générique. L'unique objectif de l'algorithme est d'améliorer le réseau étudiant, généralement constitué d'un faible nombre de paramètres. Son utilisation est alors très peu coûteuse. Il est à noter que la stratégie proposée se couple efficacement avec l'augmentation de données, très utilisée dans le cadre du *few shot learning*.

Enfin, des discussions sur les éventuelles faiblesses du protocole ont été évoquées. Le réseau responsable d'inverser la réponse du professeur souffre de difficultés lors de son apprentissage. Des cas particuliers peuvent également l'empêcher de fonctionner correctement. Enfin, l'intégration de ces efforts pour la résolution des problèmes multicellulaires n'est pas encore résolue. Des travaux plus ambitieux sont nécessaires et font parties des futurs axes de recherche à développer.

# Chapitre 6

# Conclusion Générale

## 6.1 Bilan des travaux réalisés

Les travaux présentés dans ce manuscrit se sont principalement concentrés sur l'amélioration des stratégies de segmentation d'images de microscopie cellulaire.
Le chapitre 2 recense les grandes lignes de la littérature sur les stratégies de segmentation ne s'appuyant pas sur de la supervision. Bien que ces méthodes soient génériques de part leur nature, elles souffrent également de performances limitées. En effet, résoudre une tâche de segmentation à partir de l'analyse d'une seule image d'a priori est un problème difficile.

Avec l'avènement du *Big Data*, les stratégies reposant sur l'exploitation des informations comprises dans un vaste ensemble d'images ont pris l'ascendant. Parmi elles, c'est l'apprentissage profond dont le fonctionnement est détaillé dans le chapitre 3 qui bénéficie des meilleures performances de l'état de l'art. Les bases ainsi que les clés de l'optimisation des réseaux de neurones ont également été détaillées dans ce chapitre. Toutefois, pour pouvoir tirer le meilleur profit de ces méthodes, un point important et qui a retenu notre attention dans ces travaux, est celui de la taille du corpus d'images annotées nécessaire afin d'atteindre des résultats pleinement satisfaisants. Si celle-ci est trop réduite, il n'existe pas de solution miracle pour y remédier autre qu'annoter manuellement de nouvelles données ou bien de simuler artificiellement les images manquantes (par une augmentation de données par exemple). Nos travaux se sont alors attachés à dépasser cette limitation et contribuer à rendre les méthodes d'apprentissage profond utilisables avec une quantité de supervision très réduite.

Le premier axe de travail, longuement détaillé dans le chapitre 4 consiste à améliorer l'efficacité des mécanismes de l'augmentation de données. Après avoir défini ce qu'est une donnée contre-productive, nos travaux se sont principalement concentrés sur la réalisation d'une méthode en ligne permettant de directement les écarter au cours de l'apprentissage. L'enjeu qui émane de cette série de travaux est de pouvoir optimiser des augmentations de données massives contenant également de nombreuses images peu efficaces. La création d'une augmentation de données générique à grande échelle serait alors possible, car les désagréments liés aux données contre-productives seraient traités directement au sein de la boucle d'apprentissage.
La solution originale proposée dans cette thèse et nommée SODA pour *Self*

*Organizing Data Augmentation* permet d'atteindre un tel objectif à coût calculatoire très modeste puisque que la solution utilise les gradients déjà calculés lors de la descente de gradient du réseau. Les augmentations fournissant des gradients moyens alignés avec ceux des données originelles voient leur probabilité d'être sélectionnées augmenter, tandis que les autres voient leur probabilité baisser. La mise en œuvre de cette politique de sélection automatique de mécanismes d'apprentissage est orchestrée via l'algorithme HEDGE d'apprentissage en ligne. Cette approche a le bon goût d'épargner à l'utilisateur la sélection des mécanismes d'augmentation qui seront réellement profitables. Ces travaux furent également l'objet d'une publication acceptée à la conférence internationale sur l'acoustique, le langage et le traitement du signal (ICASSP 2022) [26].

Le second axe associé au chapitre 5 s'est principalement concentré sur l'amélioration des performances d'un réseau U-net quand le nombre d'images annotées est si faible que l'augmentation de données à elle seule ne peut plus constituer une réponse fonctionnelle. Pour ce faire, un nouveau protocole s'inscrivant dans la lignée du *self-learning* est proposé. En plus du réseau U-net que nous souhaitons entraîner, un autoencodeur est créé à l'aide de deux autres réseaux tiers, et permet l'ajout d'un corpus d'images non annotées dans l'apprentissage. Le gain de performance est significatif, notamment car la forme des cellules est exploitée dans l'apprentissage de l'un des réseaux, dit "professeur". Si cette approche emprunte une philosophie déjà présente dans les méthodes de distillation, la façon dont le réseau professeur va accompagner l'apprentissage du réseau cible (dit étudiant) est véritablement originale par rapport à la littérature dédiée.
L'une des finalités de ces travaux consiste à faire du *few shot learning* pour être capable de segmenter des jeux de données entiers à partir d'un nombre d'images annotées fournies par un expert typiquement inférieur à 10. Dans nos expériences, ce nombre a pu être abaissé à 3 images annotées en couplant la stratégie proposée avec de l'augmentation de données qui semble offrir une supervision d'une nature différente et complémentaire à notre approche. Les gains en performance de segmentation sont très substantiels comparativement à un réseau U-net qui devrait apprendre sans être guidé par le réseau professeur. Pour arriver à ce résultat, certaines limites ont été posées, notamment sur le fait que l'image d'entrée ne doit contenir qu'une seule instance d'objet à segmenter. Des pistes de réflexion sur ce point vont être évoquées dans la section suivante.
Enfin, intégrer ces travaux dans des perspectives dépassant le cadre du biomédical est également un enjeu majeur bien que cela demanderait d'adapter le réseau correcteur en conséquence. Les travaux actuels font état d'une soumission auprès du journal **Computers in Biology and Medicine** et sont au moment de la rédaction de ce manuscrit, dans une étape de révision.

## 6.2 Perspectives futures

Les deux sujets de recherche présentés dans ce manuscrit sont indépendants (bien que complémentaires) et font l'objet de nombreux travaux supplémentaires potentiels à effectuer. Comme tous travaux de recherche, la quête d'une approche toujours meilleure est presque sans fin. Nous listons toutefois ci-après les évolutions possibles qui nous semblent les plus importantes et les plus cruciales à mettre en œuvre.

### 6.2.1 Évolutions possibles de SODA

Les axes d'amélioration pour SODA sont divers et vont principalement se concentrer sur l'amélioration des performances de l'algorithme, ainsi que la diversification de son cadre d'utilisation. Une partie des travaux restants est listée ci-dessous :

— L'algorithme SODA est parfois confronté à des problèmes de stabilité en début d'apprentissage. L'utilisation de HEDGE permet entre autres d'avoir un signal pour chaque mécanisme d'augmentation ce qui lutte contre ce problème, au sens où tous les mécanismes sont testés pendant un certain temps mais ne constitue pas une réponse suffisante. Les informations issues des premières *epochs* sont généralement plus volatiles car elles proviennent de mises à jour violentes du vecteur de paramètres, le réseau étant très peu performant au départ. Entre guillemets, le dilemme exploitation/exploration apparaît trop vite, c'est à dire qu'on pourrait écarter un mécanisme d'augmentation fournissant des résultats décevant en pensant cela légitime car calculé à partir d'un nombre conséquent de données. Or, il se peut que ce retour sur l'action d'un mécanisme ne soit pas fiable en début d'apprentissage. Un paramètre permettant de retarder le déclenchement de HEDGE a été introduit, mais l'automatisation de ce déclenchement serait souhaitable. Un critère basé sur la variance des directions de gradients issus des données non-augmentées pourrait jouer le rôle d'arbitre.

— Toujours dans l'optique d'améliorer les performances, il peut sembler intéressant de changer l'algorithme d'optimisation utilisé. En retenant le signal des gradients non pas à la fin d'une *epoch* , mais après chaque *minibatch*, il serait possible d'utiliser des algorithmes à base de bandits ainsi que toute la littérature associée.

— Une autre piste pour améliorer la convergence de l'algorithme consiste à modifier la fonction d'erreur pour qu'elle considère la répartition des classes. Un jeu de données assez spécifique comme le DF-C2DL-MuSC proposé par l'ISBI, met en difficulté SODA par un ajout de bruit très élevé sur les gradients obtenus au début de l'apprentissage. Une hypothèse est que dans ce jeu de données, la cellule est sous-représentée dans l'image, de l'ordre de 1 pour 1000. Ceci peut ajouter un effet de bruitage non négligeable qui pourrait être réduit en modifiant correctement la fonction de perte associée.

— Dans certains cas, il pourrait être intéressant de travailler avec un budget variable de données augmentées afin de laisser plus de place

aux données originales quand celles augmentées n'offrent pas ou plus d'intérêt. On peut alors imaginer une variante de l'algorithme qui calcule un signal de récompense pour les données non augmentées (ce qui revient à rajouter une nouvelle action), pour ensuite définir un budget à allouer pour la partie augmentée. Des tests ont déjà été effectuées et montrent que les gradients obtenus pour les données réelles sont généralement plus élevés que ceux des données augmentées. La stratégie doit donc aussi être capable de pondérer efficacement ces valeurs.

— Une dernière stratégie, bien plus simple à mettre en œuvre consiste à utiliser un ensemble de validation pour améliorer les performances de l'attribution de données. Le souci étant que cela s'accompagnera d'une baisse du nombre de données dans l'ensemble d'entrainement. Une compétition naturelle va alors s'installer entre l'amélioration des performances grâce à une augmentation de données mieux contrôlée, et la baisse de performances entrainée par une réduction du nombre de données d'entrainement.

— Afin de diversifier le cadre d'utilisation de SODA, il faudrait être capable de l'utiliser sur de nombreuses tâches différentes (classification simple, régression). De plus, ces modifications doivent aussi toucher les réseaux de neurones utilisés. Actuellement, rien n'indique que SODA se doit de marcher exclusivement pour des tâches de segmentation. À noter que les informations intrinsèquement liées à la caractérisation des cellules n'ont pas été utilisées. Il semble raisonnable de croire qu'utiliser SODA sur un panorama plus étendu de l'apprentissage profond ne nécessite pas de modifications majeures de l'approche.

### 6.2.2 Futurs travaux sur l'approche auto-supervisée

Le protocole décrit lors du chapitre 5 fait état de performances remarquables. Cependant, c'est son cadre d'utilisation qui fait malheureusement défaut. Par exemple, la segmentation de plusieurs instances ainsi que les images présentant de forts contrastes (au niveau de fond) sont deux des limites majeures de l'approche. Les différents travaux en cours, et à réaliser vont avoir pour objectif de pallier ces faiblesses. Une partie des pistes permettant d'étendre le domaine applicatif du protocole sont listées ci-dessous :

— L'utilisation de la stratégie sur des données multicellulaires est la priorité car elle répond à une forte demande dans le cadre biomédical. Une piste consiste à utiliser un réseau dit Mask R-CNN qui peut partitionner les différentes instances de l'image avant d'utiliser le réseau professeur. L'ensemble des images de masque de segmentation fourni par Mask R-CNN peut alors simplement être traité comme un *batch* par le réseau professeur.

— Une forte variabilité sur le fond des images, ou sur l'objet à segmenter peut également perturber l'utilisation du réseau inverse. Effectuer différents tests pour mettre en lumière les faiblesses de ce réseau semble

primordial. Une idée consiste à utiliser des variables auxiliaires capables d'encoder le style de l'image à reconstruire et les divers facteurs de variabilité. Si un encodeur arrive à faire cela, le réseau inverse aura alors suffisamment d'informations pour reconstruire l'image d'entrée à partir du masque de segmentation.

— L'utilisation d'un *Curriculum Learning* au sein de notre stratégie revêt une importance capitale. La définition d'une donnée dite simple est généralement à définir par l'utilisateur. Nous utilisons entre autres la taille de l'objet à segmenter pour définir notre ordre de priorité. Ce calcul de taille basé sur les données non annotées est fortement dépendant du fond de ces mêmes images, ce qui limite nos cas d'utilisation. De ce fait, trouver une heuristique plus robuste à la nature des données non annotées semble particulièrement intéressant.

— Le couplage de l'augmentation de données avec notre protocole est un enjeu majeur. Pour l'instant, augmenter la partie non supervisée de notre stratégie ne semble pas porter ses fruits. L'utilisation d'une fonction de perte caractéristique d'un autoencodeur rend notre approche très sensible aux données contre-productives. Cela est encore plus vrai quand une grande partie des poids inclus dans nos réseaux sont gelés, ce qui bride les possibilités du modèle. Coupler les deux contributions de ce manuscrit permettrait de résoudre ce problème. Une augmentation de données massive pourrait être utilisée sur la partie non annotée, et les données contre-productives seraient automatiquement écartées.

— Il peut également sembler intéressant de changer les réseaux utilisés de notre approche. Tous les réseaux populaires pour la résolution des tâches de segmentation présentent-ils les mêmes défauts que U-net ? Si ce n'est pas le cas, les défauts de ces nouveaux réseaux peuvent-ils être résolus de manière simple (avec un professeur générique) ? Un idéal pourrait être de garder notre professeur générique et d'uniquement changer les deux autres réseaux de la stratégie.

— Dans le même esprit que la remarque plus haut, il convient de vérifier si les faiblesses de U-net concernant la segmentation des cellules se retrouvent également pour la segmentation d'objets plus complexes. Dans ce cadre, est-il possible de généraliser notre approche pour des domaines plus étendus que le biomédical ?

— Concernant la partie performance, il semble intéressant de vérifier la validité de notre stratégie pour un jeu de données de grande taille. Pour le jeu de données Capsule, un indice de Jaccard (JI) de 0.94 est obtenu en moyenne pour un apprentissage sur la presque totalité des données (263 exactement, le reste est affecté à la validation-test). Notre stratégie auto-supervisée peut-elle dépasser cette limite pour un nombre de données identiques ? Pouvons-nous espérer améliorer les performances maximales avec notre protocole ? Dans le cas contraire, il peut être intéressant de trouver la borne minimale de données annotées permettant d'obtenir 0.94 de JI via l'utilisation de notre stratégie.

# Bibliographie


[1] Rolf ADAMS et Leanne BISCHOF. « Seeded region growing ». In : *IEEE Transactions on pattern analysis and machine intelligence* 16.6 (1994), p. 641-647.

[2] Vijay BADRINARAYANAN, Alex KENDALL et Roberto CIPOLLA. « Segnet : A deep convolutional encoder-decoder architecture for image segmentation ». In : *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), p. 2481-2495.

[3] Rodney J BAXTER. *Exactly solved models in statistical mechanics*. Elsevier, 2016.

[4] Yoshua BENGIO, Nicholas LÉONARD et Aaron COURVILLE. « Estimating or propagating gradients through stochastic neurons for conditional computation ». In : *arXiv preprint arXiv :1308.3432* (2013).

[5] Yoshua BENGIO et al. « Curriculum learning ». In : *Proceedings of the 26th annual international conference on machine learning*. 2009, p. 41-48.

[6] Yoshua BENGIO et al. « Generalized denoising auto-encoders as generative models ». In : *Advances in neural information processing systems* 26 (2013).

[7] Alexandre BENOIT et al. « Using human visual system modeling for bio-inspired low level image processing ». In : *Computer vision and Image understanding* 114.7 (2010), p. 758-773.

[8] Luca BERTELLI et al. « Kernelized structural SVM learning for supervised object segmentation ». In : *CVPR 2011*. IEEE. 2011, p. 2153-2160.

[9] Marc BERTHOD et al. « Bayesian image classification using Markov random fields ». In : *Image and vision computing* 14.4 (1996), p. 285-295.

[10] Serge BEUCHER. « The watershed transformation applied to image segmentation ». In : *Scanning Microscopy* 1992.6 (1992), p. 28.

[11] Serge BEUCHER. « Use of watersheds in contour detection ». In : *Proceedings of the International Workshop on Image Processing*. CCETT. 1979.

[12] Serge BEUCHER et Fernand MEYER. « The morphological approach to segmentation : the watershed transformation ». In : *Mathematical morphology in image processing* 34 (1993), p. 433-481.

[13] Christopher M BISHOP et Nasser M NASRABADI. *Pattern recognition and machine learning*. T. 4. 4. Springer, 2006.

[14] Hans-Dieter BLOCK. « The perceptron : A model for brain functioning. i ». In : *Reviews of Modern Physics* 34.1 (1962), p. 123.

[15] Jane BROMLEY et al. « Signature verification using a" siamese" time delay neural network ». In : *Advances in neural information processing systems* 6 (1993).

[16] Laurent BUSIN, Nicolas VANDENBROUCKE et Ludovic MACAIRE. « Color spaces and image segmentation ». In : *Advances in imaging and electron physics* 151.1 (2008), p. 1.

[17] AP CALDERÓN. « Intermediate spaces and interpolation ». In : *Studia Mathematica* 1 (1963), p. 31-34.

[18] John CANNY. « A computational approach to edge detection ». In : *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), p. 679-698.

[19] M Emre CELEBI, Hassan A KINGRAVI et Patricio A VELA. « A comparative study of efficient initialization methods for the k-means clustering algorithm ». In : *Expert systems with applications* 40.1 (2013), p. 200-210.

[20] Nicolo CESA-BIANCHI et Gábor LUGOSI. *Prediction, learning, and games*. Cambridge university press, 2006.

[21] Tony CHAN et Luminita VESE. « An active contour model without edges ». In : *International conference on scale-space theories in computer vision*. Springer. 1999, p. 141-151.

[22] Patrick C CHEN et Theodosios PAVLIDIS. « Segmentation by texture using a co-occurrence matrix and a split-and-merge algorithm ». In : *Computer graphics and image processing* 10.2 (1979), p. 172-182.

[23] Yizong CHENG. « Mean shift, mode seeking, and clustering ». In : *IEEE transactions on pattern analysis and machine intelligence* 17.8 (1995), p. 790-799.

[24] Dorin COMANICIU et Peter MEER. « Mean shift : A robust approach toward feature space analysis ». In : *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), p. 603-619.

[25] Ekin D CUBUK et al. « Autoaugment : Learning augmentation strategies from data ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, p. 113-123.

[26] Arnaud DELERUYELLE, John KLEIN et Cristian VERSARI. « SODA : Self-Organizing Data Augmentation in Deep Neural Networks Application to Biomedical Image Segmentation Tasks ». In : *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, p. 3204-3208.

[27] AP DEMPSTER, NM LAIRD et DB RUBIN. « Maximum likelihood from incomplete data via the EMalgorithm ». In : *Journal of the Royal statistical Society* 39.1 (1977), p. 1-38.

[28] Jacob DEVLIN et al. « Bert : Pre-training of deep bidirectional transformers for language understanding ». In : *arXiv preprint arXiv :1810.04805* (2018).

[29] Michal DROZDZAL et al. « The importance of skip connections in biomedical image segmentation ». In : *Deep learning and data labeling for medical applications*. Springer, 2016, p. 179-187.

[30] John DUCHI, Elad HAZAN et Yoram SINGER. « Adaptive subgradient methods for online learning and stochastic optimization. » In : *Journal of machine learning research* 12.7 (2011).

[31] Dennis DUNN et William E HIGGINS. « Optimal Gabor filters for texture segmentation ». In : *IEEE Transactions on image processing* 4.7 (1995), p. 947-964.

[32] Jeffrey L ELMAN. « Learning and development in neural networks : The importance of starting small ». In : *Cognition* 48.1 (1993), p. 71-99.

[33] Pedro F FELZENSZWALB et Daniel P HUTTENLOCHER. « Efficient graph-based image segmentation ». In : *International journal of computer vision* 59.2 (2004), p. 167-181.

[34] William FINNOFF, Ferdinand HERGERT et Hans Georg ZIMMERMANN. « Improving model selection by nonconvergent methods ». In : *Neural Networks* 6.6 (1993), p. 771-783.

[35] Yoav FREUND et Robert E SCHAPIRE. « A decision-theoretic generalization of on-line learning and an application to boosting ». In : *Journal of computer and system sciences* 55.1 (1997), p. 119-139.

[36] Maayan FRID-ADAR et al. « Synthetic data augmentation using GAN for improved liver lesion classification ». In : *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*. IEEE. 2018, p. 289-293.

[37] Keinosuke FUKUNAGA et Larry HOSTETLER. « The estimation of the gradient of a density function, with applications in pattern recognition ». In : *IEEE Transactions on information theory* 21.1 (1975), p. 32-40.

[38] Valentin GABEUR et al. « Multi-modal transformer for video retrieval ». In : *European Conference on Computer Vision*. Springer. 2020, p. 214-229.

[39] Stuart GEMAN et Donald GEMAN. « Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images ». In : *IEEE Transactions on pattern analysis and machine intelligence* 6 (1984), p. 721-741.

[40] Xavier GLOROT et Yoshua BENGIO. « Understanding the difficulty of training deep feedforward neural networks ». In : *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop et Conference Proceedings. 2010, p. 249-256.

[41] Mircea GURBINĂ, Mihaela LASCU et Dan LASCU. « Tumor detection and classification of MRI brain image using different wavelet transforms and support vector machines ». In : *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*. IEEE. 2019, p. 505-508.

[42] Robert M HARALICK, Karthikeyan SHANMUGAM et Its' Hak DINSTEIN. « Textural features for image classification ». In : *IEEE Transactions on systems, man, and cybernetics* 6 (1973), p. 610-621.

[43] Ryuichiro HATAYA et al. « Faster autoaugment : Learning augmentation strategies using backpropagation ». In : *European Conference on Computer Vision*. Springer. 2020, p. 1-16.

[44] Kaiming HE et al. « Deep residual learning for image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 770-778.

[45] Kaiming HE et al. « Mask r-cnn ». In : *Proceedings of the IEEE international conference on computer vision*. 2017, p. 2961-2969.

[46] Geoffrey HINTON, Oriol VINYALS, Jeff DEAN et al. « Distilling the knowledge in a neural network ». In : *arXiv preprint arXiv :1503.02531* 2.7 (2015).

[47] Sepp HOCHREITER et Jürgen SCHMIDHUBER. « Long short-term memory ». In : *Neural computation* 9.8 (1997), p. 1735-1780.

[48] Sepp HOCHREITER et Jürgen SCHMIDHUBER. « Long short-term memory ». In : *Neural computation* 9.8 (1997), p. 1735-1780.

[49] Arthur E HOERL et Robert W KENNARD. « Ridge regression : Biased estimation for nonorthogonal problems ». In : *Technometrics* 12.1 (1970), p. 55-67.

[50] SA HOJJATOLESLAMI et Josef KITTLER. « Region growing : a new approach ». In : *IEEE Transactions on Image processing* 7.7 (1998), p. 1079-1084.

[51] Steven L HOROWITZ. « Picture segmentation by a directed split-and-merge procedure ». In : *IJCPR*. 1974, p. 424-433.

[52] Gao HUANG et al. « Densely connected convolutional networks ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, p. 4700-4708.

[53] Thomas HUANG, GJTGY YANG et Greory TANG. « A fast two-dimensional median filtering algorithm ». In : *IEEE transactions on acoustics, speech, and signal processing* 27.1 (1979), p. 13-18.

[54] Hiroshi INOUE. « Data augmentation by pairing samples for images classification ». In : *arXiv preprint arXiv :1801.02929* (2018).

[55] Sergey IOFFE et Christian SZEGEDY. « Batch normalization : Accelerating deep network training by reducing internal covariate shift ». In : *International conference on machine learning*. PMLR. 2015, p. 448-456.

[56] Philip TG JACKSON et al. « Style augmentation : data augmentation via style randomization. » In : *CVPR workshops*. T. 6. 2019, p. 10-11.

[57] Stephane JAFFARD. *Wavelet techniques in multifractal analysis*. Rapp. tech. PARIS UNIV (FRANCE), 2004.

[58] Eric JANG, Shixiang GU et Ben POOLE. « Categorical reparameterization with gumbel-softmax ». In : *arXiv preprint arXiv :1611.01144* (2016).

[59] T JAYALAKSHMI et A SANTHAKUMARAN. « Statistical normalization and back propagation for classification ». In : *International Journal of Computer Theory and Engineering* 3.1 (2011), p. 1793-8201.

[60] H JIAN. « Comparative performance evaluation of segmentation methods based on region growing and division ». In : *System and Computers in Japan* (1993).

[61] John JUMPER et al. « Highly accurate protein structure prediction with AlphaFold ». In : *Nature* 596.7873 (2021), p. 583-589.

[62] Hassana Grema KAGANAMI et Zou BEIJI. « Region-based segmentation versus edge detection ». In : *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE. 2009, p. 1217-1221.

[63] Nick KANOPOULOS, Nagesh VASANTHAVADA et Robert L BAKER. « Design of an image edge detection filter using the Sobel operator ». In : *IEEE Journal of solid-state circuits* 23.2 (1988), p. 358-367.

[64] Michael KASS, Andrew WITKIN et Demetri TERZOPOULOS. « Snakes : Active contour models ». In : *International journal of computer vision* 1.4 (1988), p. 321-331.

[65] Shehroz S KHAN et Amir AHMAD. « Cluster center initialization algorithm for K-means clustering ». In : *Pattern recognition letters* 25.11 (2004), p. 1293-1302.

[66] Diederik P KINGMA et Jimmy BA. « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980* (2014).

[67] Diederik P KINGMA et Max WELLING. « Auto-encoding variational bayes ». In : *arXiv preprint arXiv :1312.6114* (2013).

[68] Gregory KOCH, Richard ZEMEL, Ruslan SALAKHUTDINOV et al. « Siamese neural networks for one-shot image recognition ». In : *ICML deep learning workshop*. T. 2. Lille. 2015, p. 0.

[69] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON. « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems* 25 (2012).

[70] Samuli LAINE et Timo AILA. « Temporal ensembling for semi-supervised learning ». In : *arXiv preprint arXiv :1610.02242* (2016).

[71] Brenden LAKE et al. « One shot learning of simple visual concepts ». In : *Proceedings of the annual meeting of the cognitive science society*. T. 33. 33. 2011.

[72] Yann LECUN et al. « Backpropagation applied to handwritten zip code recognition ». In : *Neural computation* 1.4 (1989), p. 541-551.

[73] Yann LECUN et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.

[74] Yann LECUN et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.

[75] Jong-Sen LEE. « Digital image smoothing and the sigma filter ». In : *Computer vision, graphics, and image processing* 24.2 (1983), p. 255-269.

[76] Yonggang LI et al. « Differentiable automatic data augmentation ». In : *European Conference on Computer Vision*. Springer. 2020, p. 580-595.

[77] Sungbin LIM et al. « Fast autoaugment ». In : *Advances in Neural Information Processing Systems* 32 (2019).

[78] Claudia LINDNER et al. « Fully automatic segmentation of the proximal femur using random forest regression voting ». In : *IEEE transactions on medical imaging* 32.8 (2013), p. 1462-1472.

[79] Aoming LIU et al. « Direct differentiable augmentation search ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, p. 12219-12228.

[80] Stuart LLOYD. « Least squares quantization in PCM ». In : *IEEE transactions on information theory* 28.2 (1982), p. 129-137.

[81] Jonathan LONG, Evan SHELHAMER et Trevor DARRELL. « Fully convolutional networks for semantic segmentation ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, p. 3431-3440.

[82] Pablo MARQUEZ-NEILA, Luis BAUMELA et Luis ALVAREZ. « A morphological approach to curvature-based evolution of curves and surfaces ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.1 (2013), p. 2-17.

[83] Martin MAŠKA et al. « A benchmark for comparison of cell tracking algorithms ». In : *Bioinformatics* 30.11 (2014), p. 1609-1617.

[84] Geoffrey J MCLACHLAN et Thriyambakam KRISHNAN. *The EM algorithm and extensions*. John Wiley & Sons, 2007.

[85] Andrew MEHNERT et Paul JACKWAY. « An improved seeded region growing algorithm ». In : *Pattern Recognition Letters* 18.10 (1997), p. 1065-1071.

[86] Fernand MEYER et Serge BEUCHER. « Morphological segmentation ». In : *Journal of visual communication and image representation* 1.1 (1990), p. 21-46.

[87] Agnieszka MIKOŁAJCZYK et Michał GROCHOWSKI. « Data augmentation for improving deep learning in image classification problem ». In : *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE. 2018, p. 117-122.

[88] Erik G MILLER, Nicholas E MATSAKIS et Paul A VIOLA. « Learning from one example through shared densities on transforms ». In : *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*. T. 1. IEEE. 2000, p. 464-471.

[89] P Read MONTAGUE, Peter DAYAN et Terrence J SEJNOWSKI. « A framework for mesencephalic dopamine systems based on predictive Hebbian learning ». In : *Journal of neuroscience* 16.5 (1996), p. 1936-1947.

[90] John L MUERLE. « Experimental evaluation of techniques for automatic segmentation of objects in a complex scene ». In : *Pictorial pattern recognition* (1968), p. 3-13.

[91] David Bryant MUMFORD et Jayant SHAH. « Optimal approximations by piecewise smooth functions and associated variational problems ». In : *Communications on pure and applied mathematics* (1989).

[92] Timo OJALA, Matti PIETIKAINEN et Topi MAENPAA. « Multiresolution gray-scale and rotation invariant texture classification with local binary patterns ». In : *IEEE Transactions on pattern analysis and machine intelligence* 24.7 (2002), p. 971-987.

[93] Sinno Jialin PAN et Qiang YANG. « A survey on transfer learning ». In : *IEEE Transactions on knowledge and data engineering* 22.10 (2009), p. 1345-1359.

[94] Jay PATRAVALI, Shubham JAIN et Sasank CHILAMKURTHY. « 2D-3D fully convolutional neural networks for cardiac MR segmentation ». In : *International Workshop on Statistical Atlases and Computational Models of the Heart*. Springer. 2017, p. 130-139.

[95] José M PENA, Jose Antonio LOZANO et Pedro LARRANAGA. « An empirical comparison of four initialization methods for the k-means algorithm ». In : *Pattern recognition letters* 20.10 (1999), p. 1027-1040.

[96] Daifeng PENG, Yongjun ZHANG et Haiyan GUAN. « End-to-end change detection for high resolution satellite images using improved UNet++ ». In : *Remote Sensing* 11.11 (2019), p. 1382.

[97] Luis PEREZ et Jason WANG. « The effectiveness of data augmentation in image classification using deep learning ». In : *arXiv preprint arXiv :1712.04621* (2017).

[98] Hieu PHAM et al. « Meta pseudo labels ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, p. 11557-11568.

[99] Boris T POLYAK. « Some methods of speeding up the convergence of iteration methods ». In : *Ussr computational mathematics and mathematical physics* 4.5 (1964), p. 1-17.

[100] Sachin RAVI et Hugo LAROCHELLE. « Optimization as a model for few-shot learning ». In : (2016).

[101] Mengye REN et al. « Meta-learning for semi-supervised few-shot classification ». In : *arXiv preprint arXiv :1803.00676* (2018).

[102] Xiaofeng REN et Jitendra MALIK. « Learning a classification model for segmentation ». In : *Computer Vision, IEEE International Conference on*. T. 2. IEEE Computer Society. 2003, p. 10-10.

[103] Jean-Francois RIVEST, Pierre SOILLE et Serge BEUCHER. « Morphological gradients ». In : *Journal of Electronic Imaging* 2.4 (1993), p. 326-336.

[104] Cédric ROMMEL et al. « CADDA : Class-wise automatic differentiable data augmentation for EEG signals ». In : *arXiv preprint arXiv :2106.13695* (2021).

[105] Olaf RONNEBERGER, Philipp FISCHER et Thomas BROX. « U-net : Convolutional networks for biomedical image segmentation ». In : *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, p. 234-241.

[106] David E RUMELHART, Geoffrey E HINTON et Ronald J WILLIAMS. « Learning representations by back-propagating errors ». In : *nature* 323.6088 (1986), p. 533-536.

[107] Anne-Virginie SALSAC et al. *Capsule : a dataset for the segmentation of a transparent and deformable capsule.* Online at : https://github.com/ArnaudDeleruyelle/Dataset_Capsule. 2021. URL : https://github.com/ArnaudDeleruyelle/Dataset_Capsule.

[108] John SCHULMAN et al. « Proximal policy optimization algorithms ». In : *arXiv preprint arXiv :1707.06347* (2017).

[109] Jianbo SHI et Jitendra MALIK. « Normalized cuts and image segmentation ». In : *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), p. 888-905.

[110] Hidetoshi SHIMODAIRA. « Improving predictive inference under covariate shift by weighting the log-likelihood function ». In : *Journal of statistical planning and inference* 90.2 (2000), p. 227-244.

[111] Connor SHORTEN et Taghi M KHOSHGOFTAAR. « A survey on image data augmentation for deep learning ». In : *Journal of big data* 6.1 (2019), p. 1-48.

[112] Connor SHORTEN et Taghi M KHOSHGOFTAAR. « A survey on image data augmentation for deep learning ». In : *Journal of big data* 6.1 (2019), p. 1-48.

[113] Nahian SIDDIQUE et al. « U-net and its variants for medical image segmentation : A review of theory and applications ». In : *Ieee Access* 9 (2021), p. 82031-82057.

[114] Patrice Y SIMARD, David STEINKRAUS, John C PLATT et al. « Best practices for convolutional neural networks applied to visual document analysis. » In : *Icdar*. T. 3. 2003. Edinburgh. 2003.

[115] Karen SIMONYAN et Andrew ZISSERMAN. « Very deep convolutional networks for large-scale image recognition ». In : *arXiv preprint arXiv :1409.1556* (2014).

[116] Jake SNELL, Kevin SWERSKY et Richard ZEMEL. « Prototypical networks for few-shot learning ». In : *Advances in neural information processing systems* 30 (2017).

[117] Nitish SRIVASTAVA et al. « Dropout : a simple way to prevent neural networks from overfitting ». In : *The journal of machine learning research* 15.1 (2014), p. 1929-1958.

[118] Joes STAAL et al. « Ridge-based vessel segmentation in color images of the retina ». In : *IEEE transactions on medical imaging* 23.4 (2004), p. 501-509.

[119] Thomas G STOCKHAM JR. « High-speed convolution and correlation ». In : *Proceedings of the April 26-28, 1966, Spring joint computer conference*. 1966, p. 229-233.

[120] Michael J SWAIN et Dana H BALLARD. « Color indexing ». In : *International journal of computer vision* 7.1 (1991), p. 11-32.

[121] Antti TARVAINEN et Harri VALPOLA. « Mean teachers are better role models : Weight-averaged consistency targets improve semi-supervised deep learning results ». In : *Advances in neural information processing systems* 30 (2017).

[122] Robert TIBSHIRANI. « Regression shrinkage and selection via the lasso ». In : *Journal of the Royal Statistical Society : Series B (Methodological)* 58.1 (1996), p. 267-288.

[123] Robert J TIBSHIRANI et Bradley EFRON. « An introduction to the bootstrap ». In : *Monographs on statistics and applied probability* 57 (1993), p. 1-436.

[124] Tijmen TIELEMAN, Geoffrey HINTON et al. « Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude ». In : *COURSERA : Neural networks for machine learning* 4.2 (2012), p. 26-31.

[125] Michael TSCHANNEN, Olivier BACHEM et Mario LUCIC. « Recent advances in autoencoder-based representation learning ». In : *arXiv preprint arXiv :1812.05069* (2018).

[126] Jessica VAN BRUMMELEN et al. « Autonomous vehicle perception : The technology of today and tomorrow ». In : *Transportation research part C : emerging technologies* 89 (2018), p. 384-406.

[127] Andrea VEDALDI et Stefano SOATTO. « Quick shift and kernel methods for mode seeking ». In : *European conference on computer vision*. Springer. 2008, p. 705-718.

[128] Alex VIGNERON et Jean MARTINET. « A critical survey of STDP in Spiking Neural Networks for Pattern Recognition ». In : *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, p. 1-9.

[129] Pascal VINCENT et al. « Extracting and composing robust features with denoising autoencoders ». In : *Proceedings of the 25th international conference on Machine learning*. 2008, p. 1096-1103.

[130] Oriol VINYALS et al. « Matching networks for one shot learning ». In : *Advances in neural information processing systems* 29 (2016).

[131] Yuan WANG et al. « Fully Self-Supervised Learning for Semantic Segmentation ». In : *arXiv preprint arXiv :2202.11981* (2022).

[132] Bernard WIDROW et Marcian E HOFF. *Adaptive switching circuits*. Rapp. tech. Stanford Univ Ca Stanford Electronics Labs, 1960.

[133] D Randall WILSON et Tony R MARTINEZ. « The general inefficiency of batch training for gradient descent learning ». In : *Neural networks* 16.10 (2003), p. 1429-1451.

[134] Ren WU et al. « Deep image : Scaling up image recognition ». In : *arXiv preprint arXiv :1501.02876* 7.8 (2015).

[135] Zhenyu WU et Richard LEAHY. « An optimal graph theoretic approach to data clustering : Theory and its application to image segmentation ». In : *IEEE transactions on pattern analysis and machine intelligence* 15.11 (1993), p. 1101-1113.

[136] Qizhe XIE et al. « Self-training with noisy student improves imagenet classification ». In : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, p. 10687-10698.

[137] Bing XU et al. « Empirical evaluation of rectified activations in convolutional network ». In : *arXiv preprint arXiv :1505.00853* (2015).

[138] Charles T ZAHN. « Graph-theoretical methods for detecting and describing gestalt clusters ». In : *IEEE Transactions on computers* 100.1 (1971), p. 68-86.

[139] Yu ZHENG et al. « Deep autoaugment ». In : *arXiv preprint arXiv :2203.06172* (2022).

[140] Zhun ZHONG et al. « Random erasing data augmentation ». In : *Proceedings of the AAAI conference on artificial intelligence*. T. 34. 07. 2020, p. 13001-13008.

[141] Zongwei ZHOU et al. « Unet++ : A nested u-net architecture for medical image segmentation ». In : *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2018, p. 3-11.

[142] Xinyue ZHU et al. « Emotion classification with data augmentation using generative adversarial networks ». In : *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2018, p. 349-360.

[143] Steven W ZUCKER. « Region growing : Childhood and adolescence ». In : *Computer graphics and image processing* 5.3 (1976), p. 382-399.