



Rapport de projet annuel
M2 Informatique parcours Image et Données Multimédia
Année scolaire 2022-2023

Segmentation automatique de lambeaux de tissus mous par intelligence artificielle pour la radiothérapie post-opératoire des patients atteints d'un cancer de la tête et du cou

UNIVERSITÉ DE CAEN, NORMANDIE
UFR DES SCIENCES

Projet réalisé par :
Guillaume LETELLIER
Raphaëlle LEMAIRE

Durée :
Octobre 2022 à Janvier 2023

Projet encadré par :
Youssef CHAHIR
Alexis LECHERVY
Juliette THARIAT

Table des matières

Table des figures	1
Liste des tableaux	1
Introduction	2
1 Le projet	3
1.1 La mission	3
1.1.1 Objectif	3
1.1.2 Idée de réalisation	4
1.2 Intérêt pour les collaborateurs	4
1.3 Organisation du travail	5
1.3.1 Outils utilisés	5
1.3.2 Temps accordé au projet et organisation	6
2 Travail effectué	8
2.1 Les données	8
2.1.1 Création	8
2.1.2 État de l'art sur les bases de données	9
2.1.3 Traitement des données	9
2.1.4 Conversion en NIfTI	11
2.1.5 <i>Preprocessing</i> pour le modèle	12
2.2 Modèles et outils pour leur entraînement	16
2.2.1 État de l'art	16
2.2.2 Sélection et tests des différentes architectures	19
2.3 Sélection de l'architecture finale	22
2.3.1 Compréhension de l'architecture du réseau	23
2.3.2 <i>Finetuning</i> du modèle	25
2.3.3 Entraînement de notre modèle	26
2.3.4 Premiers résultats sur nos données	27
2.3.5 Résultats sur les données d'origines	30
2.3.6 Derniers résultats obtenus sur les <i>flaps</i>	32
Conclusion	35
Conclusion générale et apports du projet	35
Ce qu'il reste à effectuer	35
Axes d'amélioration pouvant être envisagés	35
Références	37
Annexes	41
A Recherches sur l'état de l'art	41
B Hyperparamètres du meilleur modèle	41

Table des figures

1	Diagramme de Gantt	7
2	Structuration des fichiers et dossiers	10
3	Schéma de la pipeline de pré-traitement des données pour la validation/test et prédiction	14
4	Schéma de la pipeline de pré-traitement des données pour l'entraînement	15
5	Segmentation d'une photo d'un château par un modèle basé sur SegNet	19
6	Segmentation d'une image médicale par le modèle pré-entraîné sur les images de châteaux	20
7	Segmentation d'instance par Mask-RCNN entraîné sur le jeux de données COCO	20
8	Test du modèle pré-entraîné sur des images du dataset BTCV	21
9	Test du modèle pré-entraîné sur des images du dataset MSD	22
10	Architecture d'un 3D U-Net	23
11	Architecture d'un UNETR	24
12	<i>Slice</i> d'un patient du jeu de validation lors des tout premiers tests	28
13	<i>Slice</i> d'un patient du jeu de validation après correction de la <i>loss</i>	29
14	<i>Slice</i> d'un patient du jeu de validation (version large)	29
15	<i>Slice</i> d'un patient du jeu de validation après corrections (version large)	30
16	<i>Slice</i> du premier patient du jeu de validation	30
17	<i>Slice</i> du deuxième patient du jeu de validation	31
18	<i>Slice</i> ne contenant pas de rate du premier patient du jeu de validation	31
19	Courbes loggées durant l'entraînement (pour les rates)	32
20	<i>Slice</i> d'un patient du jeu de validation	33
21	<i>Slice</i> d'un second patient du jeu de validation (derniers résultats)	33
22	Courbes loggées durant l'entraînement (derniers résultats)	34

Liste des algorithmes

Liste des tableaux

1	Échelle de Hounsfield	3
2	Tableau des hyperparamètre utilisés pour le <i>preprocessing</i> des données	41
3	Tableau des hyperparamètres du modèle	42

Introduction

Étudiants en 2nde année de Master Informatique au sein du parcours Images et données Multimédia (IDM), à l'UFR des Sciences de Caen, nous avons travaillé sur un projet annuel en collaboration avec le centre de cancérologie François Baclesse et le GREYC (Groupe de Recherche en Informatique, Image et Instrumentation de Caen). Ce projet, qui s'est déroulé entre octobre 2022 et janvier 2023, nous a permis d'aider dans les recherches dans le domaine médical et de développer de nouvelles compétences en intelligence artificielle.

Lors du choix pour le projet annuel, plusieurs sujets nous ont été proposés. Quatre d'entre eux portaient sur le domaine médical, un sur la sécurité routière ainsi qu'un dernier sur les réseaux de neurones sur graphes. Deux de ces sujets ont éveillé notre attention. Il s'agissait, pour l'un, de réutiliser un ancien travail pour segmenter les volumes à risques et les volumes cibles, alors que, pour l'autre, nous devions faire une segmentation de greffe, après ablation d'une zone cancéreuse. Suite à un vote pour attribuer les projets à chacun des groupes, nous avons pu obtenir le sujet sur la segmentation de greffe.

Aujourd'hui, différents traitements sont utilisés pour soigner les cancers. Parmi eux, on retrouve l'ablation. Cette technique consiste à retirer la tumeur cancéreuse ainsi qu'une partie des tissus sains autour puis, de procéder à une greffe, souvent graisseuse, à cet endroit. Par la suite, les médecins mettent en place un suivi du patient afin de connaître l'évolution du greffon et, parfois procéder à une nouvelle opération. Jusqu'alors, les médecins devaient détourner le greffon manuellement sur les données d'imagerie par résonance magnétique (IRM). Ainsi, notre objectif, lors du projet annuel, était de faciliter le suivi de cette greffe en permettant sa segmentation automatique sur les résultats des examens via des techniques d'intelligence artificielle.

Les quelques mois passés à travailler sur ce projet nous ont donné l'occasion d'appliquer beaucoup de notions vues en cours d'intelligence artificielle, d'aborder des problèmes quant à leurs applications, ainsi que d'utiliser et découvrir d'autres outils mis à notre disposition par des laboratoires ou la communauté.

1 Le projet

1.1 La mission

1.1.1 Objectif

Pour ce projet, nous devions repérer sur des coupes en trois dimensions issues d'imagerie à résonance magnétique (IRM) la présence de lambeaux. Les lambeaux étant des greffes faites après le retrait de tumeurs cancéreuses. Par la suite, nous appellerons aussi les lambeaux, des greffes ou *flaps*.

Pour détecter les greffes, nous nous référons à l'[échelle de Hounsfield](#). Cette échelle, présentée à la table 1, permet de connaître le type de tissus présents en fonction de son absorption de l'eau à un endroit donné. Les niveaux de gris sur les images IRM sont, eux aussi, liés à l'absorption de l'eau des tissus. Plus précisément, ils sont dus à la résonance magnétique des atomes d'hydrogène présents dans l'eau. Une couleur proche du blanc montre la présence d'os et plus proche de noir, d'air. Dans le cas de nos *flaps*, l'absorption est moins élevée que le tissu trouvable normalement à cet endroit donc, sur les images, cela se transcrit par une zone plus sombre à l'endroit du *flap*. L'absorption moins élevée est due au choix du tissu à greffer qui est, le plus souvent, de la graisse.

Matière	UH	Couleur IRM
Air	-1000	Noir
Poumon	-500	
Graisse	-100 à -50	
Eau	0	Gris
Liquide cérébro-spinal	15	
Rein	30	
Sang	+30 à +45	
Muscle	+10 à +40	
Matière grise	+37 à +45	
Matière blanche	+20 à +30	
Foie	+40 à +60	
Tissus mous	+100 à +300	
Os	+700 (os spongieux) à +3000 (os denses)	Blanc

TABLE 1 – Échelle de Hounsfield

Pour détecter les lambeaux, nous utiliserons un modèle de type U-Net [1]. Ces modèles ont prouvé leur supériorité quant à la segmentation sémantique d'images. Ils sont basés sur une architecture consistant à avoir une partie dit de "contraction" (ou encodeur) et une partie "d'expansion" (ou décodeur). Ceci permet simplement de passer une image, d'en extraire des caractéristiques grâce à des filtres de convolution

et à chaque étage dans les deux parties et à la fin, de ressortir une carte ayant pour chaque pixel, une prédiction de classe. Le but est donc de faire en sorte de rassembler les pixels ayant les mêmes classes en des blocs permettant de les catégoriser en objet. L'intérêt principal de ce type d'architecture face à un réseau entièrement convolutionnel (FCN) [2], c'est l'utilisation de *skip connections* qui permet de passer des informations extraites localement dans la première partie directement dans la seconde (par étage) pour éviter une perte d'information au fur et à mesure de l'appel des couches et favorise donc la réutilisation d'attributs (et donc la suppression de redondance). L'architecture sera plus longuement approfondie à la section 2.2.

1.1.2 Idée de réalisation

Après la première réunion, nous avons défini un plan d'action et des outils à utiliser, basé sur les premières idées de réalisation. Ainsi, dans un premier temps il nous fallait réaliser un état de l'art sur notre problématique exacte ainsi que sur son type, notamment trouver des données en lien avec les greffes après cancers Oto-Rhino-Laryngologiques (ORLs) ou, dans le cas où nous ne trouvions pas, agrandir à la segmentation de tumeurs puis à la segmentation de tous les types de cancers. Par la suite, il nous faudrait tester les modèles sur leurs données d'entraînement afin de déterminer si ces derniers pourraient être adaptés à notre problème. Ensuite, il nous faudrait passer à la réalisation d'un composant qui permet de charger les images de chaque patient. Et enfin, il nous faudrait finetuner le modèle choisi. Cela nous permettra de diminuer le temps d'apprentissage en réutilisant une partie des poids calculés dans le modèle pré-entraîné. Cette partie signerait la fin du projet d'après les premières idées de réalisation que nous avions eu ou que nos encadrants nous ont donné.

1.2 Intérêt pour les collaborateurs

Le centre François Baclesse a pour but d'améliorer le traitement des cancers. Il existe plusieurs façons de procéder, par exemple, chercher de nouvelles méthodes de traitement qui seraient plus efficaces sur tel ou tel cancer, ce qui serait fait par une partie plus médicale et biologique du laboratoire ; permettre une prise en charge plus rapide que ce soit par l'achat de nouveaux appareils, ce qui a un coût élevé, ou par la création et utilisation de logiciels accélérant la prise de bonnes images lors des examens, permettant ainsi à davantage de patients de passer des examens et donc une possible détection plus précoce de la maladie ; automatiser des tâches de détection avec une vérification humaine par la suite est aussi une façon de lutter contre le développement des cancers puisque cela permet de diminuer le besoin humain et donc de libérer les experts sur d'autres tâches.

La tâche qui nous a été donnée se classerait dans la dernière partie des procédés listés plus tôt. Pour rappel, nous voulons segmenter les greffes après cancer des patients. Cela a un objectif bien réel qui est de libérer les experts de cette tâche fastidieuse. Si la segmentation est réalisée automatiquement, les médecins n'auront plus à chercher l'endroit où se trouve le *flap*. Ils pourront donc directement passer à l'étude des résultats de l'IRM et ainsi donner rapidement leurs points de vue sur la

greffe. Ce projet pourra donc orienter les réponses des experts sur certaines questions comme "a-t-elle été bien acceptée par le corps ?", "n'a-t-on pas une nouvelle tumeur qui se développe à cette endroit ?". De plus, ils n'auront plus à passer autant de temps sur le repérage du lambeau ce qui leur permettra, par la suite, de libérer ce temps à l'étude d'autres résultats d'examens.

En résumé, notre projet annuel pourra permettre l'accélération de l'étude des résultats d'IRM des personnes ayant subi une ablation de tumeur puis une greffe au niveau de la tête et du cou. Cela donc donne accès, si besoin, à des nouveaux soins plus rapidement pour les patients. Sachant qu'une prise en charge des problèmes liés à un cancer a besoin d'être faite le plus tôt possible afin d'aider au mieux à la rémission du patient.

1.3 Organisation du travail

1.3.1 Outils utilisés

Langage de programmation Afin de trouver les meilleurs outils à utiliser pour notre projet, nous avons tout d'abord regardé les différentes bibliothèques pouvant être utiles quant à la résolution du problème associé au projet. Comme nous étions dans un problème dans la science des données, des langages comme **Python**, **R** ou encore **MATLAB** ou **Julia** permettent de traiter de larges ensembles de données de par les bibliothèques natives ou pouvant être installées séparément. Nous avons choisi le langage Python de part notre familiarité avec ce langage, mais aussi pour le nombre gigantesque de bibliothèques externes disponibles ainsi que pour sa facilité à prototyper.

Bibliothèques pour les fichiers Le projet s'inscrivant dans le domaine médical, les images ne possèdent pas les mêmes extensions que dans la vie de tous les jours comme JPEG ou PNG. Elles ont un format différent et donc une extension spécifique.

Le données étant dans le format **DICOM** (Digital Imaging and COmmunications in Medicine), il nous a fallu un moyen de les lire afin de pouvoir les visualiser mais aussi les traiter. Concernant la visualisation, nous avons utilisé le logiciel nommé **3D Slicer** permettant de lire très facilement des images médicales. Pour le traitement, différentes bibliothèques ont été utilisées pour la lecture des images (**Pydicom**) mais aussi des fichiers RT-Struct (**rt-utils**) qui permettent de stocker les contours des régions d'intérêt.

Nous avons fait le choix par la suite, pour différentes raisons qui seront abordées par la suite (à la section 2.1.4), à des images au format **NIFTI** (Neuroimaging Informatics Technology Initiative) permettant d'avoir non plus une image par coupe (comme fait pour les DICOM) mais toutes les coupes dans une archive compressée. Étant aussi plus simple pour lire les masques (que nous avons préalablement extraits et convertis en NIfTI avec le module **dcmrtstruct2nii**), cela a conclu notre choix de format sur ce dernier. Au final, les images étant sous ce format peuvent être facilement lues en utilisant les objets au sein de la bibliothèque **nibabel**.

Bibliothèques pour la science des données Différentes bibliothèques nous permettent de traiter un nombre imposant de données et cela très facilement. Comme les images peuvent être numériquement perçues comme des matrices (et dans notre cas, des tenseurs), la bibliothèque [NumPy](#) s'imposait d'elle-même ainsi que les outils de visualisation tels que [Matplotlib](#) et [Seaborn](#). Plus tard dans le projet, [Pandas](#) s'est avéré utile pour la gestion de la base de données pour les entraînements et plus globalement pour faciliter certains traitements.

Bibliothèque pour l'apprentissage profond En plus des bibliothèques classiques en sciences des données, le sujet spécifiait d'utiliser des méthodes basées sur l'intelligence artificielle et notamment les réseaux de neurones artificiels profonds. Ces derniers se trouvent dans des bibliothèques spécifiques permettant d'utiliser d'autres types d'appareils pouvant être massivement parallélisables comme les Graphics Processing Units (GPUs) ou encore les Tensor Processing Units (TPUs). Deux grandes librairies sont en compétition, [TensorFlow](#) et [PyTorch](#). Comme elles permettent tout deux de construire et d'entraîner des modèles d'apprentissage profond, le choix s'est rapidement porté sur PyTorch. En effet, nous avons réalisé ce choix là très rapidement en remarquant que bon nombre de papiers de recherche utilisaient cette bibliothèque car elle possède l'avantage d'être plus bas niveau que TensorFlow et donc de permettre d'être plus proche des opérations élémentaires de la bibliothèque mais aussi car une autre bibliothèque nommée [Monai](#) utilise PyTorch comme base. Cette dernière est une bibliothèque d'apprentissage profond basée sur PyTorch destinée à être utilisée dans le domaine de l'imagerie médicale. Étant spécifique à ce domaine, les outils trouvés lors de nos recherches sur l'état de l'art comme les métriques, les fonctions de perte ou bien encore certains réseaux sont d'ores et déjà implémentés dans Monai et donc nous permet de se focaliser uniquement sur la résolution du problème en utilisant le travail des différents contributeurs et chercheurs comme base de travail pour ce projet. De plus, afin de rendre le code plus lisible et faciliter l'utilisation de notre projet, nous avons choisi d'utiliser les bibliothèques [PyTorch Lightning](#) afin d'éviter de refaire tout le système depuis zéro et [Weights & Biases](#) (W&B) pour la visualisation des résultats et le suivi des entraînements. Ces bibliothèques ont été utilisées sur conseils de nos encadrants tout comme Monai.

1.3.2 Temps accordé au projet et organisation

Le diagramme de Gantt présenté à la figure 1 représente le temps passé en semaine sur chaque partie du projet pour chacun d'entre nous. À savoir que Guillaume n'a pas noté tous les moments où il a pu travailler sur le projet mais cela pouvait s'étendre de 6 à 20h par semaine et Raphaëlle 2 jours, soit environ 9 heures sur ce projet par semaine. Sans compter la réunion qui avait lieu chaque semaine où nous avions cours.

Dans un premier temps, nous avons fait des recherches chacun de notre côté pour la réalisation de l'état de l'art puis, avons mis en commun. Raphaëlle s'est principalement occupée de trouver des bases de données indépendantes, là où Guillaume a recherché des métriques, des fonctions de coût, des bases de données ainsi que différentes

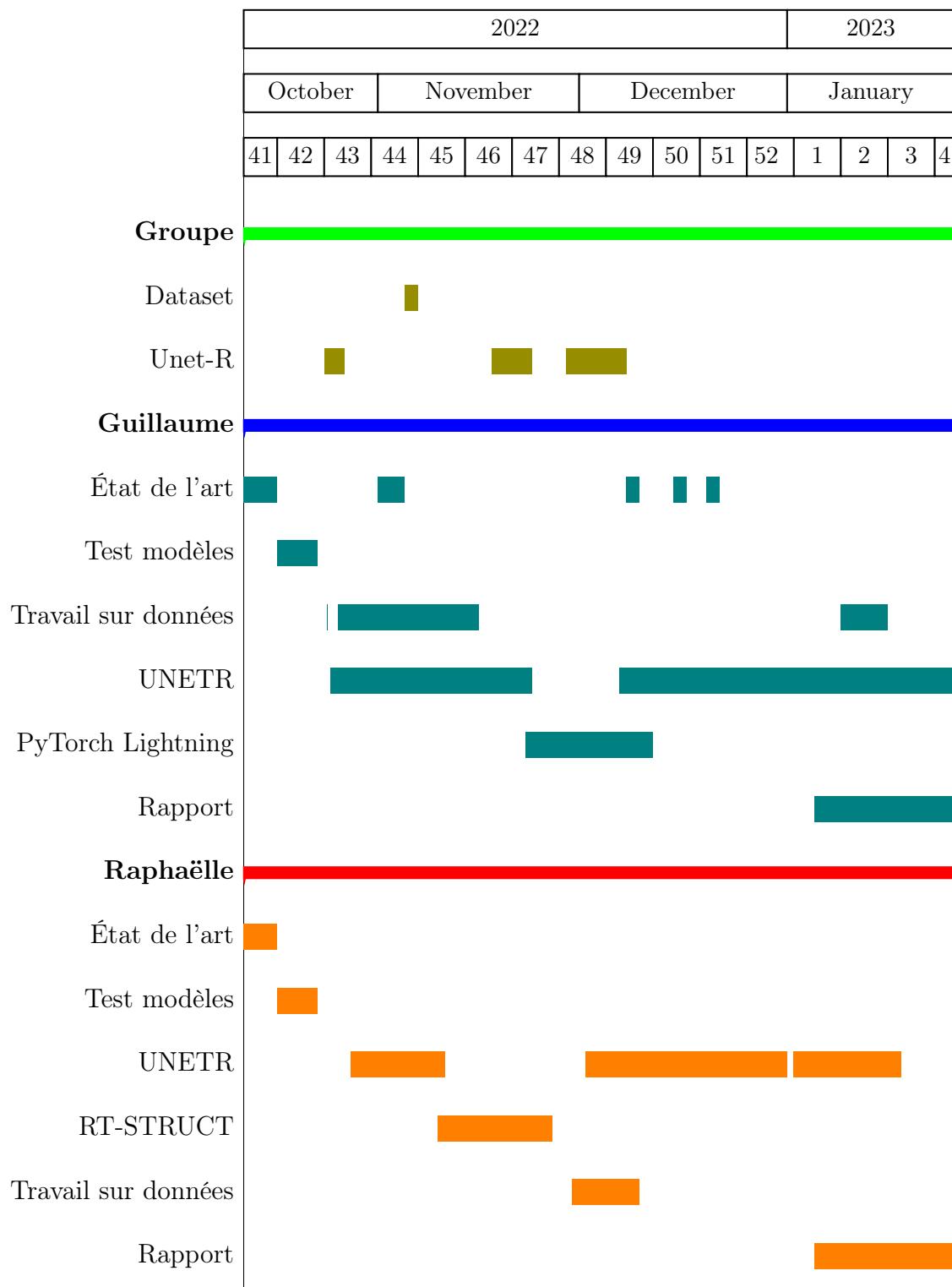


FIGURE 1 – Diagramme de Gantt

architecture pour le réseau en recherchant dans des papiers. À côté, nous nous sommes tous deux attelés à la recherche de modèles pré-entraînés, si possible, sur des bases de données accessibles.

À la suite de la réalisation de l'état de l'art, nous nous sommes partagés les différents modèles afin de pouvoir les tester. Certains d'entre eux ayant été entraînés sur des données auxquelles nous n'avions pas accès ou présentant des problèmes une fois installées. À la fin, il nous restait trois modèles et parmi eux, se trouvait le modèle que nous avons utilisé pour l'entraînement, UNETR [3] (voir section 2.3.1 pour d'information quant à son fonctionnement).

Une fois les tests effectués, nous avons créé un code permettant de transformer les données DICOM en NIfTI. L'ordinateur de Raphaëlle ne pouvant pas charger les images, nous nous sommes retrouvés afin de travailler sur le projet ensemble. Guillaume a aussi passé du temps sur cette partie de son côté.

Une fois les images transformées, il fut temps de passer à l'adaptation du modèle aux données, c'est là que l'on a créé le chargeur de données et que nous avons modifié le réseau pour qu'il puisse s'adapter au mieux à nos données. Pour faire ce changement, nous avons dû réaliser un finetuning du modèle ainsi que différents essais. C'est Guillaume qui a trouvé une solution. Raphaëlle s'est donc détachée sur une autre tâche du projet qui était de récupérer les zones d'intérêts sur les CT-scans (Computed Tomography scans) en les ressortant des RT-Struct. Cette partie a, au final, été arrêtée en cours suite à des problèmes avec le modèle et à une mise à jour des attendus du projet.

Suite à un manque de patients avec des greffes, nous avons reçu d'autres données qui n'étaient pas toutes structurées comme nous avions dans la précédente base de données. Ainsi, pendant que Guillaume continuait de travailler sur le modèle, Raphaëlle a créé un script pour ranger correctement les données.

Pour finir, Raphaëlle s'est occupé de lancer les différents entraînements du modèle, nous permettant ainsi de voir les problèmes de ce dernier et faire quelques tests sur les hyperparamètres dans le but d'améliorer les résultats, pendant que Guillaume ajustait les différentes erreurs qui se présentaient à nous.

2 Travail effectué

2.1 Les données

2.1.1 Création

Au cours de notre utilisation des données, nous avons eu des problèmes comme des absences de données dans des dossiers de patients ou, l'absence de lambeau chez d'autres. Nous avons donc demandé la raison de ces manques. Ainsi, nous avons appris comment étaient créées les données des différents patients.

Lorsqu'un patient se voit programmé un examen IRM, un dossier lui est créé avec son numéro de patient. Ce n'est qu'une fois l'examen passé que les données sont ajoutées au dossier. De plus, une fois les données récupérées, des médecins les segmentent en indiquant où se trouvent les zones d'intérêts comme le *flap*, le cœur, le cerveau, etc. Certains dossiers ont eu un traitement supplémentaire qui est la création d'une image DICOM avec uniquement la présence du *flap* sans les autres régions d'intérêt.

Ainsi, les données auxquelles nous avons eu accès n'étaient pas toujours semblables et ont nécessité des traitements particuliers afin de les homogénéiser (voir section 2.1.3).

2.1.2 État de l'art sur les bases de données

L'un des objectifs de la réalisation de l'état de l'art fut de trouver des jeux de données médicales proches de notre problème. Certains d'entre eux sont très couramment utilisés pour introduire de nouvelles méthodes ou architectures de réseau.

C'est le cas des ensembles issus du jeu de données Medical Segmentation Decathlon (MSD) [4]. Cette base de données présente différentes images IRM/CT avec des zones d'intérêt préalablement segmentées correspondant à 10 tâches à réaliser (tumeurs cérébrales, détection d'organes comme la rate, les poumons, etc). En prenant l'ensemble des données, quatorze classes sont à segmenter. D'autres bases de données ont pu être trouvées comme le Multi-Atlas Labeling Beyond the Cranial Vault (BTCV) [5] et Brain Tumor Segmentation (BraTS) qui permettent de réaliser des segmentations d'images médicales sur des zones du cerveau et plus globalement de la boîte crânienne. On peut noter aussi les bases de données Liver Tumor Segmentation (LiTS) [6], Segmentation of THoracic Organs at Risk (SegTHOR) [7] ou encore Kidney Tumor Segmentation (KiTS19) [8] qui permettent de détecter certains organes ainsi que des tumeurs qui s'y trouvent.

D'autres bases de données sont des regroupements de certaines autres bases de données comme Combined Healthy Abdominal Organ Segmentation (CHAOS) [9] ou bien encore de The Cancer Imaging Archive (TCIA) [10]. Cette dernière contient un très grand nombre de jeux de données mais malheureusement, bon nombre d'entre eux sont sous accès restreint.

Dans notre recherche, nous avons trouvé des bases de données qui mentionnaient des *flaps*. Malheureusement, ces bases de données n'étaient pas en accès public donc nous n'avons pas pu en apprendre plus et encore moins les utiliser. Elles provenaient, pour la majorité d'entre elles, de TCIA.

2.1.3 Traitement des données

Organisation Le diagramme de la figure 2 présente deux des diverses organisations des dossiers des patients que nous avons rencontré. Au départ, nous avions sous la

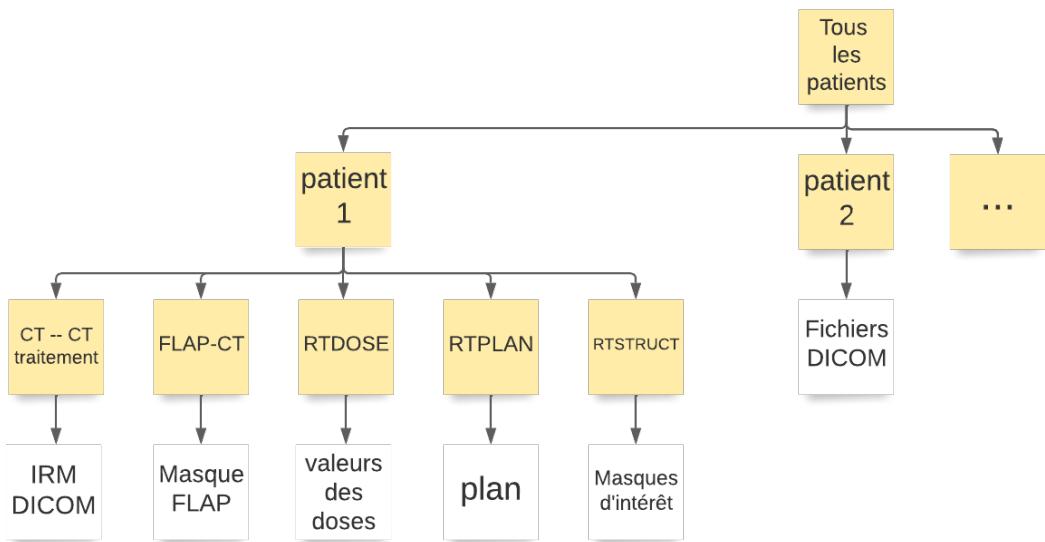


FIGURE 2 – Structuration des fichiers et dossiers

main un dossier avec peu de patients possédant des greffes, ou dossier FLAP-CT, environ 50 sur 300. Ces dossiers étaient organisés de la même manière qu'est le "patient 1" sur notre diagramme, hormis quelques exceptions : certains n'avaient pas de données dans certains des sous-dossiers comme dans CT-CT traitement, RTSTRUCT ou RTDOSE. Dues à différentes erreurs que nous avions, nous avons décidé de convertir les images des patients en NIfTI (voir section 2.1.4) en nous basant sur cette structure.

Or, par la suite, n'ayant pas assez de patients avec des *flaps*, nous avons eu accès à d'autres données qui étaient, quant à elles, organisées à la fois comme le "patient 1" mais aussi comme le "patient 2" pour d'autres. C'est-à-dire que tous les dossiers n'avaient pas la même structure car certains d'entre eux possédaient la structure que nous avions précédemment rencontré. Notamment, dans le second cas, tous les fichiers étaient dans le dossier du patient avec, comme début de nom, leur catégorie. Par exemple, nous pouvions avoir un fichier nommé "CT0211215412.dcm" correspondant à une certaine *slice* (ou coupe en français) du scan de ce patient.

Réorganisation des nouvelles données Avec cette nouvelle organisation des données, il a fallu agir. Nous avions deux choix : soit modifier le convertisseur en données NIfTI, soit créer un algorithme qui va réorganiser les fichiers des patients dans des dossiers comme pour le "patient 1". Nous avons finalement opté pour la seconde solution. Ainsi, nous avons créé un script permettant de réaliser cette réorganisation en comparant dans chaque dossier de patients, les noms des sous-dossiers et s'ils avaient la même structure que le "patient 1", on passait au suivant. Dans le cas contraire, il déplaçait les fichiers dans les dossiers cibles créés en fonction des noms de fichiers. À la fin, tous les dossiers ont pu être traités. Par ailleurs, certains fichiers dans les dossiers de certains patients n'ont pas été déplacés car ils n'étaient pas utiles pour notre tâche et dont nous ne connaissions pas l'usage de ces fichiers. Ne pas les

classer semblait plus intéressant que mal les classer, notamment les mettre dans un dossier nécessaire pour nous donnant ainsi des informations erronées à notre modèle.

Les fichiers RT-Struct Au delà des problèmes d'organisation des données, nous avons rencontré des problèmes avec la nomenclature des régions d'intérêts dans les fichiers RT-Struct des patients. Comme nous l'avons expliqué plus haut, les zones d'intérêts sont annotées à la main par différents médecins. Ainsi, chacun possède ses propres termes, sa propre façon d'annoter et il est possible qu'il fasse des fautes de frappe. Nous nous sommes donc retrouvés avec des zones d'intérêts possédants plein de noms différents les rendant difficiles à utiliser et à mettre en correspondance entre les patients. C'est d'ailleurs l'un des problèmes qui a mené à l'abandon d'une des parties du projet qui était, en plus de segmenter les *flaps*, de segmenter trois éléments : la tumeur, les organes à risque ainsi que le contour externe du patient. Par la suite, nous allons utiliser l'exemple des noms donnés aux *flaps* pour simplifier les explications. Il est important de noter que cela a dû être réalisé sur la première base de données que nous avions, les noms des régions d'intérêts étaient beaucoup plus homogènes dans la seconde et concernant les lambeaux, il n'y avait plus intérêt à réaliser cela car les régions associées à cette partie avaient toutes le même nom.

RT-Struct et récupération du lambeau Il arrivait que le *flap* du patient n'est pas été extrait du fichier RT-Struct. Nous devions donc être à même de le récupérer dans ce fichier. Pour ce faire, il a fallu ressortir tous les noms des éléments du RT-Struct pour chaque patient. Nous avons travaillé afin d'en fusionner une bonne partie afin de pouvoir rechercher les différents noms des lambeaux. En général, les médecins avaient écrit le nom de la zone suivit d'un "f" ou "F" pour indiquer qu'un *flap* y était présent. Mais, parfois nous nous sommes retrouvés à devoir traduire des mots provenant de l'anglais ou même de l'espagnol afin de comprendre ce qui se trouvait à l'endroit annoté. D'autres fois, le *flap* était annoté "flap" ou "Flap" mais aussi "flp" ou écrit avec des fautes de frappe. Nous avons aussi retrouvé des noms comme "lambeaux" ou "lambeau". Nous avons donc fait de notre mieux pour en récupérer un maximum afin que l'entraînement se passe au mieux avec le plus de patients possédant un *flap* mais, il est possible que certains ai été omis.

2.1.4 Conversion en NIfTI

Comme nous n'avions jamais abordé de sujets liés au domaine médical, il était important que nous nous renseignions sur les différents formats de données médicales ainsi que leur utilisation dans un système informatisé.

Les différents formats Lorsque nous avons eu les données, nous les avions dans le format DICOM. C'est le nouveau standard pour les images médicales afin de faciliter les transferts d'images entre les différents constructeurs de systèmes d'imagerie médical. Nous avons pu visualiser les données de certains patients grâce au logiciel **3D Slicer**. Lors de nos recherches pour en apprendre plus sur ce format, le format NIfTI est aussi ressorti qui lui était initialement utilisé pour stocker des images IRM

de cerveaux. Voici les avantages que nous avons pu trouver dans l'utilisation, dans notre contexte, du format NIfTI sur DICOM :

- permet de prendre moins de place sur disque dur car il compresse les données ;
- rassemble tous les *slices* d'un même patient dans un seul fichier ;
- peut être utilisé dans n'importe quelle version de Monai ;
- peut servir pour stocker les données de chacun des masques extraits depuis les RT-Struct.

Ces deux dernières raisons ont motivé le choix de la conversion du format DICOM au format NIfTI car sinon, nous n'allions pas pouvoir utiliser le modèle pré-entraîné du UNETR [3] (voir section 2.2) dû aux erreurs de versions introduites par Monai ainsi que PyTorch. De plus, ayant réalisé que des extracteurs de masques stockés dans les RT-Struct vers un format NIfTI existait, cela nous a davantage motivé car il nous était impossible de charger un masque sans l'image CT. Ainsi, si nous avions juste besoin d'un masque, il fallait charger toute l'image CT ainsi que le fichier RT-Struct et récupérer le masque souhaité (tout en gérant les erreurs de nomenclatures comme dit à la section 2.1.3). Il nous a donc apparu plus simple de tout convertir en NIfTI et être ainsi totalement libres sur la façon de charger les données pour le modèle ou encore pour la visualisation de celles-ci.

Réalisation de la conversion Pour réaliser la conversion, nous avons utilisé la bibliothèque `dcmrtstruct2nii`, qui dans son API, permettait de manière programmatique, de réaliser l'extraction de chacun des masques associés aux zones d'intérêt stockés dans le fichier RT-Struct du patient considéré. De plus, une option était aussi présente afin de récupérer les nomenclatures de chacune des zones d'intérêt (qui nous a donc servi pour filtrer les noms ainsi que pour réaliser des regroupements comme discuté dans la section 2.1.3). On peut aussi noter la présence de conversion automatique de l'image DICOM du CT-scan vers NIfTI.

Il n'y avait pas réellement de difficulté, mise à part de faire attention aux structures des dossiers/fichiers utilisés dans le cas où les données sont en DICOM et dans le cas où elles sont en NIfTI.

2.1.5 *Preprocessing* pour le modèle

Maintenant que nos données ont toutes la même structure et le même format, il est temps de construire la pipeline pour charger et pré-traiter les données afin de le faire passer dans le modèle, soit pour l'entraînement, la validation, les tests ou bien encore une simple prédiction.

Chargement des données Même si nos données ont été converties en NIfTI, nous avons laissé en place le code permettant de lire et charger des données au format DICOM via Monai en changeant un paramètre lié à la structure des fichiers choisie. Cela est permis car plusieurs lecteurs ont été créés pour gérer les formats et les arborescences des fichiers. De plus, ils permettent de gérer le fait qu'il y ait un masque ou non (car si aucun masque n'a été extrait, il est nécessaire d'initialiser un masque contenant que des zéros).

Pour lire les images, nous utilisons la transformation `LoadImaged` permettant de spécifier le lecteur (Pydicom, Nibabel, etc). Concernant les images au format DICOM, seule l'image 3D peut être chargée, le RT-Struct doit être chargée avec une autre bibliothèque comme `rt-utils`.

Ensuite, pour éviter des problèmes dans les pipelines, on fait appel à la transformation `EnsureChannelFirstd` permettant de s'assurer que la dimension des *slices* est en première position (après la dimension de *batch* évidemment).

Désormais, peu importe le format de données choisi, à partir d'ici, les données auront toute la même forme : taille de $512 \times 512 \times N$ où N est un nombre variable en fonction des patients et cela, pour l'image ainsi que pour le label. À noter que sur les schémas des figures 3 et 4, lorsque les deux flèches pointent vers un même bloc, cela signifie que la transformation s'applique sur l'image et le label (soit la clé principale est l'image soit le label).

Pipeline de pré-traitement pour les données de validation, test et pré-diction La figure 3 présente la pipeline utilisée pour faire le pré-traitement des données avant leur envoi dans le modèle pour la validation, le test ou bien encore la prédiction. Tout d'abord, on peut noter que les pipelines entre la validation/test et pour les prédictions sont les mêmes à la différence que pour la validation, il est obligatoire d'avoir les labels afin de pouvoir faire le calcul des métriques tandis que pour des prédictions, on suppose n'avoir aucun label (car on souhaite prédire et non vérifier). Au sein de la pipeline, il se trouve y avoir :

- la transformation `Orientationd` permettant de mettre toutes les images dans le même sens (celui étant choisi est le *LAS* avec le lit en bas et la tête la première) ;
- la transformation `Spacingsd` permettant de rééchantillonner l'image en fonction du paramètre `pixdim` afin de standardiser les *slices* des différentes images ;
- la transformation `ResizeOrDoNothingd` permettant de redimensionner les images d'entrée si l'image est trop grande (concernant la dimension des *slices*, on peut couper les *slices* qui se trouvent être les plus proches du bas du corps en fonction de l'orientation de l'image discutée précédemment). Ceci permet donc d'avoir une image pas trop grande pour éviter les problèmes mémoire ;
- la transformation `ScaleIntensityRanged` permet de normaliser les valeurs entre `a_min` et `a_max` vers `b_min` et `b_max`. Ceci facilite les traitements pour nous ainsi que pour le réseau de neurones artificiels ;
- la transformation `CropBredd` qui a été créée afin de supprimer (au mieux) le lit se trouvant le patient passant le scan en fonction de l'orientation de l'image ;
- la transformation `CropForegroundd` permet de récupérer uniquement les éléments du corps en utilisant une boîte englobante. Elle coupe ce qui est inutile permettant ainsi de recentrer l'image sur l'essentiel et éviter l'utilisation de trop de mémoire ;
- la transformation `ToTensord` permet de convertir tout tableau ou tenseur en un tenseur PyTorch pour une utilisation simplifié dans le futur.

Pipeline de pré-traitement pour les données d'entraînement La figure 4 présente la pipeline utilisée pour faire le pré-traitement des données avant leur

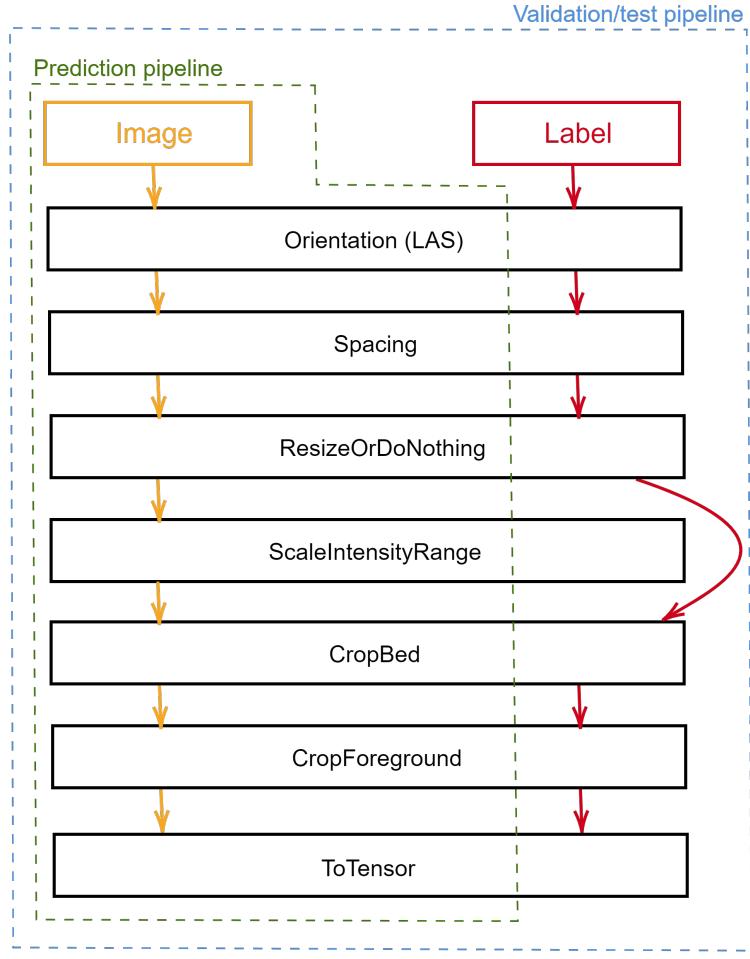


FIGURE 3 – Schéma de la pipeline de pré-traitement des données pour la validation/test et prédiction

envoi dans le modèle pour l’entraînement. Comme sur la figure 3, on retrouve un certain nombre de transformations en commun. Le but de cette pipeline est d’avoir des imagettes de taille fixe pour l’entraînement ainsi que des transformations supplémentaires sur celles-ci afin de réaliser de l’augmentation de données pour contrer le fait d’avoir peu de données. Dans cette pipeline, on y retrouve :

- la transformation **Orientationd** permettant de mettre toutes les images dans le même sens (celui étant choisi est le *LAS* avec le lit en bas et la tête la première) ;
- la transformation **Spacingsd** permettant de rééchantillonner l’image en fonction du paramètre `pixdim` afin de standardiser les *slices* des différentes images ;
- la transformation **ScaleIntensityRanged** permet de normaliser les valeurs entre `a_min` et `a_max` vers `b_min` et `b_max`. Ceci facilite les traitements pour nous ainsi que pour le réseau de neurones artificiels ;
- la transformation **CropBredd** qui a été créée afin de supprimer (au mieux) le lit se trouvant le patient passant le scan en fonction de l’orientation de l’image ;
- la transformation **CropForegroundd** permet de récupérer uniquement les éléments du corps en utilisant une boîte englobante. Elle coupe ce qui est

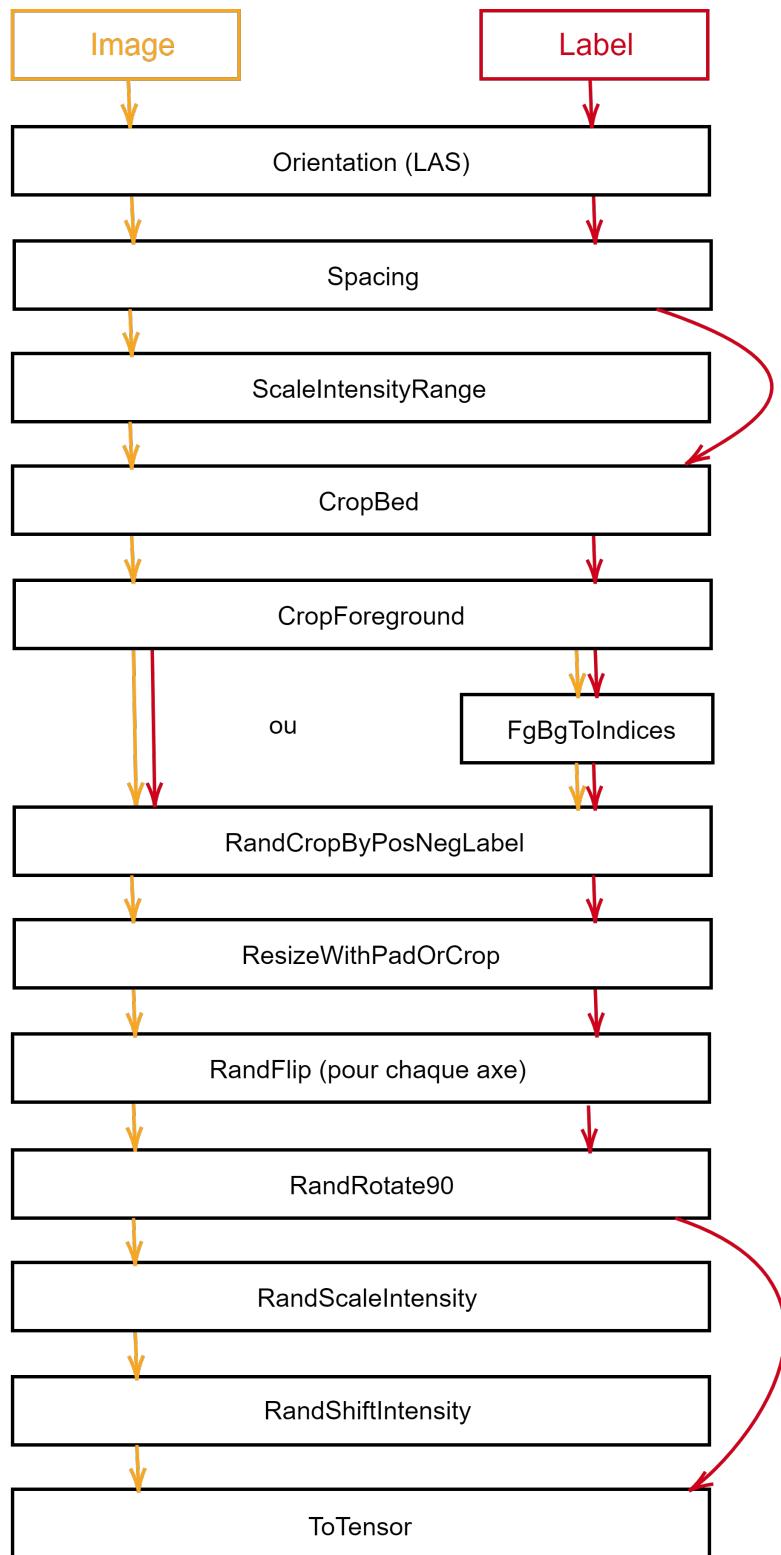


FIGURE 4 – Schéma de la pipeline de pré-traitement des données pour l’entraînement

- inutile permettant ainsi de recentrer l’image sur l’essentiel et éviter l’utilisation de trop de mémoire ;
- la transformation **FgBgToIndices** permet d’extraire les indices des pixels

- appartenant à l'arrière-plan et à ceux de l'objet. Dans notre code, on s'en sert pour spécifier que l'objet est le *flap* et que tout le reste se trouve être l'arrière-plan. Cette transformation est optionnelle et est activable via l'option `enable_fgbg2indices_feature` dans l'interface en ligne de commande (CLI) ;
- la transformation `RandCropByPosNegLabeld` permet de tirer aléatoirement plusieurs imagettes de taille fixe (spécifiée en paramètre). Celles-ci seront centrées vers l'objet tout en suivant la probabilité définie par les paramètres `pos` et `neg`. De plus, si la transformation précédente a été activée via `enable_fgbg2indices_feature`, les imagettes seront non plus centrées sur le corps du patient mais sur le lambeau. Cette transformation est la première permettant d'augmenter artificiellement le nombre de données car nous pouvons créer plusieurs imagettes contrôlé par le paramètre `num_samples` ;
 - la transformation `ResizeWithPadOrCropd` permet de redimensionner (en ajoutant des bandes ou en retirant) l'imagette en la taille de l'entrée du réseau. Cette transformation est nécessaire car il se peut que l'image soit plus petite que prévue si le centre choisi est trop proche d'un bord ;
 - la transformation `RandFlipd` permet de retourner l'imagette avec une probabilité de la faire retourner. Elle aide à rendre le modèle invariant aux retournements sur chacun des trois axes de l'image ;
 - la transformation `RandRotate90d` permet d'autoriser une rotation de 90 degrés avec une certaine probabilité dans les différents axes spatiaux. Cette transformation aide à rendre le modèle invariant aux rotations ;
 - la transformation `RandScaleIntensityd` permet d'aléatoirement (aussi avec une certaine probabilité) modifier l'étendue des valeurs d'intensité (celles que nous avons normalisées). Ceci aide le modèle à mieux généraliser et donc de ne pas se focaliser sur des valeurs particulières d'intensité ;
 - la transformation `RandShiftIntensityd` permet de bouger l'étendue vers le haut ou le bas de manière aléatoire avec une certaine probabilité. Le but est le même que pour la transformation précédente ;
 - la transformation `ToTensord` permet de convertir tout tableau ou tenseur en un tenseur PyTorch pour une utilisation simplifié dans le futur.

2.2 Modèles et outils pour leur entraînement

Dans cette partie, nous allons vous présenter notre travail de recherche et de réflexion ainsi que d'ingénierie concernant la partie apprentissage profond, celle qui nous a été incitée à utiliser pour résoudre le problème auquel nous devions faire face.

2.2.1 État de l'art

Comme nous allons vous présenter les différents outils, il est tout d'abord nécessaire d'avoir une vue globale autour de la segmentation de zones d'intérêt dans des images et plus spécifiquement dans des données stockées et modélisées en trois dimensions ainsi que dans le domaine médical.

Approches Différentes approches sont disponibles en apprentissage profond : apprentissage supervisé, apprentissage non-supervisé, apprentissage semi-supervisé, apprentissage auto-supervisé, etc. Nous avons survolé différentes méthodes au sein de quelques unes de ces approches comme l'ASDNet [11] (Attention Based Semi-supervised Deep Networks) ou encore HSSL [12] (Hierarchical Self-Supervised Learning) mais celles-ci étaient assez loin de ce qui était attendu. Le but était d'obtenir des résultats et plus globalement expérimenter autour du problème dans un contexte d'un apprentissage supervisé. En effet, les données étant annotées d'une vérité terrain, il est plus intéressant de les utiliser, au moins dans un premier temps. Nous nous sommes donc par la suite uniquement concentrés sur des modèles avec une supervision complète.

Recherche sur les modèles avec entraînement supervisé Dans cette approche, la plupart des réseaux utilisent un principe commun qui a été introduit avec les U-Net 2D/3D [1, 13] qui a été désigné pour la segmentation sémantique. UNet++ [14], nnU-Net [15], UNETR (UNet TRansformer) [3] sont quelques unes des adaptations qui nous avons pu croisé. Mais ce type d'architecture n'est pas la seule existante, il y a notamment les FCN (Fully Convolutional Networks) [2], SegNet [16] ou bien encore Mask-RCNN¹ [17] et un modèle de segmentation multi-modale [18]. Ce dernier a notamment été placée dans notre recherche car il y avait eu une discussion à la première réunion du projet sur la possibilité d'utiliser d'autres données que la zone à segmenter (comme d'autres zones d'intérêts ou bien des informations sur des caractéristiques diverses concernant les patients comme le sexe ou bien la taille).

Métriques Dans les différents papiers de recherche ou thèses qui nous sont passés sous les yeux, nous avons pu déterminer un certain nombre de métriques mais seules quelques unes sont très utilisées. Les autres utilisent des principes très similaires à celles que nous allons présenter. Pour avoir une vue plus complète de nos recherches là-dessus, vous pouvez vous référer à la section nommée "Métriques" du document disponible à l'annexe A.

Il existe tout d'abord la Dice Similarity Coefficient (DSC) ou Dice Score ou juste *Dice* qui calcule un score de similarité entre la zone prédite et la vérité terrain. La sortie est une valeur assimilable à un pourcentage et l'étendue se trouve être entre 0 et 1. On essaye de déterminer un score de recouvrement dans le cas de la segmentation entre la prédiction et la vérité terrain donc plus la valeur est proche de 1, plus la segmentation est réussie. En notant G_i la valeur du voxel d'indice i pour la vérité terrain, P_i celle associée dans la prédiction et I l'ensemble des voxels présents dans l'image, la DSC est définie comme ceci :

$$\begin{aligned} \text{Dice}(G, P) &= \frac{2 \cdot |G \cap P|}{|G| + |P|} \\ &= \frac{2 \sum_{i=1}^I G_i P_i}{\sum_{i=1}^I G_i + \sum_{i=1}^I P_i} \end{aligned}$$

1. Utilisé pour de la segmentation d'instance ainsi que de la détection d'objets multiples dans une image 2D.

Une autre qui reste très utilisée est la distance de Hausdorff (HD) [19]. C'est une mesure de distance permettant de mesurer l'éloignement de deux sous-ensembles provenant d'un même espace métrique en essayant de vérifier les différences de formes entre les deux sous-ensembles passés en entrée. Dans notre cas, les deux sous-ensembles seront les masques correspondant à la prédiction et à la vérité terrain. La valeur ressortie par cette métrique est une valeur en millimètres (mm). Concernant l'interprétation, plus elle est proche de 0, plus la forme de la prédiction est proche de celle attendue dans la vérité terrain et donc signifie que la segmentation est bonne. En notant de la même façon que pour la métrique précédente, avec δ la mesure de distance associée à l'espace, $S(R)$ est l'ensemble des voxels de surface de R et $d(v, S(R)) = \min_{s_R \in S(R)} \delta(v, s_R)$, on a :

$$HD(P, G) = \max \left\{ \max_{s_P \in S(P)} d(s_P, S(G)), \max_{s_G \in S(G)} d(s_G, S(P)) \right\}$$

Les mesures ayant été prises dans un espace euclidien, la fonction δ est donc la distance euclidienne. De plus, cette métrique est très sensible aux valeurs aberrantes donc il est nécessaire de n'utiliser qu'un certain pourcentage de ces valeurs. Celle communément prise est 95%, donnant ainsi la métrique HD95 et permet de spécifier qu'on ne prend que jusqu'au 95ème percentile des distances entre les deux sous-ensembles.

Il est utile de noter qu'utiliser la distance de Hausdorff seule ne garanti pas une bonne segmentation. En effet, on ne compare que les formes externes entre elles donc s'il y a un "trou" dans la zone segmentée, il se peut qu'il ne le détecte pas donc combiner cette métrique avec la DSC permet de corriger ce problème en plus d'être un indicateur pertinent. De plus, la segmentation de la vérité terrain n'étant peut être pas la plus précise possible, il se peut que la valeur ne soit pas proche de zéro.

Fonctions de coût Nous avons trouvé différentes fonction de coût dans différents papiers et notamment un [20] étant un *survey* sur cette question. Nous avons plutôt décider d'utiliser les fonctions basées sur la métrique *Dice* car cette métrique est très pertinente pour la segmentation. De plus, le choix s'est aussi porté sur cette métrique car nous voulions principalement vérifier que la méthode utilisée était pertinente pour le problème et donc nous avons pris celle qui nous semblait le plus simple et qui respectait les contraintes que nous avions sur les données et les objectifs. De plus, la plupart de ces fonctions sont disponibles dans **Monai**.

Parmi les fonctions de coût sélectionnées, nous avons la *DiceLoss* [21] définit comme $\mathcal{L}_{dice} = 1 - Dice(G, P)$. À noter que la puissance mise aux termes au dénominateur peuvent être ajoutés si on le souhaite dans Monai avec l'option `squared_pred=True` donnant ainsi

$$Dice(G, P) = \frac{2 \sum_{i=1}^I G_i P_i}{\sum_{i=1}^I G_i^2 + \sum_{i=1}^I P_i^2}$$

L'autre sélectionnée est une combinaison entre la *DiceLoss* [21] et l'entropie croisée

(CE) et est donc définie comme :

$$\begin{aligned}\mathcal{L} &= \lambda_1 \mathcal{L}_{\text{dice}} + \lambda_2 \mathcal{L}_{\text{ce}} \\ &= \lambda_1 (1 - \text{Dice}(G, P)) + \lambda_2 \left(-\frac{1}{m} \sum_{i=1}^m y_i \cdot \log(\hat{y}_i) \right)\end{aligned}$$

avec y et \hat{y} les masques vectorisés de G et P , et les paramètres λ_1 et λ_2 des coefficients de pondération (souvent mis à 1 pour que chaque partie est autant d'importance). Cette fonction a été reprise de ce qui est implémentée dans [Monai](#).

2.2.2 Sélection et tests des différentes architectures

Parmi toutes les architectures de réseaux mentionnées dans la section précédente dans l'état de l'art, nous avons décidé de nous concentrer sur les réseaux U-Net 3D [13], SegNet [16], Mask-RCNN [17] ainsi que UNETR [3]. Nous les avons préféré afin de vraiment tester des architectures qui possèdent de nombreuses différences dans leur implémentation et mécanisme. Avant de comprendre à fond leur architecture, il nous a été vivement conseillé de réaliser des tests des réseaux sélectionnés et ensuite de nous pencher sur une seule qui nous semblait la plus intéressante. Nous nous sommes donc mis à rechercher des implémentations existantes ainsi que des modèles pré-entraînés pour les tester sur les données ayant servi à leur entraînement. Malheureusement, des problèmes ont eu lieu lors du test du U-Net 3D [13] donc nous n'avions pas pu le tester. Plus tard, nous avons appris que la version 2D de ce réseau [1] avait été expérimentée par Valentin Renier, un ancien étudiant de M2 IDM ayant précédemment travaillé sur ce projet.

SegNet L'un d'eux s'appelle SegNet [16]. Il a fait partie de notre sélection car avec nos encadrants, nous ne savions pas encore si nous allions partir sur du 2D ou 3D donc cela nous permettait de tester une architecture différente que celle des U-Net [1, 13] et de leurs adaptations. Nous nous sommes principalement concentrés sur des tests mettant en jeu des bases de données non médicales car à notre connaissance, il n'a pas été appliqué dans des segmentations d'images médicales. Malgré cela, nous avons tout de même choisi de tester son efficacité car celui pouvait être utile si nous partions sur du 2D. Le modèle que nous avons sélectionné a été entraîné à segmenter des parties d'une photo de châteaux. Sur la figure 5, vous pouvez retrouver une

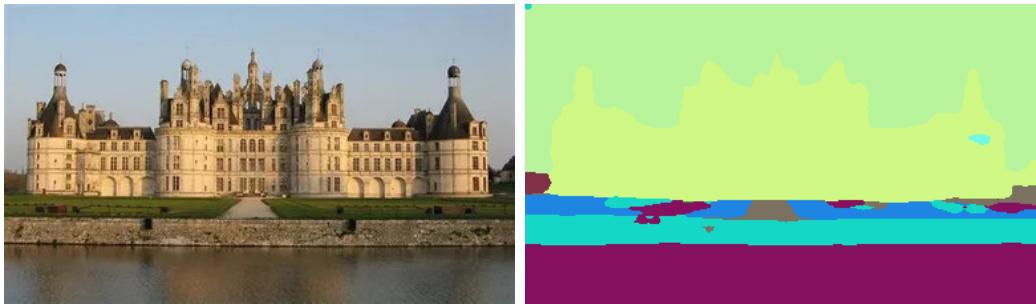


FIGURE 5 – Segmentation d'une photo d'un château par un modèle basé sur SegNet

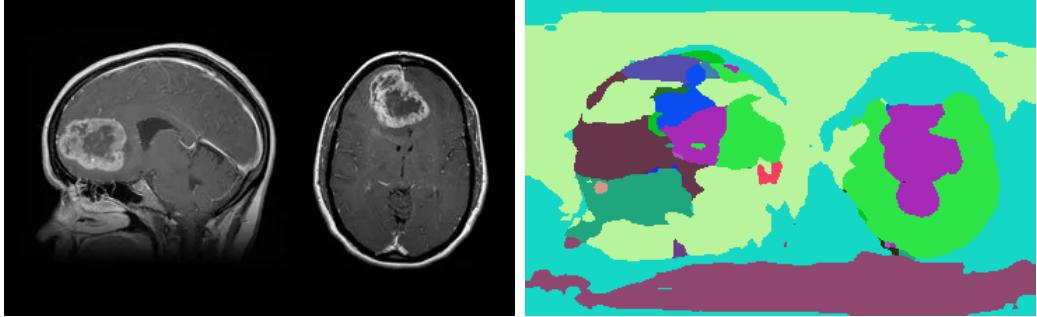


FIGURE 6 – Segmentation d’une image médicale par le modèle pré-entraîné sur les images de châteaux

segmentation réalisée sur une photo de château ainsi que sur la figure 6 qui se trouve être une image médicale tirée d’internet. Il en ressort une très bonne performance sur des images de châteaux et, une mauvaise sur les images médicales mais cela était prévisible car ce modèle n’a jamais vu d’images provenant du domaine médical auparavant. Pour ce modèle, il faudrait un entraînement complet sur nos données puisqu’elles ne sont pas de même nature. De plus, il ne prend pas en compte l’aspect 3D donc il faudrait l’adapter pour notre problème.

Mask-RCNN Dans le cas de ce réseau [17], au-delà de faire de la segmentation, il crée des boîtes de détection autour des objets d’une image. C’est la raison pour laquelle nous l’avons sélectionné car lors des discussions au début du projet, l’objectif était de réaliser une segmentation du lambeau mais aussi possiblement une détection. De la même façon que pour SegNet, nous n’avons pas trouvé de modèles ayant été entraînés sur des données médicales et nous sommes donc basés sur celui qui a été entraîné sur la base de données nommée **COCO** (Common Objects in Context) [22]. On y retrouve une bonne détection des différents objets, visibles sur la figure 7,

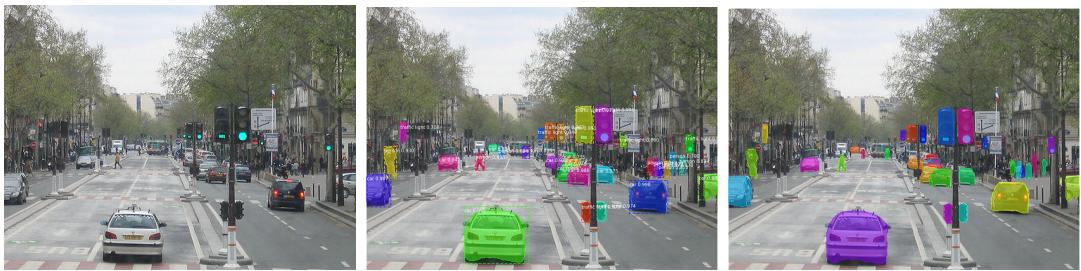


FIGURE 7 – Segmentation d’instance par Mask-RCNN entraîné sur le jeu de données COCO [22] - Ici, les boîtes de détection ont été retirées mais il est possible de les laisser en enlevant une option dans l’implémentation que nous avons utilisé.

mais, il nous faudrait l’entraîner sur nos données pour pouvoir vérifier sa validité. Cela prendrait trop de temps. De plus, ce modèle est plutôt fait pour réaliser de la segmentation d’instance et non de la segmentation sémantique, cela n’était donc pas forcément pertinent de l’utiliser dans notre contexte.

UNETR Ce dernier réseau a été sélectionné car nous voulions tout de même tester un modèle basé sur une adaptation d'un U-Net [1] et notamment de sa version 3D [13]. Nous avons choisi spécifiquement le UNETR [3] car les travaux sur celui-ci sont très récents et présentent de très bons résultats sur des jeux de données médicales. De plus, ils intègrent un Transformer [23] nommé Vision Transformer (ViT) [24] étant spécifique pour les images, un type de couche utilisant le mécanisme d'attention [23] et sur-performe les résultats obtenus avec des techniques n'utilisant pas ce genre de mécanisme.

Comme il a été entraîné sur une base de données médicales, il était très intéressant de le tester car le modèle aura vu des données de nature très similaire à ce que nous possédons, notamment en utilisant l'**échelle de Hounsfield**. Outre cela, les données sur lequel le modèle que nous avons choisi a aussi été entraîné sur des images 3D, cela a été déterminant dans le choix final de garder cette architecture pour nos expérimentations. Cependant, nous avons eu quelques problèmes avec le modèle disponible ainsi qu'avec le code sur l'**implémentation officielle** du papier [3] car celui-ci est défini comme étant été entraîné sur la base de données BTCV [5] alors qu'il l'était sur une autre base de données qui se trouve être le MSD [4]. La figure 8 présentent les résultats obtenus avec des données provenant de BTCV [5] et la figure 9 sur MSD [4] pour la tâche consistant à segmenter la rate. On observe que sur la figure 9, les

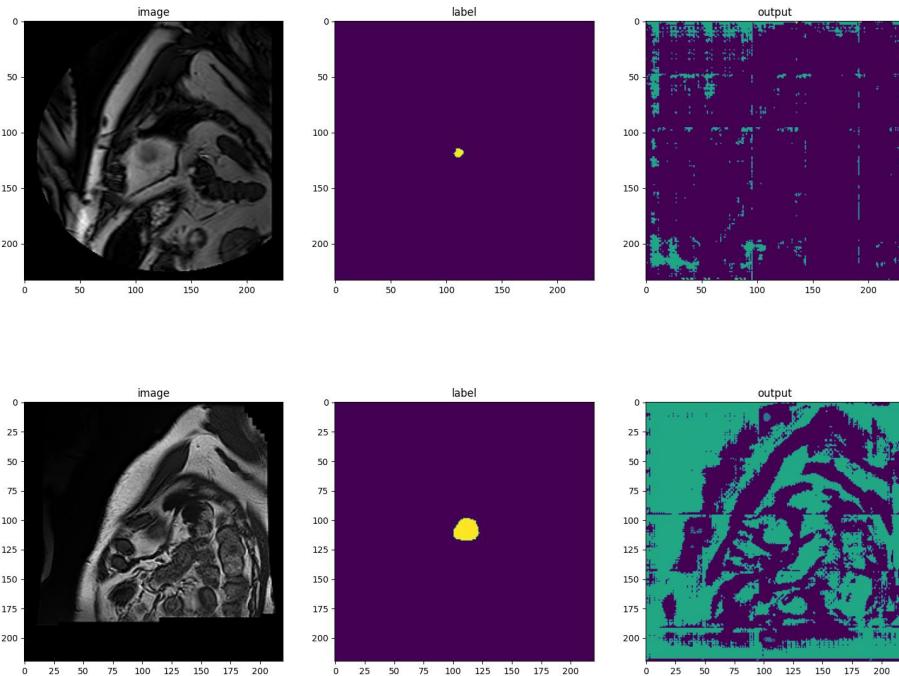


FIGURE 8 – Test du modèle pré-entraîné sur des images du dataset BTCV [5] - L'image de gauche se trouve être l'image à segmenter, l'image du centre est le masque de segmentation provenant de la vérité terrain et l'image à droite le masque de segmentation prédit par le modèle.

résultats sont vraiment bons, et donc il devrait être bon aussi pour la segmentation du *flap* qui reste dans le même principe. Les résultats montraient tout de même

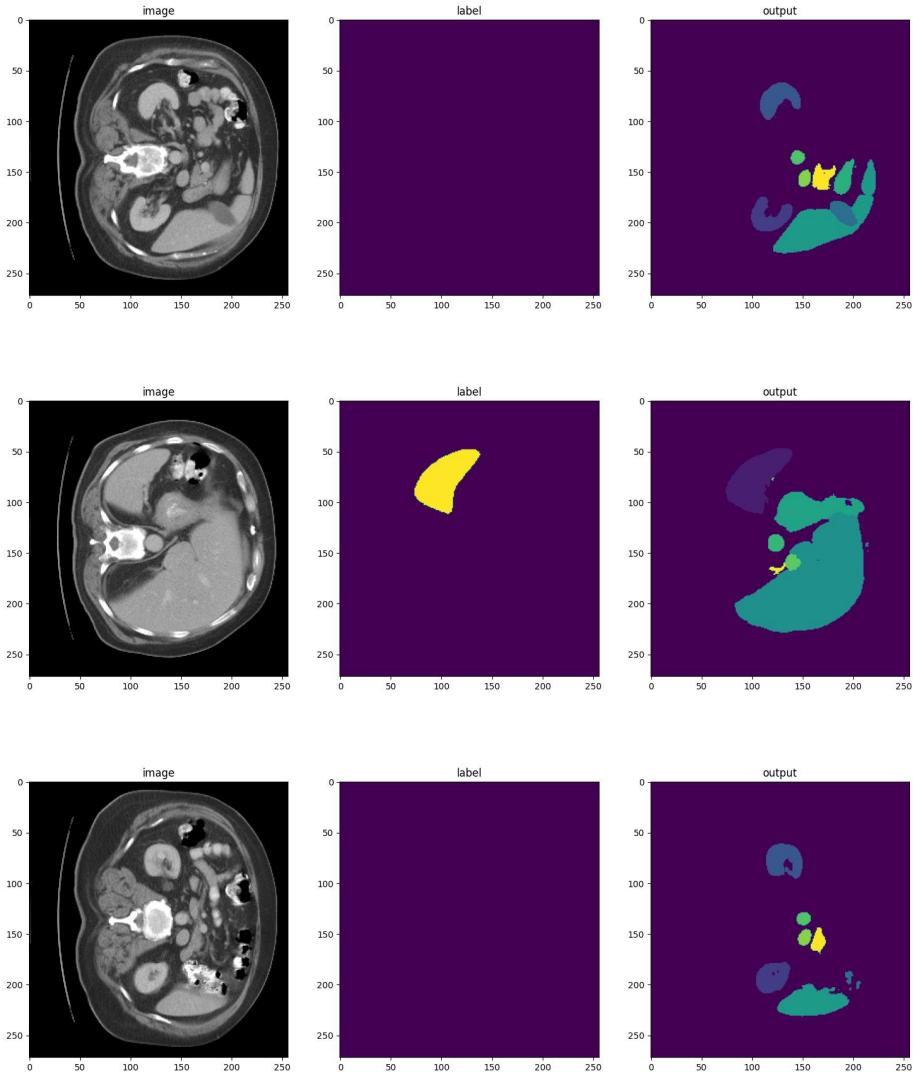


FIGURE 9 – Test du modèle pré-entraîné sur des images du dataset MSD [4] - L'image de gauche se trouve être l'image à segmenter, l'image du centre est le masque de segmentation provenant de la vérité terrain et l'image à droite le masque de segmentation prédit par le modèle. À noter ici que seule la rate cherchait à être segmenté mais comme ce modèle a été entraîné sur un total de quatorze organes, il segmente tout de même les autres lors de la prédiction.

quelques artefacts, principalement lorsque les organes ne sont pas bien visibles, mais rien qui empêche une bonne segmentation au final.

2.3 Sélection de l'architecture finale

Comme cela a pu être abordé précédemment, nous avons fini par choisir le UNETR [3] comme réseau de base pour notre réflexion et adaptation de celui-ci pour la résolution de notre problème. Ce réseau a été choisi en particulier pour ses performances de segmentation de volume face aux modèles basés sur d'autres méthodes comme 3D

U-Net [13], AttUNet [25] ou encore nnUNet [15]. Ces performances peuvent provenir du fait de l'architecture utilisée pour concevoir ce réseau. Cela va donc être abordé dans la section 2.3.1.

2.3.1 Compréhension de l'architecture du réseau

U-Net 3D Avant de discuter à propos du UNETR [3], il est nécessaire de rapidement aborder le U-Net 3D [13]. Comme le montre la figure 10, le réseau se compose de deux parties :

1. un chemin contractant (ou partie encodeur) où le principal objectif est de pouvoir extraire des attributs à différentes échelles de l'image ;
2. un chemin d'expansion (ou partie décodeur) qui a pour but de combiner efficacement les informations extraites dans la première partie aux différentes échelles.

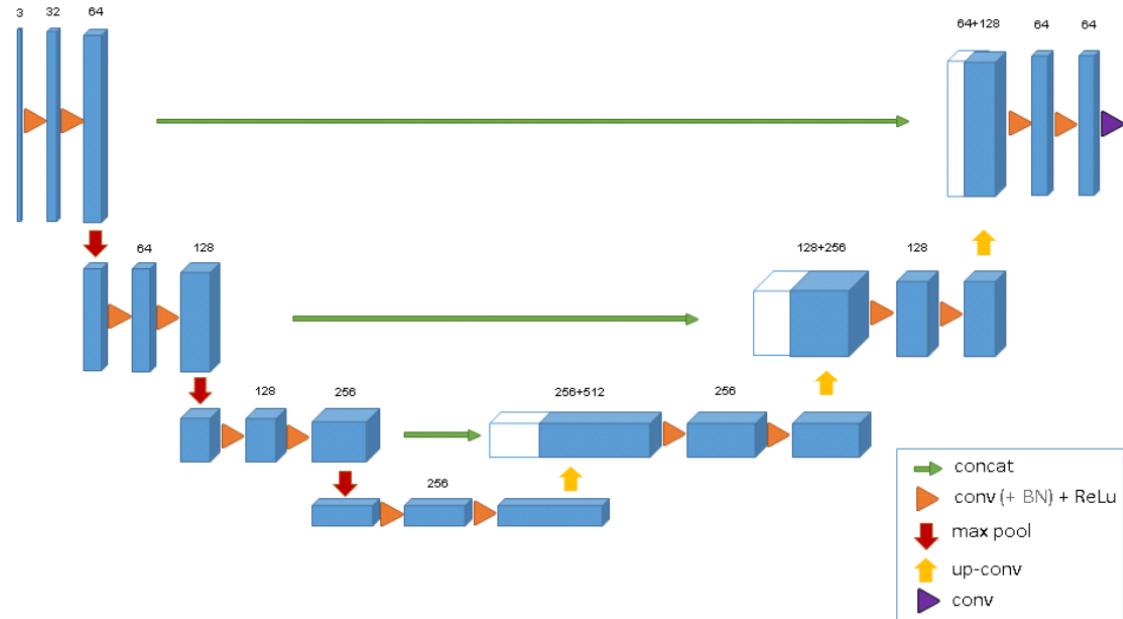


FIGURE 10 – Architecture d'un 3D U-Net

Ce réseau étant entièrement convolutionnel, aucune couche entièrement connectée ne se trouve dans celui-ci. La première partie est réalisée par plusieurs couches convolutionnelles par étage pour réussir à combiner des informations de manière efficace, mais entre les étages, on utilise une opération de *pooling* qui est utilisée pour baisser la taille de l'image afin d'extraire des informations à différentes échelles. De plus, cumuler les couches convolutionnelles permet d'avoir besoin de beaucoup moins d'exemples pour l'entraînement des paramètres du réseau. En arrivant à un certain nombre d'étages, le but est de retrouver une image avec la même taille que l'image d'entrée mais possédant des probabilités d'appartenance aux classes associées aux objets à segmenter et cela pour chaque pixel. C'est ici que la seconde partie du réseau entre en jeu : elle prend à chaque étage, la sortie de l'étage précédent

qui a été "agrandie" (*upsampled*) ainsi que la sortie associée au même étage que celui considéré provenant de la première partie. Ces deux entrées sont concaténées et l'entrée provenant de la première partie utilise donc une *skip connection* comme celle disponible dans les DenseNets [26]. Ceci permet d'avoir une certaine consistance entre les informations traitées à basse échelle mais aussi dans celles plus hautes. Les deux parties sont symétriques, d'où l'appellation "U Network". Au premier étage et dans la seconde partie, une convolution 1x1 est ajoutée afin de réduire le nombre de canaux de sortie de la carte de segmentation au nombre de classes.

UNETR Désormais que nous avons une vue assez globale d'un U-Net [2, 13], il nous est donc plus facile d'expliquer le fonctionnement du UNETR [3] car celui-ci est quasiment le même. En effet, comme le montre le schéma disponible sur la figure 11, seule la partie encodeur est construite différemment. On voit que c'est ici qu'on utilise le ViT [24] afin d'extraire les *features* en utilisant le mécanisme d'attention [23]. Pour les extraire aux différents étages, on utilise douze *layers* au sein du ViT

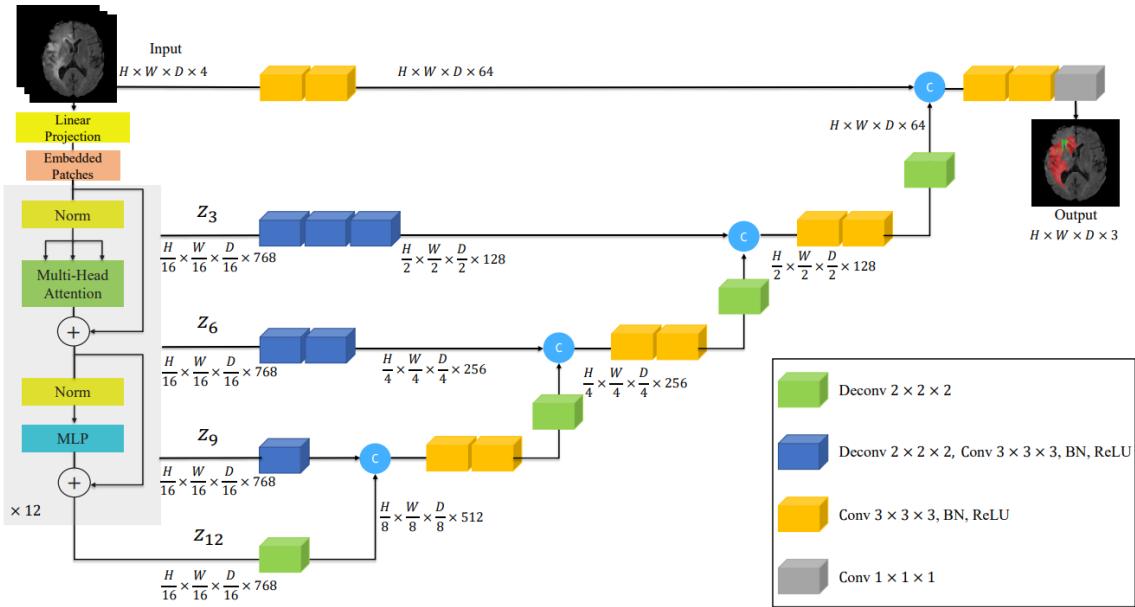


FIGURE 11 – Architecture d'un UNETR

[24] et on prend trois des douze sorties provenant des états cachés (mais il y a quatre étages, donc le premier étage prendra comme entrée l'image d'entrée comme montré dans le schéma). Le ViT [24] prend en entrée une image 2D de taille fixe dans lequel il découpe des patchs correspondant à des mots (car les Transformers [23] ont initialement été introduits pour le traitement du langage naturel) mais traite le tout comme une séquence temporelle. Or, on doit l'utiliser pour des images de taille non fixées et en 3D. Ce qui a été réalisé par les chercheurs et dans notre code, ce sont des extractions de patchs de taille fixe à partir de l'image initiale et qui se trouvent être de même taille que l'image d'entrée du ViT [24] lui permettant de créer ses propres patchs pour utiliser le mécanisme d'attention [23] entre eux. Concernant le 3D, lorsque l'on visualise l'image 3D, on la visualise comme une image 2D (chaque *slice*) que l'on fait dérouler vers le haut ou vers le bas ce qui est la même idée si

on prend d'un point de vue temporel car on a respectivement le passé et le futur. Cela permet de visualiser l'image 3D dans une vidéo par exemple (uniquement s'il n'y a pas d'aspect temporel dans l'image, comme une séquence d'image 3D). Il est donc dans la continuité de traiter la troisième dimension comme étant une dimension temporelle lors du traitement dans le ViT [24]. Afin de compléter la partie encodeur du UNETR [3], des couches de convolution et de déconvolution sont placées afin d'obtenir le même nombre de *features* extraites que dans le U-Net 3D [13].

Concernant le décodeur, il n'y a pas de changements notables car avec la partie précédentes, on s'est ramené au même nombre de canaux de sorties à chaque étage, à part le fait qu'il y est un étage supplémentaire.

2.3.2 Finetuning du modèle

Modèle pré-entraîné Comme on a pu le dire à la section 2.2.2, nous avons repris des modèles qui étaient entraînés sur les mêmes jeux de données que ceux utilisés pour les tests. Nos encadrants nous ont donc conseillé de partir du modèle pré-entraîné sur le jeux de données MSD [4]. En effet, entraîner un modèle depuis une initialisation aléatoire peut être long car il faudra partir de zéro et complexe car on peut ne pas être proche du point où les paramètres sont les meilleurs possibles pour l'ensemble de données considéré. De plus, en réutilisant un modèle déjà entraîné sur des images de même modalité, des informations déjà apprises peuvent être très utiles pour nous. Or, ce modèle a été appris avec quatorze classes en sortie alors que nous, en objectif premier, n'en avions besoin que de deux. Il est donc nécessaire de le finetuner à nos données afin d'utiliser en partie les connaissances déjà acquises.

Différentes façons de finetuner Lors de nos recherches et des explications de nos encadrants, différentes façons de finetuner un modèle pré-entraîné ont été abordés.

Parmi celles-ci, la première provient d'un papier [27] discutant du finetuning d'un modèle type U-Net [1] sur lequel ils utilisent des données ayant une modalité assez proches de celle du modèle pré-entraîné qu'ils ont choisi. Ils montrent que si décide de finetuner le moins de couches possible, il est préférable de modifier les couches qui sont peu profondes, donc celles appartenant à la partie encodeur.

L'autre façon qui nous a été conseillée ainsi qu'observée dans ce papier [28], est de supprimer et/ou réinitialiser les poids de une ou plusieurs des dernières couches du réseau. En effet, comme étant beaucoup plus liées au problème cible, modifier leur structure pour les adapter à notre problème peut permettre d'obtenir de meilleurs résultats que la méthode précédente. De plus, lors de l'entraînement, les couches non modifiées sont gelées durant un certain temps afin de permettre d'uniquement entraîner les couches modifiées pour que le tout converge vers quelque chose de stable avant de tout dégeler afin de finir l'entraînement de tout le modèle. Nous avons donc décidé de partir sur cette option.

Une autre option serait d'utiliser l'option précédente mais en dégelant les couches de manière aléatoire suivant une distribution uniforme. En effet, cette option a été abordée car nous manquions souvent de mémoire GPU ou de nombreux problèmes en lien avec elle apparaissaient lors du dégèle du backbone. Pour essayer de régler ces bugs, nous avions essayé de passer par une précision plus basse [29] sans succès.

Réalisation du *finetuning* Pour faire cela, nous avons remplacé la dernière couche du réseau qui avait quatorze canaux (car quatorze classes) en un autre initialisé aléatoirement avec deux canaux en sortie. Nous avons laissé la possibilité de réinitialiser les poids des couches les plus profondes jusqu’aux toutes premières avec un hyperparamètre contrôlable via le CLI. Il réinitialise les poids des convolutions avec l’initialisation de Kaiming [30] et les biais sont juste réinitialisés à zéro. En effet, le choix de l’initialisation et de la méthode est importante car cela peut déterminer le point le plus bas dans les alentours de l’initialisation et donc vers où le modèle convergera ainsi que potentiellement sa vitesse de convergence.

Au fur et à mesure d’entraînements ratés, nous avons pu déterminer que la meilleure valeur d’*epoch* pour dégeler l’entièreté du réseau était de 15 sur nos données avec les autres hyperparamètres mis.

2.3.3 Entraînement de notre modèle

Bibliothèques pour faciliter l’entraînement Sur conseils de nos encadrants, nous avons repris le code officiel associé au papier sur le UNETR [3] que nous avons adapté avec PyTorch Lightning. Cette bibliothèque permet de gérer elle-même tout ce qui est relatif à l’entraînement de modèles en PyTorch comme la gestion des chargeurs de données, l’envoi des données vers un ou plusieurs appareils (CPU, GPU, TPU, etc), des boucles d’entraînement, d’évaluation, de test et de prédiction, les *callbacks*, les *loggers*, et encore bien d’autres choses permettant ainsi de plutôt se concentrer sur le réseau et les données. De plus, on a utilisé la version bêta du *Lightning CLI* afin de pouvoir gérer plus facilement les hyperparamètres pour l’entraînement directement via le CLI ainsi que pour avoir une trace des hyperparamètres que nous avons pu utiliser lors des différents lancements.

De plus, nous avons aussi mis en place un *logger* lié à W&B afin de plus facilement visualiser l’entraînement. En effet, nous aurions pu utiliser les outils de visualisation que nous avions mis en place mais cette technologie possède comme avantage de stocker les informations sur un serveur distant et donc permettre l’accès à tous les membres du projet au panneau de visualisation ainsi qu’aux paramètres du modèle. Ce panneau contient différentes informations que nous décidions ou non de logger. Nous avons donc décider de logger :

- les valeurs de la fonction de perte à chaque *epoch* pour l’entraînement ainsi qu’à chaque validation ;
- les différentes métriques que nous avons utilisées soit *Dice* et *HD95* [19] (tout d’abord seulement avec l’arrière-plan mais plus tard aussi sans l’arrière-plan pour avoir une meilleure vue uniquement sur la segmentation des lambeaux) ;
- l’*epoch* en fonction du *training step* pour mieux voir où on en est dans l’entraînement et car c’est aussi plus parlant que garder le tout en *training step* ;
- le taux d’apprentissage car celui-ci décroît dû à un *learning rate scheduler* que nous discuterons plus tard ;
- lors de chaque validation, les résultats de segmentation sont loggés dans un tableau dans lequel on peut visualiser de manière interactive sous forme de masques posés sur les images. Nous utiliserons cela pour illustrer nos propos dans les sections 2.3.4, 2.3.5 et 2.3.6. À noter que nous avons réalisé plusieurs

types de visualisation qui sont disponibles dans une classe annexe au modèle afin de gérer le tout de manière efficace (celle présentée, sous forme de vidéos, et tout ça dans une table ou non le tout gérable par un paramètre pour limiter le nombre d'élément à logger).

Outils pour l’entraînement Nous avons mis en place plusieurs *callbacks* dans le but de faciliter l’entraînement. Tout d’abord, nous avons utilisé le *BackboneFinetuning* permettant de dégeler le *backbone* au moment que nous choisissons. On peut noter aussi l’utilisation d’un *early stopping* qui est une méthode de régularisation dont le but est d’éviter d’avoir l’effet de sur-apprentissage (le modèle sur-performe sur le jeu de données d’entraînement mais généralise pas assez sur ceux d’évaluation et de test). Enfin, l’utilisation d’un *checkpoint de modèle* afin de permettre de sauvegarder les modèles de manière automatique.

Méthode Lors de l’initialisation, notre base de données est séparée de la façon suivante : 70% des données pour l’entraînement, 10% pour la validation ainsi que 20% pour l’ensemble de test. Nous avons vérifié le bon équilibre des classes entre les patients ayant un lambeau et ceux n’en ayant pas pour chacun des ensembles de données. De plus, chacun des ensembles possède un ensemble de transformations associées comme expliqué dans la section 2.1.5.

Concernant l’entraînement, comme l’entrée du modèle doit être d’une taille fixe et comme dit dans les sections 2.1.5 et 2.3.1, des imagettes de taille fixe sont extraites de l’image et passés directement dans le réseau pour parfaire son apprentissage. Ensuite, un calcul classique de perte entre en jeu ainsi que son *logging* sur l’interface de W&B.

Concernant la validation, le test et plus généralement, n’importe quelle prédiction, il est nécessaire de segmenter l’entièreté de l’image et non seulement quelques parties ou seulement la zone qui semble être la plus propice à la détection d’un lambeau. Or, l’image est de taille variable (due aux opérations de *cropping* notamment pour supprimer des informations inutiles) et le modèle doit prendre en entrée une image fixe. Pour cela, nous utilisons une *fenêtre glissante* permettant ainsi d’inférer sur toutes l’image en ne prenant que des parties fixes de cette même image. Comme nous avions des erreurs de mémoire et que nous n’avions pas réellement compris ce système au départ, nous avions redimensionné, lors des premières expérimentations, les images à une taille fixe mais les résultats n’étaient pas très concluants. Ensuite, par reconstruction, on peut retrouver un masque de segmentation en sortie qui a exactement la même taille que l’image d’entrée. De plus, lorsque l’on est en validation ou en test, on calcule les valeurs des métriques en fonction de ce qu’on obtient et de ce qu’on est censé obtenir ainsi que la la *loss* et on logge le tout sur l’interface de W&B (avec les résultats des inférences réalisées).

2.3.4 Premiers résultats sur nos données

Nous allons vous présenter les résultats que nous avons pu obtenir lors des différents lancements ayant donnés des résultats mauvais ainsi que leur analyse. Veuillez noter que le masque rouge correspond à la zone du lambeau (qu’il provienne d’une prédiction et de la vérité terrain) et le bleu correspond au reste, ce qu’on ne

souhaite pas segmenter. Cela sera important pour cette section ainsi que les sections 2.3.5 et 2.3.6.

Images de taille fixe Comme nous avons pu le dire à la section 2.3.3, au tout début nous avions décidé de redimensionner les images 3D en une image 3D de taille fixe pouvant être passée directement dans le modèle. Ce choix a été motivé par le fait que nous ne comprenions pas exactement le but de la fenêtre glissante ainsi que des problèmes de mémoire. Donc nous avions décidé de réaliser un redimensionnement vers une image de la taille d'entrée du ViT [24]. La figure 12 montre les résultats obtenus lors des premiers tests dans cette configuration. On y voit clairement que

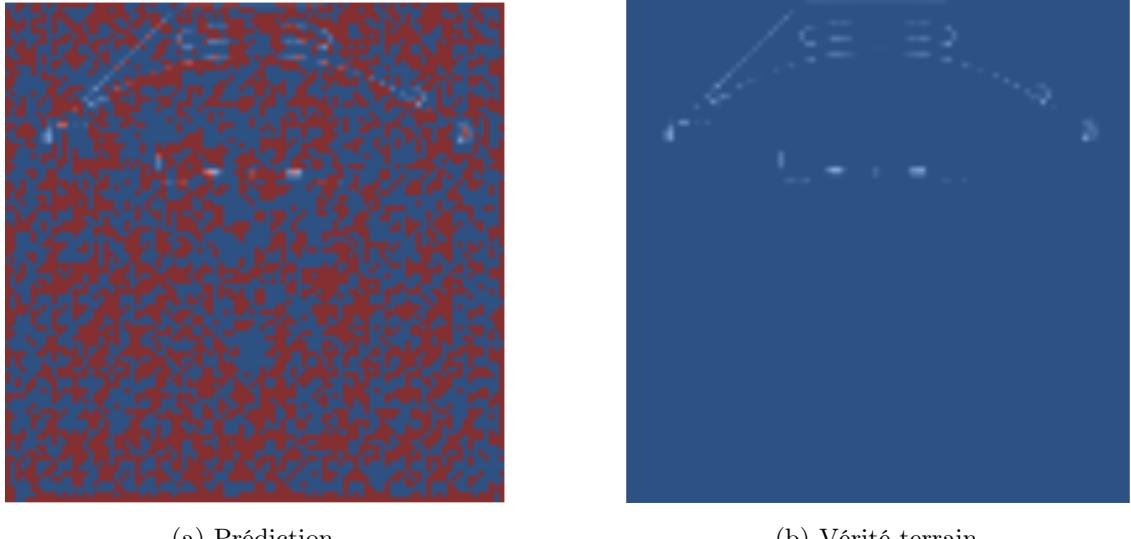
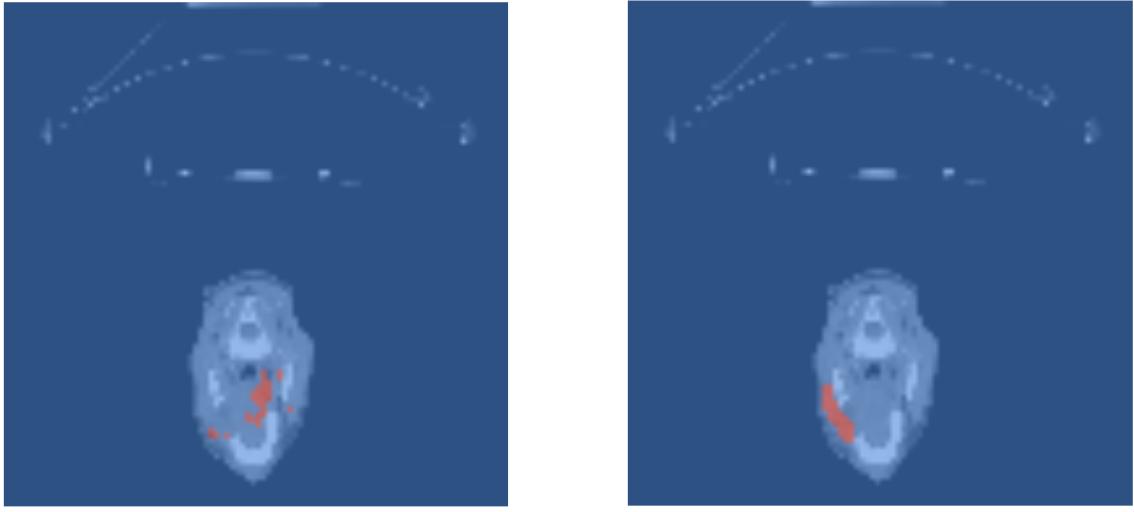


FIGURE 12 – *Slice* d'un patient du jeu de validation lors des tout premiers tests

le modèle n'a strictement rien appris. Nos encadrants nous avaient donc conseillé de revoir le réseau, mais aussi de vérifier le traitement des données, les métriques, etc. Après différentes vérifications, nous sommes parvenus à situer le problème : celui-ci provient de la fonction de coût. En effet, lorsque nous avions repris le code officiel implémentant le UNETR [3], celui-ci utilisait la *DiceCELoss* qui est censée être utilisée lorsque plusieurs classes sont en jeu au lieu de la *DiceLoss* [21] (ou la *DiceBCELoss* qui reprend le même principe que la *DiceCELoss* mais en utilisant l'entropie croisée binaire au lieu de l'entropie croisée). Ceci était dû au fait qu'ils étaient dans un problème où il y avait quatorze organes différents à segmenter contrairement à nous où nous étions dans un problème binaire (*flap* ou pas *flap*). La figure 13 présentent les résultats obtenus après cette modification. Comme on peut le voir, les résultats ne sont certes plus aléatoires, mais ne sont pas bons non plus. Notre analyse des résultats se sont plutôt portés sur la taille des images en jeu. Selon nous, si la segmentation n'était pas correcte cela pouvait venir de la façon que nous traitions les données en les redimensionnant sans prendre en compte le nombre de *slices* des images de chaque patient. Cette interprétation était en partie vraie mais nous verrons cela plus précisément dans la section 2.3.6.



(a) Prédiction

(b) Vérité terrain

FIGURE 13 – *Slice* d'un patient du jeu de validation après correction de la *loss*

Images larges En partant de cette interprétation, nous avons donc décidé de réaliser l'inférence sur des images plus larges et idéalement, ayant la même taille que l'image initiale après *cropping* sans réaliser de redimensionnement. La figure 14 présente des résultats obtenus après avoir choisi de prendre des images larges. Comme on peut aisément le voir, aucun lambeau n'est segmenté et nous avions un



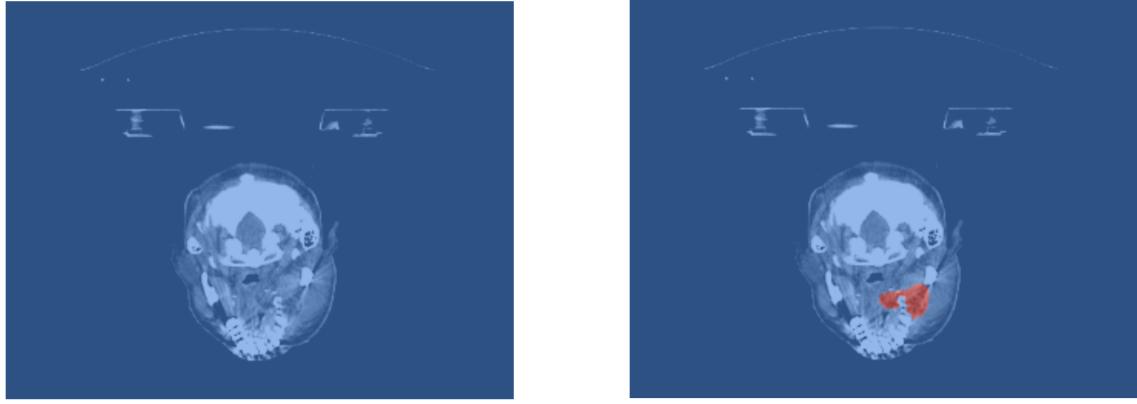
(a) Prédiction

(b) Vérité terrain

FIGURE 14 – *Slice* d'un patient du jeu de validation (version large)

Dice à 0.8181 (pour l'ensemble des classes, y compris donc l'arrière-plan). À partir de là, nous avons décidé de montrer en plus le score uniquement pour les *flaps* et nous avons donc trouvé quelque chose de très proche de 0. Le modèle ne détectait strictement rien. Nous avons donc essayé d'apporter des corrections, notamment au niveau du modèle, mais sans succès comme le montre la figure 15.

Comme le montre les différents tests, cela suggère des erreurs plus importantes et profondes dans la pipeline (que ce soit le code pour les données, celui pour finetuner ou encore les valeurs des différents hyperparamètres). Il était donc nécessaire de lui faire passer un test.



(a) Prédiction

(b) Vérité terrain

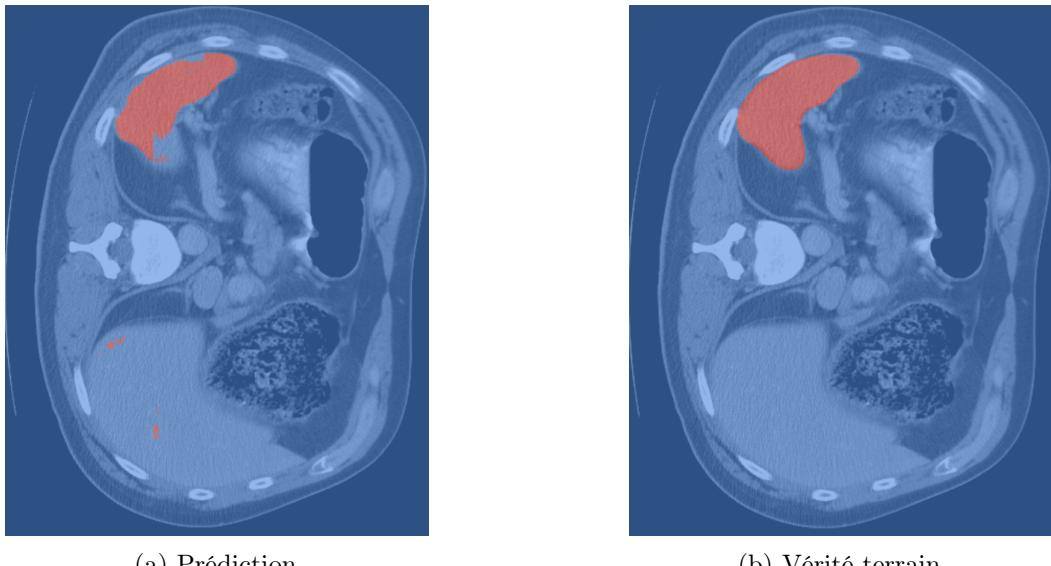
FIGURE 15 – *Slice* d'un patient du jeu de validation après corrections (version large)

2.3.5 Résultats sur les données d'origines

Suite aux résultats obtenus à la section 2.3.4, nos encadrants nous ont donc demandé de revérifier toute la pipeline de données ainsi que celle du modèle avec les données ayant servi à pré-entraîner notre modèle, soit les données provenant du MSD [4].

Nous avons donc lancé un entraînement en réalisant un *finetuning* sur uniquement une classe, celle correspondant à la rate (*spleen* en anglais) nous permettant ainsi de se mettre dans les mêmes conditions que le *finetuning* du modèle que nous réalisons (seulement deux classes : celle cible et le reste).

Résultats Les figures 16, 17 et 18 présentent des résultats de segmentation obtenus à la suite du test de notre pipeline d'entraînement sur la base de données MSD [4].



(a) Prédiction

(b) Vérité terrain

FIGURE 16 – *Slice* du premier patient du jeu de validation

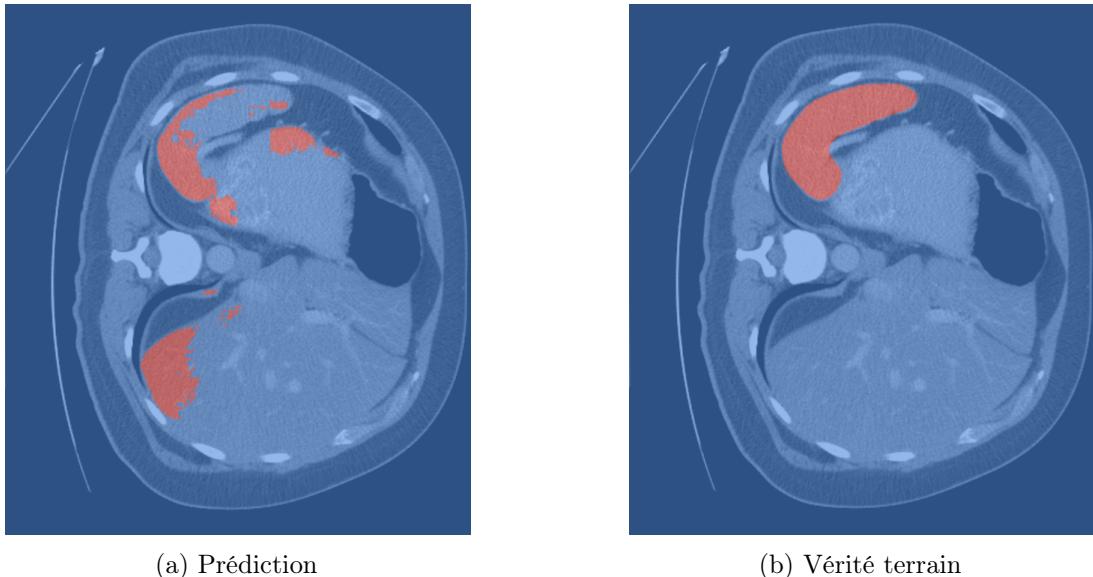


FIGURE 17 – *Slice* du deuxième patient du jeu de validation

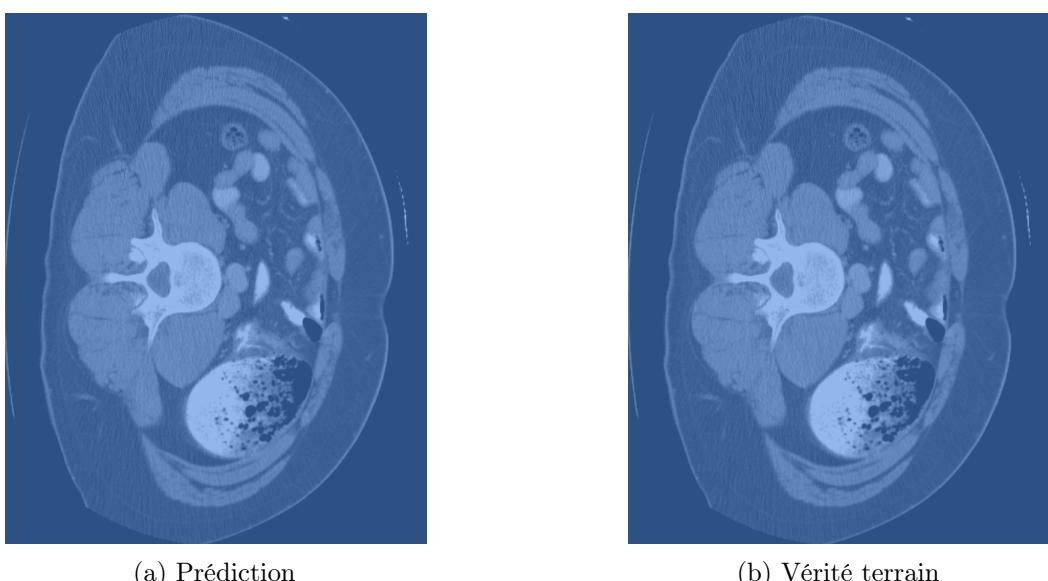


FIGURE 18 – *Slice* ne contenant pas de rate du premier patient du jeu de validation

La figure 19 présente les résultats quantitatifs à la suite du test de notre pipeline d’entraînement sur la base de données MSD [4]. On y voit la *train_loss* ainsi que la *val_loss* qui descendant (même si pour cette dernière c’est beaucoup moins marqué que la première dû à l’arrêt brutal de l’entraînement). On peut donc voir qu’il n’y a pas de sur-apprentissage. De plus, on observe bien que le *Dice* augmente tandis que la distance de Hausdorff elle diminue. Elles vont toutes deux dans le bon sens car le coefficient de Dice doit être le plus haut possible et la HD doit être la plus basse possible, comme nous en avions discuté à la section 2.2.1.

Analyse On observe bien sur les figures 16 et 17 que la rate est en partie segmentée mais des artefacts subsistent ou la rate n'est pas totalement et correctement segmentée.

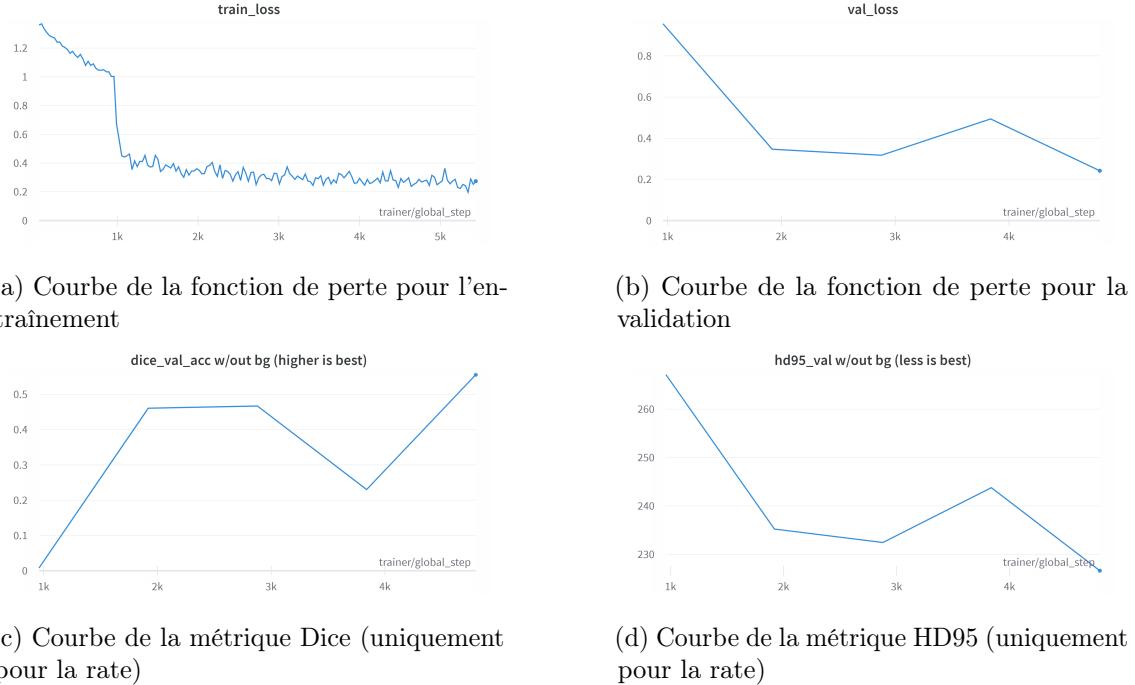


FIGURE 19 – Courbes loggées durant l’entraînement (pour les rates)

De plus, sur la figure 18, on observe qu’aux endroits où la rate n’est pas présente, le modèle réussi dans la majorité des cas à détecter que ça n’appartient pas à la classes des rates.

Ici, il est important de préciser que **l’entraînement n’a pu arriver à son terme** (jusqu’au 169ème *epoch* sur un maximum de 2000) dû à une erreur d’accès mémoire qui a pu être réglée par la suite. Néanmoins, les résultats présentés nous semblent être corrects, nous avons donc validé la pipeline d’entraînement. Ainsi, nous avons pu revenir sur nos données pour essayer de corriger les problèmes que nous avions et donc essayer de réussir l’objectif qui nous avait été fixé.

2.3.6 Derniers résultats obtenus sur les *flaps*

Nous allons donc vous présenter les résultats que nous avons obtenus après le test de la pipeline. De plus, après une réflexion que nous avons pu avoir avec nos encadrants lors d’une des réunions, nous avons modifié les valeurs des hyperparamètres liés à l’optimiseur, notamment le taux d’apprentissage ainsi que le *weight decay*. Les figures 20 et 21 présentent des résultats de segmentation de différentes *slices* de patients, le premier possédant un lambeau et le second non. On voit que dans le premier cas, le modèle segmente un lambeau dans la bonne zone mais se trouve juste pas être assez grande. De plus, lorsque l’on observe sur les résultats d’inférences, on voit clairement que le modèle réussi à déterminer la zone approximative où les lambeaux se trouvent habituellement permettant donc déjà de voir qu’il a appris quelque chose. Malheureusement, comme le montre la figure 21, il détecte un lambeau alors que dans le cas de ce patient, il n’y en a aucun.

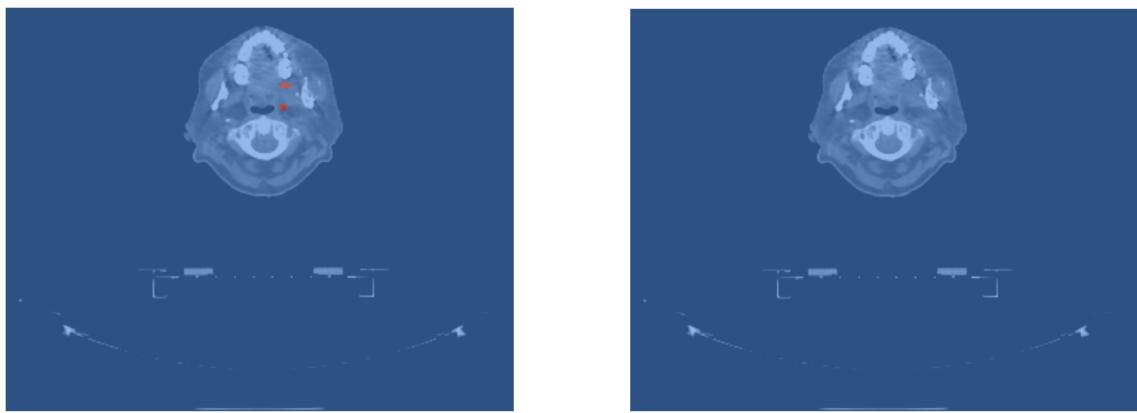
La figure 22 nous montre les courbes de la *loss* ainsi que les métriques. On y voit



(a) Prédiction

(b) Vérité terrain

FIGURE 20 – *Slice* d'un patient du jeu de validation



(a) Prédiction

(b) Vérité terrain

FIGURE 21 – *Slice* d'un second patient du jeu de validation (derniers résultats)

aucun sur-apprentissage et on voit que le *Dice* est monté jusqu'à 0,4621 et le HD95 est au plus bas à 119,15. Ces nombres sont bien meilleurs que ce que nous avions lors des précédents lancements. À noter que ces valeurs sont uniquement pour le *flap*, si l'on prend en compte toutes les classes, donc l'arrière-plan compris, on a un *Dice* à 0,902 ainsi qu'un HD95 à 21,664.

Enfin, vous trouverez dans l'annexe B, les tableaux contenant les valeurs des hyperparamètres utilisés afin d'obtenir ces résultats.

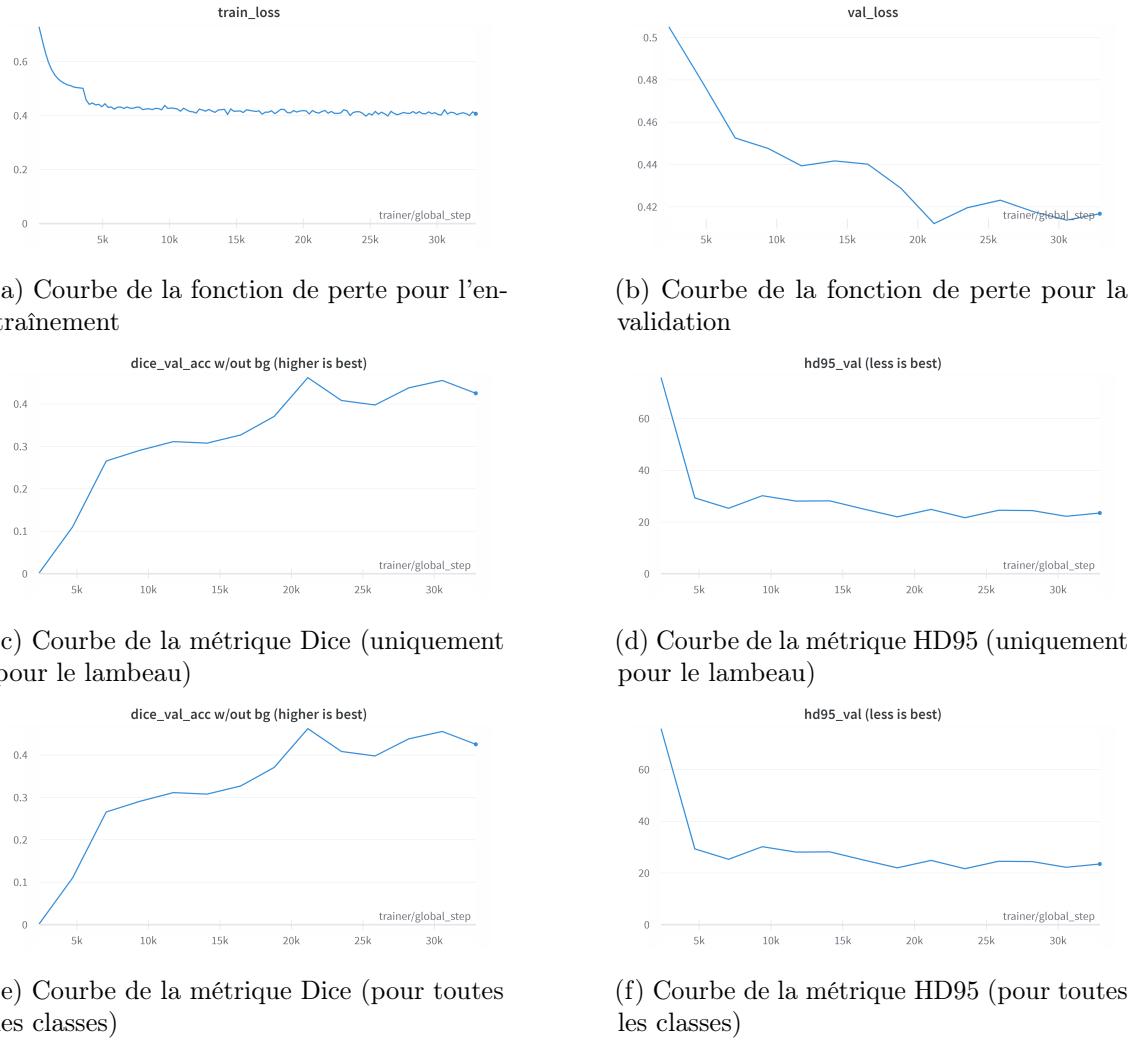


FIGURE 22 – Courbes loggées durant l’entraînement (derniers résultats)

Conclusion

Conclusion générale et apports du projet

Ce projet a été très intéressant pour tous les deux. Il nous a permis d'avoir une première approche sur des images IRM/CT et sur l'utilisation de réseaux de neurones sur une application réelle, avec tous les problèmes que cela pouvait poser. Il nous a aider à améliorer nos compétences en matière de développement de code ainsi que sur la compréhension et la résolution de problèmes. En effet, nous avons pu à la fois nous améliorer en PyTorch mais nous avons pu aussi découvrir les bibliothèques PyTorch Lightning ainsi que W&B qui sont très utilisées dans le domaine de l'apprentissage profond. Il nous a aussi permis de fortement développer et renforcer des connaissances en apprentissage profond, notamment à propos de la tâche de segmentation par IA sur laquelle nous n'avions jamais essayé d'y répondre auparavant.

Malgré les problèmes que nous avons pu avoir, nous sommes plutôt fiers d'avoir participer à ce projet, notamment pour les différentes compétences qu'il a nécessité pour sa bonne mise en oeuvre. De plus, les rendez-vous hebdomadaires nous ont aider à avancer à un bon rythme et aider à corriger les problèmes auxquels nous faisions face au fur et à mesure que nous les exposions à nos encadrants.

Ce qu'il reste à effectuer

Tout d'abord, il reste principalement à améliorer les performances du modèle afin d'obtenir une meilleure segmentation que ce que nous obtenons actuellement avec le meilleur des modèles que nous avons pu entraîner. En supplément, il aurait été intéressant d'utiliser les masques des autres régions d'intérêt qui étaient disponibles dans les RT-Struct (après un traitement des données pour filtrer). On peut imaginer que ces données supplémentaires, en utilisant les *features* extraites pour segmenter les zones autour, pourraient permettre une meilleure segmentation des lambeaux en ayant des points de repère pour pouvoir être plus précis.

Axes d'amélioration pouvant être envisagés

De nombreux axes d'amélioration peuvent être envisagés comme la modification de l'architecture en passant par une autre tâche pour pouvoir réaliser la segmentation ou encore via l'utilisation d'un autre domaine de l'intelligence artificielle (IA) pouvant aider à interpréter les résultats d'inférence obtenus.

Architecture Il est possible de modifier l'architecture du réseau utilisé. Nous nous sommes basés sur un UNETR [3] car utilisant un Transformer [23] comme encodeur, nous espérions obtenir des résultats (les mauvais résultats ne remettent pas forcément en doute l'approche eu, il est nécessaire de réaliser d'autres expérimentations pour mesurer la viabilité réelle de cette approche sur des machines plus puissantes afin d'extraire plus de patchs que nous n'avons pu le faire). D'autres approches se basent aussi sur cette technologie comme le SwinUNETR [31] utilisant quant à lui un

SwinTransformer [32] au lieu d'un ViT [24]. On peut aussi noter l'apparition de UNETR++ [33] qui semble être un UNet++ [14] utilisant un UNETR [3] comme base. Ces deux nouvelles méthodes semblent obtenir de meilleurs résultats que le UNETR [3] tout en possédant moins de paramètres. On peut imaginer que cela pourrait être intéressant pour des machines n'ayant que très peu de mémoire graphique dédiée.

On peut aussi imaginer utiliser une nouvelle approche nommée FlexViT [34] permettant d'avoir un ViT [24] avec une taille de patchs variants. Celui-ci pourrait remplacer le ViT [24] que nous avons comme encodeur dans le UNETR [3].

Une dernière approche pouvant être abordée est celle de MedSegDiff [35]. Cette approche se base sur les modèles de diffusion comme Stable Diffusion [36], Dall-e 2 [37] ou bien encore [MidJourney](#) pour réaliser de la segmentation d'images médicales. Ces approches ont montré leur supériorité dans la génération automatique d'image, notamment guidée par du texte. L'idée ici est d'avoir le masque faisant partie du processus de débruitage et avoir l'image IRM/CT comme le guide de la segmentation.

XAI Le domaine médical étant un domaine applicatif ayant besoin d'explications afin de garantir une bonne prise de décision, il est primordial de se poser la question de l'interprétation du modèle que nous avons. En effet, notre modèle prend certaines décisions sur lesquelles il se base pour réaliser la segmentation d'un lambeau. Il se peut que le modèle soit biaisé et prend des décisions car il a observé qu'en majorité du temps, les *flaps* sont proches de la mâchoire et donc il y a une forte probabilité qu'il y est un lambeau. Certes cela pourrait aider à détecter le lambeau, mais s'il se base uniquement sur cette partie du corps, cela est problématique car il ne réussit pas vraiment à comprendre ce qu'est un lambeau. Comme le modèle est basé sur des réseaux de neurones profonds, il est nécessaire d'avoir des outils permettant d'interpréter les modèles. C'est ici qu'entre en jeu l'*eXplainable AI* (XAI) ou IA explicable en français. C'est un domaine dans lequel on cherche à obtenir une capacité d'interprétation de modèles d'apprentissage automatique et notamment pour les modèles d'apprentissage profond. Permettre l'interprétation peut aider à corriger le modèle et le rendre plus robuste. Pour cela, différentes bibliothèques existent mais pour [PyTorch](#), celle qui nous semble la plus intéressante est [Captum](#) car elle s'intègrerait très facilement dans le code déjà produit. Il existe un tutoriel à [cet URL](#). Enfin, différentes recherches ont lieu dans le XAI et dans les domaines applicatifs comme le médical, notamment avec le GradXcepUNet [38] mêlant U-Net [1] et XAI pour l'interprétation de segmentation d'images médicales.

Références

- [1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net : Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [2] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [3] Ali Hatamizadeh, Yucheng Tang, Vishwesh Nath, Dong Yang, Andriy Myronenko, Bennett Landman, Holger R Roth, and Daguang Xu. Unetr : Transformers for 3d medical image segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 574–584, 2022.
- [4] Michela Antonelli, Annika Reinke, Spyridon Bakas, Keyvan Farahani, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, Olaf Ronneberger, Ronald M Summers, et al. The medical segmentation decathlon. *Nature communications*, 13(1) :4128, 2022.
- [5] BA Landman, Z Xu, JE Igelsias, M Styner, TR Langerak, and A Klein. Miccai multi-atlas labeling beyond the cranial vault - workshop and challenge, 2015.
- [6] Patrick Bilic, Patrick Christ, Hongwei Bran Li, Eugene Vorontsov, Avi Ben-Cohen, Georgios Kaassis, Adi Szeskin, Colin Jacobs, Gabriel Efrain Humpire Mamani, Gabriel Chartrand, et al. The liver tumor segmentation benchmark (lits). *Medical Image Analysis*, 84 :102680, 2023.
- [7] Zoé Lambert, Caroline Petitjean, Bernard Dubray, and Su Kuan. Segthor : Segmentation of thoracic organs at risk in ct images. In *2020 Tenth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6. IEEE, 2020.
- [8] C Weight, N Papanikopoulos, A Kalapara, and N Heller. Kidney tumor segmentation. <https://kits19.grand-challenge.org/>, 2019. accessed : 2019-07-08.
- [9] A Emre Kavur, N Sinem Gezer, Mustafa Barış, Sinem Aslan, Pierre-Henri Conze, Vladimir Groza, Duc Duy Pham, Soumick Chatterjee, Philipp Ernst, Savaş Özkan, et al. Chaos challenge-combined (ct-mr) healthy abdominal organ segmentation. *Medical Image Analysis*, 69 :101950, 2021.
- [10] National Cancer Institute. The cancer imaging archive. <https://www.cancerimagingarchive.net/>.
- [11] Dong Nie, Yaozong Gao, Li Wang, and Dinggang Shen. Asdnet : attention based semi-supervised deep networks for medical image segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 370–378. Springer, 2018.
- [12] Hao Zheng, Jun Han, Hongxiao Wang, Lin Yang, Zhuo Zhao, Chaoli Wang, and Danny Z Chen. Hierarchical self-supervised learning for medical image segmentation based on multi-domain data aggregation. In *International Conference on*

Medical Image Computing and Computer-Assisted Intervention, pages 622–632. Springer, 2021.

- [13] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net : learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.
- [14] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++ : A nested u-net architecture for medical image segmentation. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 3–11. Springer, 2018.
- [15] Fabian Isensee, Jens Petersen, Andre Klein, David Zimmerer, Paul F Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Koehler, Tobias Norajitra, Sebastian Wirkert, et al. nnu-net : Self-adapting framework for u-net-based medical image segmentation. *arXiv preprint arXiv:1809.10486*, 2018.
- [16] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet : A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12) :2481–2495, 2017.
- [17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [18] Jonathan Zopes, Moritz Platscher, Silvio Paganucci, and Christian Federau. Multi-modal segmentation of 3d brain scans using neural networks. *Frontiers in Neurology*, 12 :1167, 2021.
- [19] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Ruckridge. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9) :850–863, 1993.
- [20] Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–7. IEEE, 2020.
- [21] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net : Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.
- [22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco : Common objects in context. In *Computer Vision–ECCV 2014 : 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.
- [23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [24] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg

- Heigold, Sylvain Gelly, et al. An image is worth 16x16 words : Transformers for image recognition at scale. *arXiv preprint arXiv :2010.11929*, 2020.
- [25] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net : Learning where to look for the pancreas. *arXiv preprint arXiv :1804.03999*, 2018.
 - [26] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
 - [27] Mina Amiri, Rupert Brooks, and Hassan Rivaz. Fine tuning u-net for ultrasound image segmentation : which layers ? In *Domain Adaptation and Representation Transfer and Medical Image Learning with Less Labels and Imperfect Data : First MICCAI Workshop, DART 2019, and First International Workshop, MIL3ID 2019, Shenzhen, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13 and 17, 2019, Proceedings 1*, pages 235–242. Springer, 2019.
 - [28] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks ? *Advances in neural information processing systems*, 27, 2014.
 - [29] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv :1710.03740*, 2017.
 - [30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers : Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
 - [31] Ali Hatamizadeh, Vishwesh Nath, Yucheng Tang, Dong Yang, Holger R Roth, and Daguang Xu. Swin unetr : Swin transformers for semantic segmentation of brain tumors in mri images. In *International MICCAI Brainlesion Workshop*, pages 272–284. Springer, 2022.
 - [32] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer : Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.
 - [33] Abdelrahman Shaker, Muhammad Maaz, Hanoona Rasheed, Salman Khan, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Unetr++ : Delving into efficient and accurate 3d medical image segmentation. *arXiv preprint arXiv :2212.04497*, 2022.
 - [34] Lucas Beyer, Pavel Izmailov, Alexander Kolesnikov, Mathilde Caron, Simon Kornblith, Xiaohua Zhai, Matthias Minderer, Michael Tschannen, Ibrahim Alabdulmohsin, and Filip Pavetic. FlexiViT : One Model for All Patch Sizes. *arXiv e-prints*, page arXiv :2212.08013, December 2022.
 - [35] Junde Wu, Huihui Fang, Yu Zhang, Yehui Yang, and Yanwu Xu. Medsegdiff : Medical image segmentation with diffusion probabilistic model. *arXiv preprint arXiv :2211.00611*, 2022.

- [36] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.
- [37] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv :2204.06125*, 2022.
- [38] Amandeep Kaur, Guanfang Dong, and Anup Basu. Gradxcepunet : Explainable ai based medical image segmentation. In Stefano Berretti and Guan-Ming Su, editors, *Smart Multimedia*, pages 174–188, Cham, 2022. Springer International Publishing.

Annexes

A Recherches sur l'état de l'art

Voici [un lien](#) vers un document en lecture seule montrant les différentes recherches que nous avons pu réaliser sur l'état de l'art. Ce document a principalement été constitué au début du projet mais des informations ont été ajoutées au fur et à mesure, notamment en suivant les nouveautés dans le monde de la recherche lorsque nous étions sur le projet.

B Hyperparamètres du meilleur modèle

Les tables 2 et 3 présentent les hyperparamètres utilisés pour obtenir le meilleur modèle que nous ayons finetuné (voir section 2.3.6).

Nom de l'hyperparamètre	Valeur
train_batch_size	1
val_batch_size	1
workers	4
voxel_space	(1.5, 1.5, 2.0)
a_min	-200.0
a_max	300.0
b_min	0.0
b_max	1.0
clip	True
crop_bed_max_number_of_rows_to_remove	0
crop_bed_max_number_of_cols_to_remove	0
crop_bed_min_spatial_size	(300, -1, -1)
enable_fgbg2indices_feature	False
pos	1.0
neg	1.0
num_samples	2
roi_size	(96, 96, 96)
random_flip_prob	0.2
random_90_deg_rotation_prob	0.2
random_intensity_scale_prob	0.1
random_intensity_shift_prob	0.1
val_resize	(-1, -1, 250)

TABLE 2 – Tableau des hyperparamètre utilisés pour le *preprocessing* des données

Nom de l'hyperparamètre	Valeur
optimizer	AdamW
optimizer.learning_rate	1e-4
optimizer.weight_decay	0
lr_scheduler	CosineAnnealingLR
lr_scheduler.T_max	2000
in_channels	1
out_channels	14
roi_size	(96, 96, 96)
new_out_channels	2
number_of_blocks_to_tune	1
feature_size	16
hidden_size	768
mlp_dim	3072
num_heads	12
pos_embed	perceptron
norm_name	instance
conv_block	True
res_block	True
dropout_rate	0.0
infer_overlap	0.5
max_epochs	2000
smooth_dr	1e-6
smooth_nr	0.0
sw_batch_size	4
use_bce_loss_when_binary_problem	False
unfreeze_backbone_at_epoch	20

TABLE 3 – Tableau des hyperparamètres du modèle