

Projet de programmation

L3 Informatique

Segmentation automatique de l'aorte
dans des volumes CT-Scan sans produit de contraste

GROUPE N

HARCHA Ibrahim
LAZRAK Ilona
MALLET Samuel
SABATER ROJAS Rosa



Encadrante : HAMMAMI Houda

Faculté des Sciences
Université de Montpellier, France
12 mai 2023

Abstract

Performing non-contrasted aorta segmentation in CT-Scan volumes presents a complex challenge due to the intricate anatomy and varied visibility of the aorta. This paper proposes a three-stage method leveraging convolutional neural networks (CNNs) for robust aorta segmentation. The process initiates with the preprocessing of the CT-Scan volume, which includes resizing and normalization. Subsequently, a trained CNN is employed to perform slice-by-slice aorta segmentation in the preprocessed volume. The final stage of this procedure is the reconstruction of a 3D model from the collected masks obtained in the preceding step. This automated segmentation approach has been evaluated using a dataset comprising 900 2D images sourced from 11 patients.

Restriction

Implementation of convolutional neural networks within a project timeline spanning 4 months

Keywords

Non contrasted CT-Scan
Aorta segmentation on 2D images of CT-Scans
.nrrd and .nii format
3DSlicer
Python and Tensorflow
Artificial intelligence
CNNs and U-NET
Google Colab

Resources

GitHub : <https://github.com/SuperMuel/PP2-aorte-segmentation/tree/main>
Google Drive : https://drive.google.com/drive/folders/14P8u1HGohRuw3l6KLMvmYadaqiedoqXa?usp=share_link

Remerciements

Nous tenons à exprimer notre gratitude :

Aux professeurs d'université qui nous ont accompagnés tout au long de la licence, témoignant de notre respect et de notre reconnaissance pour leur engagement dans notre formation.

À Madame Hammami pour avoir proposé ce sujet passionnant et enrichissant, pour ses conseils sur les sujets à aborder et les idées à explorer.

À Xiao Wei Xu pour nous avoir donné accès aux données collectées dans le cadre de son projet de recherche. Bien que nous n'ayons pas utilisé ces données dans notre analyse finale, car nous nous sommes concentrés sur des scans d'aortes saines, nous apprécions sa générosité et sa collaboration. Ses données pourraient être utilisées pour approfondir le projet.

Aux chercheurs et autres internautes qui partagent leurs travaux sur internet. Grâce à leur générosité, nous avons pu accéder à des informations précieuses et fiables concernant les réseaux de neurones.

Enfin, nous tenons à exprimer notre profonde gratitude à notre famille et à toutes les personnes qui nous ont soutenus tout au long de notre parcours universitaire. Ce travail final de licence est le fruit de nos efforts, mais aussi de votre encouragement, de votre confiance et de votre générosité. Nous dédions ce mémoire en témoignage de notre reconnaissance et de notre affection.

Table des matières

1 Présentation du sujet	1
1.1 Introduction aux anévrismes	1
1.2 Visualisation de l'aorte	2
1.2.1 Echographie	2
1.2.2 Radiographie (rayons X)	2
1.2.3 IRM	2
1.3 Tomodensitométrie (scanner)	2
1.4 Les produits de contraste dans l'imagerie par scanner	3
1.4.1 Avantages du produit de contraste	3
1.4.2 Inconvénients du produit de contraste	3
1.5 L'intérêt de notre projet	3
1.5.1 Segmentation automatique de l'aorte	3
1.5.2 L'application des réseaux de neurones convolutifs	4
1.6 Choix et structure du projet	4
2 Technologies utilisées	5
2.1 Langage de programmation : Python	5
2.2 Ressources informatiques : Google Colab	5
2.3 Stratégies de segmentation	6
2.4 Réseaux de neurones	6
2.4.1 U-Net	7
2.4.2 Présentation des slices utilisées dans l'architecture U-Net	8
3 Structures des données	11
3.1 Le format .nrrd	11
3.2 Introduction à la normalisation	11
3.2.1 Normalisation des images CT-Scan	12
3.2.2 Implémentation	13
3.3 Extraction de slices	14
3.3.1 Choix de traiter uniquement l'axe Z	14
3.3.2 Nombre de slices par volume	14
3.3.3 Gestion des slices	15
3.4 Organisation des données pour l'entraînement du modèle U-Net	15
4 Algorithmes	17
4.1 Implémentation du modèle U-Net	17
4.2 Reconstruction 3D	18
4.3 Statistiques	18
5 Gestion du Projet - Groupe N de Notion	19
5.1 Présentation générale de Notion	19
5.2 Collaboration et intégrations	19
5.3 Organisation des données de CT-Scans	20
5.4 Timeline et organisation des tâches	20
6 Bilan et analyse des résultats	22
6.1 Problème rencontré lors de l'exécution du programme testing.py	22
6.2 Programmes finaux : Google Colab	22
6.2.1 Plot training & validation loss values	23
6.2.2 Plot training & validation accuracy values	23
6.3 Reconstruction 3D de l'aorte	24
7 Conclusions	27
8 Bibliographie	28

9 Annexes	31
9.1 3DSlicer	31
9.2 Premier programme : dataPreparation.py	32
9.2.1 Explication et utilisation	32
9.3 Deuxième programme : testing.py	33
9.3.1 Définitions des paramètres	34
9.3.2 Modèle	35
9.3.3 Plots des images CT-Scan	36
9.3.4 Prédictions	40
9.3.5 Résultats	46
9.3.6 Overfit et curiosités	47
9.4 PROGRAMMES FINAUX : Google Colab	49
9.4.1 Extrait de code de predict_and_reconstruct.ipynb - Explication des seuils	49
9.4.2 Reconstruction 3D - D15.nrrd	50
9.4.3 Reconstruction 3D - R17.nrrd	53
9.4.4 Reconstruction 3D - K17.nrrd	56
9.4.5 Reconstruction 3D - Exemple qui n'a pas fonctionné R16.nrrd	59

1 Présentation du sujet

1.1 Introduction aux anévrismes

Il existe différents types d'anévrismes de l'aorte selon leur localisation. Les anévrismes de l'aorte thoracique (AAT) sont ceux qui se situent au-dessus du diaphragme, tandis que les anévrismes de l'aorte abdominale (AAA) se situent généralement en dessous des artères rénales (infrarénal) et peuvent parfois les atteindre ou toucher les artères iliaques. Une aorte abdominale mesure de 2,5 à 3,5 cm de diamètre en moyenne. Un diamètre aortique supérieur ou égal à 3 cm définit un AAA. Le diamètre normal de l'aorte thoracique ascendante est de 3 à 4 cm, celui de l'arc aortique est de 2,5 à 3,5 cm et celui de l'aorte thoracique descendante est de 2 à 3 cm. Ainsi, à partir de ces valeurs, nous pouvons estimer qu'un anévrisme thoracique de l'aorte est présent lorsque le diamètre dépasse environ 4,5 à 6 cm pour l'aorte thoracique ascendante, 3,75 à 5,25 cm pour l'arc aortique et 3 à 4,5 cm pour l'aorte thoracique descendante.

L'histoire du traitement des anévrismes de l'aorte peut être divisée en trois étapes principales :

La première étape remonte au XVI^e siècle,

lorsque Andrea Vesalius a identifié pour la première fois un AAA lors d'une autopsie. En 1817, Astley Cooper a réalisé la première ligature d'une artère iliaque externe rompue, mais il a fallu attendre 1923 pour que Rudolf Matas réussisse la première ligature d'un AAA non rompu. Avant cela, tous les AAA rompus étaient mortels. La ligature de l'aorte a été suivie par d'autres techniques invasives (câblage, coagulation, enveloppement, etc.), mais avec une mortalité très élevée.

La deuxième étape commence à la fin du XIX^e siècle

avec Rudolf Matas, qui a introduit le concept d'endoanévrisme, une technique qui permet de préserver le flux sanguin vers les membres inférieurs. Le premier succès majeur dans la chirurgie des AAA est attribué à Charles Dubost, qui a rapporté en 1951 la première résection d'un AAA et le remplacement de l'aorte par une homogreff. Quelques années plus tard (1953), Blakemore et Voorhees ont implanté la première prothèse synthétique dans un AAA rompu, qui a été popularisée par Rob (1954) et améliorée par Creech (1966) avec son concept de reconstruction intrasacculaire.

La troisième étape remonte au début du XX^e siècle,

lorsque les premières radiographies du thorax ont été réalisées. Avec le développement de l'échographie dans les années 1950, une nouvelle technique non invasive et plus sensible a été introduite pour examiner l'aorte thoracique. Dans les années 1970, la tomodensitométrie (TDM) a apporté une autre avancée dans sa détection. Juan Parodi a apporté sa contribution révolutionnaire appelée EndoVascular Aneurysm Repair (EVAR) en 1991, qui a changé la chirurgie des AAA pour la deuxième fois en moins de 50 ans et cette technique a été adaptée tant pour le traitement des AAA que des AAT.

Les articles [1, 2, 3], respectivement de Santé Publique de France, de Francisco S. Lozano Sánchez, professeur et chef de service de l'Hôpital Universitaire de Salamanque, Espagne, et la version du Manuel MSD pour les professionnels de la santé indiquent que la rupture des anévrismes thoracoabdominaux entraîne un taux de mortalité global compris entre 80% et 97%. Parmi les patients qui parviennent à l'hôpital en vie, la mortalité postopératoire d'un AAA est de 40% ou 50%, et la survie du patient qui présente un AAT non traité est de 65% à 1 an et 20% à 5 ans. La rupture d'un anévrisme entraîne donc une hémorragie interne potentiellement fatale.

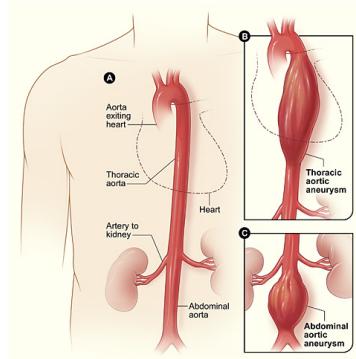


FIGURE 1 – A. Aorte normale B. Anévrisme aortique thoracique (situé derrière le cœur) C. Anévrisme aortique abdominal, situé sous les artères rénales. [1]

1.2 Visualisation de l'aorte

La visualisation de l'aorte, la plus grande artère du corps humain, est une tâche fondamentale en imagerie médicale. Les médecins utilisent une variété de techniques pour l'observer et diagnostiquer les pathologies potentielles. Dans cette section, nous discuterons de trois méthodes couramment utilisées pour visualiser l'aorte : l'échographie, la radiographie et l'imagerie par résonance magnétique (IRM).

1.2.1 Echographie

L'échographie est une méthode d'imagerie médicale qui utilise des ondes sonores pour produire des images de l'intérieur du corps. Elle est non-invasive et ne nécessite pas l'utilisation de radiations. L'échographie de l'aorte peut être utile pour détecter des anévrismes ou d'autres anomalies structurelles. Cependant, la qualité des images produites par l'échographie peut être inférieure à celle d'autres méthodes d'imagerie, et elle peut être limitée par des facteurs tels que l'obésité du patient ou la présence de gaz intestinaux.

1.2.2 Radiographie (rayons X)

La radiographie est une autre méthode couramment utilisée pour visualiser l'aorte. Elle utilise des radiations pour produire des images des structures internes du corps. La radiographie peut révéler des anomalies de l'aorte, comme un élargissement ou une calcification. Cependant, contrairement à l'échographie, elle expose le patient à des radiations. De plus, elle ne fournit pas une image aussi détaillée de l'aorte que d'autres méthodes d'imagerie, comme le scanner ou l'IRM.

1.2.3 IRM

L'Imagerie par Résonance Magnétique (IRM) utilise un champ magnétique et des ondes radio pour produire des images détaillées des organes et des tissus internes du corps. L'IRM peut fournir des images très détaillées de l'aorte, ce qui peut être utile pour diagnostiquer des conditions telles que les anévrismes de l'aorte, les dissections de l'aorte, et d'autres anomalies. L'IRM ne nécessite pas l'utilisation de radiations, ce qui est en fait une option plus sûre pour certains patients. Cependant, l'IRM est une procédure plus longue que la radiographie ou le scanner, et elle peut ne pas être appropriée pour les patients qui ont certaines conditions, comme les implants métalliques, ou pour ceux qui sont claustrophobes.

1.3 Tomodensitométrie (scanner)



FIGURE 2 – Tomodensitomètre [2]

La tomodensitométrie, également connue sous le nom de scanner ou de scanographie, est une technique d'imagerie médicale qui utilise des rayons X pour obtenir des images détaillées de l'intérieur du corps sous forme de tranches ou de coupes transversales. Ces images peuvent ensuite être utilisées pour créer une image tridimensionnelle ou volume de la zone étudiée.

Le terme tomodensitométrie se décompose en *tomographie*, qui signifie imagerie par sections, et *densitométrie*, qui renvoie à la mesure de la densité des tissus.

Durant un examen de tomodensitométrie, le patient est allongé sur une table qui se déplace lentement à travers un grand anneau, tandis qu'une source de rayons X tourne autour du patient, produisant de nombreuses images depuis différents angles. Ces images sont ensuite traitées par un ordinateur pour produire une image finale qui peut montrer une section précise du corps en détail.

La tomodensitométrie peut être utilisée pour visualiser pratiquement toutes les parties du corps et est utile pour diagnostiquer diverses conditions.

tiquer les maladies, guider les interventions médicales, planifier la radiothérapie, et surveiller l'efficacité des traitements.

En anglais, le terme utilisé est CT-Scan qui est l'acronyme de *Computed Tomography*, se traduisant en français par *tomographie assistée par ordinateur*.

1.4 Les produits de contraste dans l'imagerie par scanner

Les produits de contraste sont des substances souvent utilisées en imagerie médicale pour améliorer la visibilité des structures ou des fluides dans le corps. Dans le contexte de l'imagerie par scanner, ils sont couramment utilisés pour augmenter le contraste entre l'aorte et les tissus environnants, ce qui facilite sa visualisation et sa segmentation.

1.4.1 Avantages du produit de contraste

Le principal avantage de l'utilisation d'un produit de contraste est qu'il permet d'améliorer significativement la qualité de l'image en augmentant le contraste entre les différentes structures anatomiques. Ceci est particulièrement utile dans le cas de l'aorte, car elle est entourée de nombreux autres organes et tissus qui peuvent avoir des densités similaires sur les images de scanner non contrastées. En augmentant le contraste, le produit de contraste facilite la détection et la segmentation de l'aorte, ce qui peut aider à améliorer la précision du diagnostic et du traitement.

1.4.2 Inconvénients du produit de contraste

Malgré ses avantages, l'utilisation de produits de contraste n'est pas sans inconvénients. Tout d'abord, ils peuvent provoquer des réactions allergiques, bien que ces cas soient relativement rares. De plus, ils peuvent être néfastes pour les patients ayant une fonction rénale réduite, car ils peuvent aggraver une insuffisance rénale préexistante. Enfin, l'utilisation de produits de contraste augmente le coût de l'examen et prolonge sa durée.

De plus, tous les patients n'ont pas accès à des examens avec produit de contraste pour des raisons variées, comme les contre-indications médicales ou les contraintes économiques.

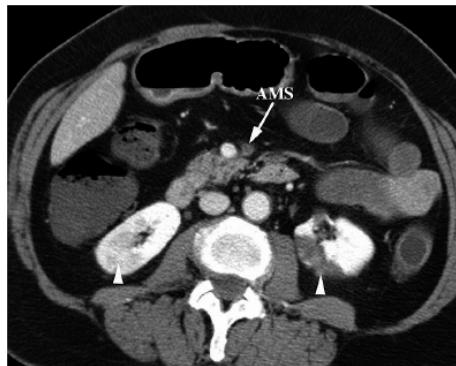


FIGURE 3 – CT-Scan avec contraste [3]

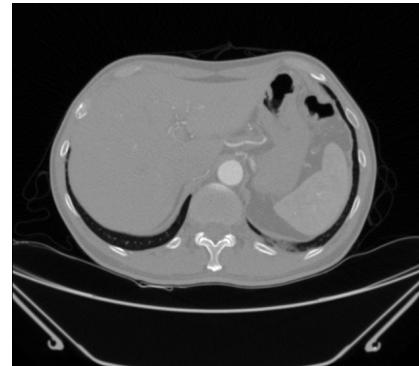


FIGURE 4 – CT-Scan sans contraste D4_im [4]

1.5 L'intérêt de notre projet

1.5.1 Segmentation automatique de l'aorte

L'aorte joue un rôle crucial dans la circulation sanguine et sa pathologie, telle que l'anévrisme de l'aorte abdominale (AAA) ou thoracique (AAT), peut avoir des conséquences fatales. La plupart des AAA et AAT restent silencieux jusqu'à leur rupture, bien que certains soient découverts lors d'examens d'imagerie réalisés pour d'autres raisons médicales. Un dépistage efficace peut réduire la mortalité de

53% parmi les patients asymptomatiques. Ainsi, une technologie capable de segmenter l'aorte de manière rapide et précise revêt une importance capitale.

En outre, notre projet, en se concentrant sur la segmentation de l'aorte dans les volumes de scanner non contrastés, permettrait de réduire l'utilisation de produits de contraste, minimisant ainsi les risques pour les patients et rendant l'examen plus accessible, en particulier pour ceux qui ont des contre-indications à l'utilisation de ces produits. Cela éliminerait également le besoin d'une injection pour le patient, réduisant le coût et la durée de l'examen.

1.5.2 L'application des réseaux de neurones convolutifs

L'application de réseaux de neurones convolutifs (CNN) à l'imagerie médicale, et en particulier à la segmentation automatique de structures anatomiques comme l'aorte, est une approche prometteuse. Les CNN ont la capacité de s'adapter aux variations de forme, de taille, de contraste et de bruit dans les images. En conséquence, ils peuvent surpasser les méthodes classiques basées sur des règles ou des caractéristiques manuelles en termes de performances.

L'application de CNN à la segmentation de l'aorte peut aider à améliorer la qualité, la précision et l'efficacité de l'imagerie médicale. Cela peut se traduire par une réduction du temps de traitement, une diminution du bruit, des artefacts et des erreurs humaines. Ces améliorations ouvrent des perspectives nouvelles et passionnantes pour la médecine du futur.

1.6 Choix et structure du projet

Ce rapport présente le projet que nous avons réalisé dans le cadre de l'UE *Projet de programmation 2* de la Licence 3 Informatique de l'Université de Montpellier, France. Ce sujet nous a attirés car il nous permettait d'acquérir des connaissances dans un domaine que nous n'avions jamais abordé auparavant, à savoir les réseaux de neurones, et aussi car il correspondait à nos centres d'intérêt personnels tels que la médecine. Ce projet nous a également permis de mettre en pratique les techniques acquises tout au long de la licence Informatique en collaborant au sein de notre groupe, où nous avons pu bénéficier des idées, des énergies, des forces, des connaissances et des compétences de chacun. Cette synergie a facilité le déroulement des tâches et amélioré l'efficacité et la qualité des résultats finaux du projet.

Pour ce faire, nous avons suivi les recommandations de madame Hammami, notre encadrante, et nous avons consulté des sources pertinentes, notamment des sources libres disponibles sur internet tels que :

- Tutoriels de Deep Learning trouvés sur www.kaggle.com [4, 5].
- Première utilisation des bibliothèques *Tensorflow* [6] et *OpenCV* [7].
- Logiciels de segmentation d'images : Seg3D [8], ITK-SNAP [9] et celui que nous avons utilisé, 3DSlicer [10].
- Lecture de l'article *Modelo mecánico para la detección de la pared externa de la aorta en imágenes de rayos X mediante sistemas de partículas para el modelado de AAA* [11] : dans ce document, nous avons identifié la plupart des mots-clés pertinents pour le projet.
- Première approche à une segmentation de l'aorte [12].

Finalement, après avoir rassemblé toutes ces informations et exemples, **les missions de ce projet étaient les suivantes :**

1. Collecter des données (CT scans)
2. État de l'art : choisir un réseau de neurones adapté en fonction du type de données, du temps de traitement, de la précision des résultats, etc.
3. Effectuer un prétraitement des images si besoin (découpage, redimensionnement, filtres, etc.)
4. Implémenter le réseau de neurones
5. Présenter les résultats obtenus
6. Améliorer les résultats en affinant les hyperparamètres

Nous exposons dans ce document la méthodologie que nous avons employée pour effectuer la segmentation automatique de l'aorte, ainsi que les résultats obtenus et les conclusions que nous en avons tirées.

2 Technologies utilisées

2.1 Langage de programmation : Python

Dans le cadre de ce projet, nous avons opté pour le langage de programmation **Python**, largement répandu dans le domaine de l'intelligence artificielle. Python est un langage interprété qui facilite le prototypage rapide, en partie grâce à son système de vérification de types flexible et moins strict que d'autres langages. De nombreuses bibliothèques performantes sont disponibles pour Python, telles que TensorFlow et PyTorch, qui sont écrites en C, garantissant ainsi une exécution rapide. Par ailleurs, Python est fréquemment utilisé dans le domaine du Machine Learning, ce qui se traduit par une documentation abondante et de nombreuses ressources en ligne. Bien que d'autres langages de programmation, tels que **JavaScript ou R**, puissent également être utilisés, leurs bibliothèques ne bénéficient généralement pas d'une documentation aussi complète que celles de Python.

TensorFlow est une bibliothèque open-source développée par Google, conçue pour faciliter la création et l'entraînement de modèles d'apprentissage profond. Nous avons choisi d'utiliser TensorFlow en raison de sa popularité, de la qualité de sa documentation et de sa capacité à évoluer en fonction des besoins [13].

2.2 Ressources informatiques : Google Colab

La complexité du modèle, la taille des données et la résolution des images sont autant de facteurs qui influencent les ressources informatiques requises. Les modèles d'IA nécessitent une quantité importante de mémoire vive (RAM) et de puissance de calcul pour l'entraînement et la réalisation d'inférences (prédiction d'étiquettes pour de nouvelles données). Par exemple, entraîner un modèle de segmentation d'image 3D sur 50 patients peut prendre plusieurs heures ou jours sur un ordinateur portable standard, tandis que l'utilisation des ressources de **Google Colab** peut réduire ce temps à quelques minutes.

Compte tenu des exigences en matière de ressources informatiques, nos ordinateurs portables ne sont pas adaptés pour gérer de telles tâches. C'est pourquoi nous avons choisi d'utiliser Google Colab, une plateforme cloud offrant des ressources de calcul conséquentes, y compris des **GPU**¹. Les GPU sont particulièrement adaptés pour traiter de nombreux calculs en parallèle, accélérant ainsi considérablement les performances de notre modèle. Ils sont particulièrement performants pour les opérations mathématiques telles que la multiplication de matrices, essentielles pour réaliser des calculs de convolution.

Google Colab offre deux types de comptes : un compte gratuit et un compte payant. Le compte gratuit donne accès à des ressources limitées, dont 12 Go de RAM et un GPU Nvidia Tesla K80. Le compte payant, appelé Colab Pro, offre des ressources plus conséquentes, notamment 25 Go de RAM et des GPU Nvidia Tesla T4, ainsi que des fonctionnalités supplémentaires telles que l'accès prioritaire aux ressources de calcul et la gestion de sessions prolongées. En termes de stockage, Google Colab met à disposition un disque dur virtuel de 100 Go pour le stockage des données et des modèles.

Nous avons choisi d'utiliser 100 unités de calcul avec la formule Pay As You Go de Google Colab.

Pour notre projet, nous avons également utilisé les Jupyter Notebooks de Google Colab, qui permettent d'intégrer des cellules de code et des cellules de texte formaté en **markdown**. Cette fonctionnalité facilite l'explication des objectifs et des résultats du code, car elle permet de commenter directement le code dans le notebook. Elle réduit également la nécessité de recourir à des plateformes de communication externes telles que **Discord**, où les informations peuvent se perdre au fil des échanges de messages.

Enfin, nous avons utilisé **Google Drive pour stocker les images**. Cette solution est pratique car elle est facile à utiliser et offre suffisamment d'espace de stockage. De plus, l'intégration de Google Colab avec Google Drive facilite l'accès et l'utilisation des données stockées sur cette plateforme.

1. Un GPU, ou unité de traitement graphique, est un type de processeur spécialisé dans le traitement de tâches liées à l'affichage graphique. Le GPU permet d'optimiser le rendu des images 2D et 3D, ainsi que des vidéos. Le GPU est notamment utilisé pour l'animation 3D, le montage vidéo, les jeux vidéo, l'intelligence artificielle ou le minage de cryptomonnaies.

2.3 Stratégies de segmentation

La segmentation d'images est une technique de traitement d'images qui consiste à partitionner une image en régions homogènes selon certains critères. Il existe différentes types de segmentation d'images, donc différentes manières d'aborder ce projet, parmi lesquelles nous pouvons citer :

- **La segmentation par seuillage** : elle consiste à attribuer un label binaire (0 ou 1) à chaque pixel en fonction de son intensité, en comparant celle-ci à un seuil fixé ou calculé automatiquement. Cette méthode est simple et rapide, mais elle ne prend pas en compte la continuité spatiale des régions et peut être sensible au bruit ou aux variations d'éclairage.
- **La segmentation par croissance de région** : elle consiste à partir d'un ensemble de pixels initiaux (appelés germes) et à les faire croître en incorporant les pixels voisins qui ont des caractéristiques similaires (intensité, couleur, texture, etc.). Cette méthode permet de prendre en compte la continuité spatiale des régions, mais elle nécessite de choisir les germes de façon judicieuse et peut être sensible aux discontinuités ou aux trous dans les régions (fait significatif pour les aortes avec dissections²).
- **La segmentation par contour actif** : elle consiste à déformer un contour initial (appelé snake) pour qu'il épouse la forme des objets présents dans l'image, en minimisant une fonctionnelle qui dépend de l'énergie interne du contour (sa courbure) et de l'énergie externe (son attraction vers les bords des objets). Cette méthode permet de s'adapter aux formes complexes et irrégulières des objets, mais elle nécessite de choisir le contour initial de façon appropriée et peut être sensible aux minima locaux de la fonctionnelle.
- **La segmentation par réseaux de neurones convolutionnels** : elle consiste à utiliser un modèle d'apprentissage profond qui prend en entrée une image et qui produit en sortie une carte de labels pour chaque pixel, en fonction de la classe d'objet à laquelle il appartient. Cette méthode permet de traiter des images complexes et variées, mais elle nécessite de disposer d'un grand nombre d'images annotées pour entraîner le modèle et peut être coûteuse en termes de temps et de ressources.

Dans le cas de notre projet, la stratégie de segmentation a été basée sur l'utilisation d'un réseau de neurones convolutionnel de type U-Net, qui a été entraîné sur un ensemble de 900 images 2D annotées avec leurs masques, extraites d'un prétraitement des CT-Scan en format .nrrd. Cette approche a permis d'obtenir des résultats satisfaisants en termes de précision.

2.4 Réseaux de neurones

Un réseau de neurones est un modèle mathématique conçu pour traiter et analyser des données inspiré du fonctionnement du cerveau humain. Il se compose de couches de neurones interconnectées qui reçoivent des données en entrée, les transforment et produisent une sortie (Figure 5).

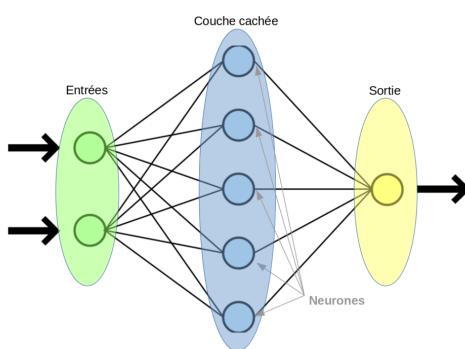


FIGURE 5 – Réseau de neurones [5]

2. Une dissection de l'aorte est une déchirure de la couche interne de la paroi de l'aorte. Le sang s'infiltra entre les couches de la paroi aortique. Cela crée un faux canal qui peut réduire ou bloquer le flux sanguin vers les organes vitaux, ou provoquer la rupture de l'aorte.

Parmi les différentes architectures de réseaux de neurones, certaines sont spécifiquement adaptées à des tâches particulières. Les architectures les plus répandues comprennent les réseaux de neurones convolutionnels (CNN) (Figure 6), les réseaux de neurones récurrents (RNN) et les réseaux de neurones à propagation avant (Feedforward Neural Network).

Les réseaux de neurones convolutionnels (CNN) sont fréquemment utilisés pour l'analyse d'images et de vidéos. Ils sont particulièrement performants pour la reconnaissance d'images, car ils peuvent apprendre automatiquement des caractéristiques à partir des données d'entrée. Les CNN exploitent le principe de la convolution, une opération mathématique qui consiste à appliquer un filtre sur une image pour en extraire des caractéristiques. Les filtres sont conçus pour détecter des motifs et des structures dans les données d'entrée. Ces caractéristiques sont ensuite transmises à des couches de neurones entièrement connectées, qui les combinent pour réaliser une classification ou une prédiction.

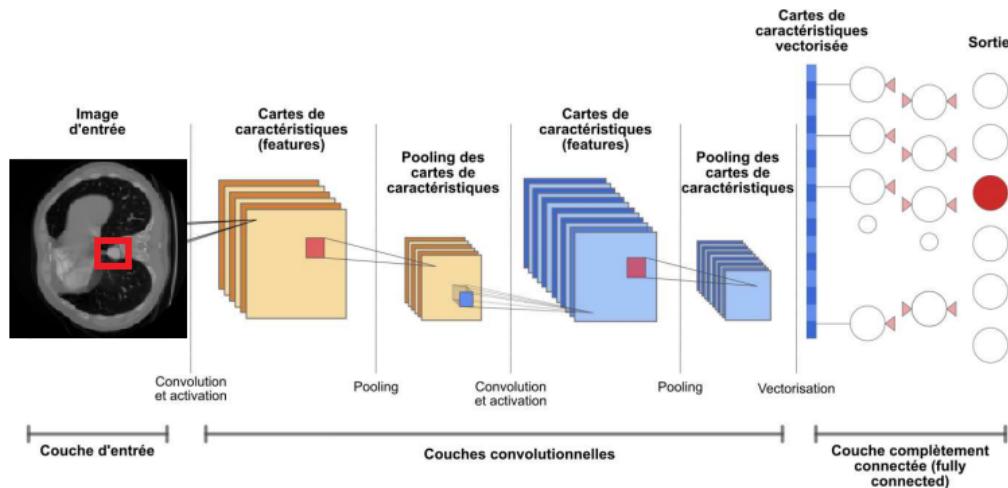


FIGURE 6 – Réseau de neurones convolutif [6]

En revanche, **les réseaux de neurones récurrents (RNN)** sont particulièrement adaptés aux tâches impliquant des séquences de données, comme la prédiction de séries temporelles ou la traduction automatique. Les RNN ont la capacité de prendre en compte l'information temporelle et de se souvenir des éléments précédents de la séquence pour produire une sortie. **Les réseaux de neurones à propagation avant (Feedforward Neural Network)**, quant à eux, sont une architecture de base qui ne tient pas compte de la structure spatiale ou temporelle des données. Ils sont plus adaptés aux problèmes de classification et de régression où les données d'entrée peuvent être considérées comme indépendantes.

2.4.1 U-Net

U-Net est un modèle de réseau de neurones convolutionnel spécifiquement conçu pour la segmentation d'images, notamment dans le domaine biomédical. Il s'appuie sur l'architecture entièrement convolutionnelle tout en l'adaptant et l'étendant pour fonctionner avec un nombre réduit d'images d'entraînement et produire une segmentation plus précise. U-Net possède une structure en forme de U (Figure 7), composée d'un chemin contractant et d'un chemin expansif, qui permet de combiner les informations contextuelles et les informations locales de l'image. Le chemin contractant diminue progressivement la dimension spatiale de l'image tout en augmentant le nombre de canaux de caractéristiques. À l'inverse, le chemin expansif rétablit la dimension spatiale et utilise des connexions transversales pour fusionner les caractéristiques issues du chemin contractant. Le réseau se concentre uniquement sur la partie valide de chaque convolution, sans inclure de couches entièrement connectées.

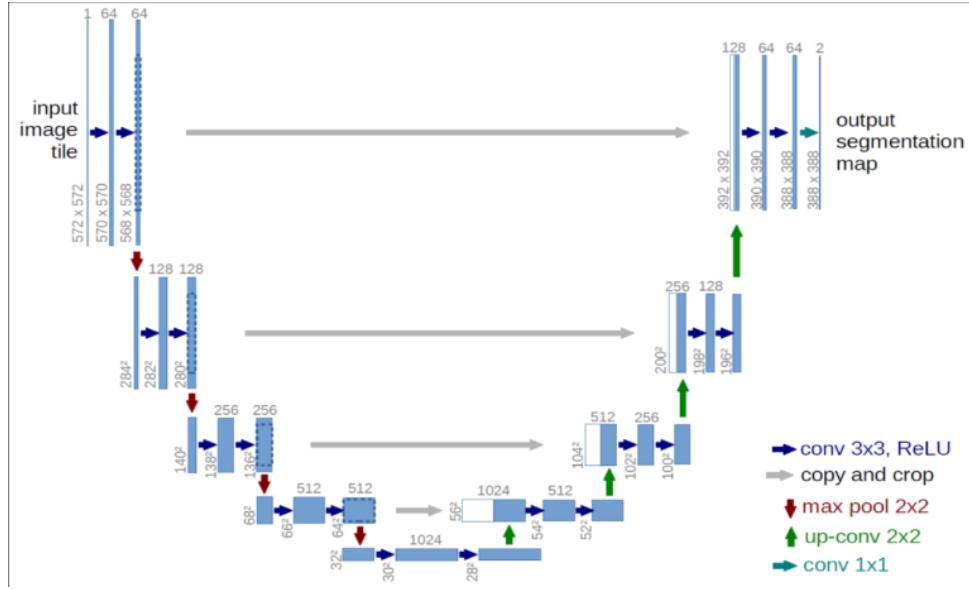


FIGURE 7 – Architecture U-Net [7]

Cette architecture est divisée en deux voies principales : la voie de contraction (Figure 8) et la voie d’expansion (Figure 9).

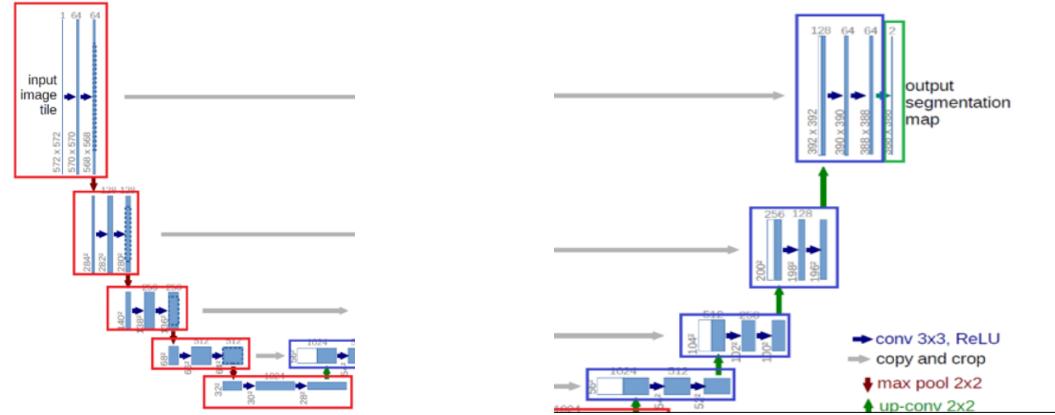


FIGURE 8 – Voie de contraction

FIGURE 9 – Voie d’expansion

La voie contractante du modèle U-Net est chargée de réduire progressivement la taille de l’image en entrée tout en augmentant la profondeur de la représentation. Elle le fait en utilisant des blocs de convolution suivis de couches de max-pooling, pour extraire les caractéristiques importantes de l’image nécessaires à la segmentation.

La voie expansive, quant à elle, est composée d’une série de blocs de convolution transposée, permettant d’augmenter la résolution spatiale de la carte de sortie. Ces blocs de convolution transposée sont associés à des blocs de convolution qui fusionnent les cartes de sortie de la voie contractante avec celles de la voie expansive.

2.4.2 Présentation des slices utilisées dans l’architecture U-Net

Les couches de convolution (Figure 10) sont responsables de l’extraction des caractéristiques des images en appliquant une matrice de filtres sur l’image d’entrée. Chaque filtre est une matrice de valeurs apprises pendant l’entraînement du modèle, permettant de capturer des caractéristiques spécifiques

telles que les bords, les textures ou les motifs. La sortie de chaque couche de convolution est une carte de caractéristiques qui est ensuite passée à la couche suivante pour extraire des caractéristiques plus complexes.

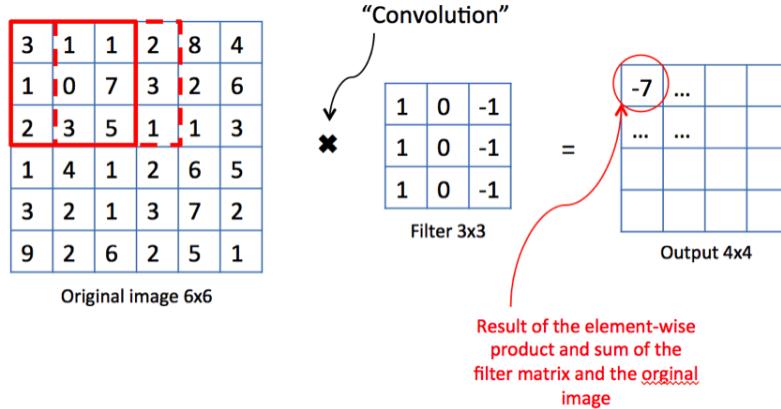


FIGURE 10 – Convolution [8]

Les couches de pooling, ou couches de regroupement en français, réduisent la résolution spatiale des cartes de caractéristiques en agrégant les valeurs voisines. Le but des couches de pooling est de réduire le nombre de paramètres à apprendre, d'augmenter l'invariance aux translations et de prévenir le surapprentissage ou overfit. Les couches de pooling les plus courantes sont max-pooling et average-pooling, qui extraient respectivement la valeur maximale et la valeur moyenne dans chaque fenêtre de pooling (Figure 11). La réduction de la résolution permet d'économiser des calculs et de rendre le modèle plus robuste face aux variations mineures de l'emplacement des caractéristiques.

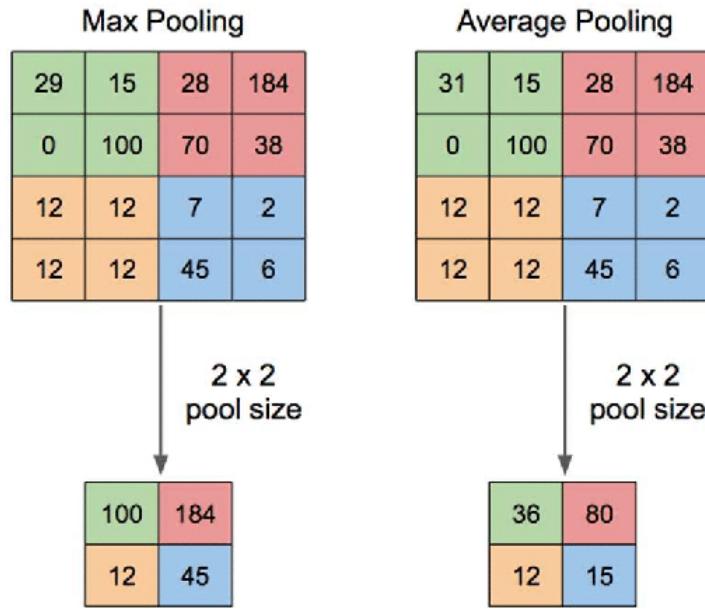


FIGURE 11 – Pooling [9]

Les couches de concaténation (Figure 12) sont utilisées pour combiner des informations provenant de différentes parties du réseau. Dans U-Net, elles sont utilisées pour fusionner des informations de haute et de basse résolution afin de segmenter une image. Plus précisément, les couches de concaténation

relient la voie contractante et la voie expansive du réseau. Les informations de haute résolution de la voie contractante (qui ont été "réduites" par des couches de pooling) sont transmises à la voie expansive via des couches de concaténation. Ces informations de haute résolution sont combinées avec les informations de basse résolution de la voie expansive pour produire une image segmentée avec des détails précis.

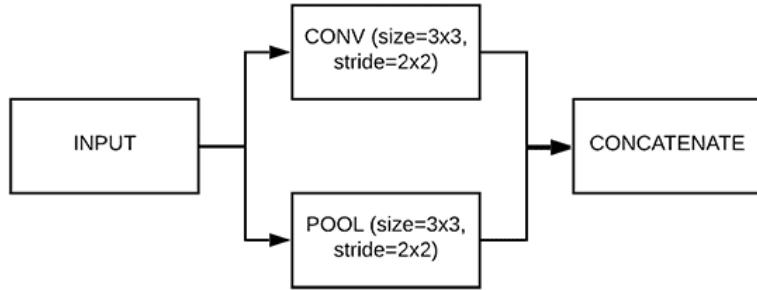


FIGURE 12 – Concaténation [10]

En résumé³, l'architecture U-Net est spécialement conçue pour la segmentation d'images, en particulier les images biomédicales. La voie contractante réduit la taille de l'image en entrée tout en capturant les caractéristiques importantes, tandis que la voie expansive récupère la résolution spatiale et fusionne les informations de haute et de basse résolution pour obtenir une segmentation précise de l'image. Les différentes couches de l'architecture, telles que les couches de convolution, de pooling et de concaténation, travaillent ensemble pour accomplir cette tâche de manière efficace.

3. Informations extraites des références [14, 15] et des pages source des images.

3 Structures des données

3.1 Le format .nrrd

Notre premier dataset (AVT) comprend des fichiers au format .nrrd [16].

Les fichiers au format **.nrrd (Nearly Raw Raster Data)** contiennent des données volumétriques et permettent de stocker des informations sur la taille, l'orientation, l'espacement et l'origine des voxels, ainsi que des métadonnées optionnelles. Ces fichiers sont utiles pour annoter les images médicales, par exemple pour entraîner des algorithmes de deep learning dédiés à la détection en imagerie médicale. Ils ont été développés par **Gordon Kindlmann** pour des applications de visualisation scientifique et de traitement d'image, telles que l'IRM, le scanner ou l'échographie.

Le format .nrrd est textuel et peut être facilement lu et modifié avec un éditeur de texte. Il prend également en charge la compression pour réduire la taille du fichier et est compatible avec plusieurs logiciels gratuits, tels que **3D Slicer**, **ITK-SNAP** ou **Seg3D**.

Contrairement aux images .png dont les valeurs de chaque pixel pour chaque couleur sont comprises entre [0,255], les fichiers .nrrd peuvent contenir des valeurs entières, signées ou non signées, ainsi que des valeurs flottantes. Cela permet d'inclure des valeurs négatives et très grandes. Par conséquent, une *normalisation des images* pourrait être nécessaire pour travailler avec ces fichiers.

3.2 Introduction à la normalisation

La normalisation des images est une étape importante du traitement des données visuelles. Elle consiste à ajuster la luminosité, le contraste et la gamme de couleurs des images pour les rendre plus homogènes et comparables. La normalisation des images peut avoir plusieurs avantages, tels que :

1. **Améliore la vitesse de convergence** : accélère l'apprentissage automatique de nombreux algorithmes, comme l'optimisation par descente de gradient, en harmonisant l'échelle des caractéristiques d'entrée et la complexité des données.
2. **Réduit le risque de gradients disparaissant ou explosant** : notamment avec les réseaux neuronaux profonds, la normalisation peut aider à atténuer le risque de gradients disparaissant ou explosant. Cela se produit lorsque les gradients deviennent trop petits ou trop grands, rendant difficile la poursuite du processus d'apprentissage.
3. **Améliore les performances du modèle** : Lorsque les caractéristiques d'entrée ont des échelles différentes, certaines caractéristiques peuvent avoir un impact disproportionné sur les prédictions du modèle. La normalisation égalise l'influence de chaque caractéristique, permettant au modèle de mieux identifier les motifs et les relations dans les données. En réduisant la plage des valeurs des pixels, on diminue la taille et la complexité des images. On peut ainsi accélérer le traitement et l'analyse des images, tout en économisant de l'espace de stockage et de la mémoire.
4. **Simplifie le réglage des hyperparamètres** : De nombreux algorithmes d'apprentissage automatique ont des hyperparamètres qui dépendent de l'échelle des caractéristiques d'entrée. En normalisant les données d'entrée, nous pouvons simplifier le processus de réglage des hyperparamètres.
5. **Permet la comparaison de différentes caractéristiques** : La normalisation nous permet de comparer l'importance ou la pertinence de différentes caractéristiques sur une échelle similaire. En rendant les images plus homogènes, on simplifie le travail des algorithmes de reconnaissance de formes ou de motifs. On peut ainsi détecter plus facilement les similitudes ou les différences entre les images.

3.2.1 Normalisation des images CT-Scan

Dans notre projet, avons-nous besoin de normaliser les données ?

Le modèle accordera plus d'importance à un dataset dont les valeurs varient entre -1000 et 1000 qu'à un dataset dont les valeurs se situent entre 0 et 100, ce qui peut entraîner un biais. Il est donc courant de normaliser ces datasets sur une plage commune, par exemple [0,1].

Nous chargeons d'abord les volumes un par un, puis examinons la plus petite et la plus grande valeur de chaque CT-Scan. Cependant, un problème se pose : les scanners, par nature, fonctionnent sur un cylindre. L'information recueillie est donc un cylindre, mais le format de données ne peut stocker que des hyperrectangles. Il y aura alors une "limite" entre le volume de travail du scanner et la limite du fichier (Figure 14). D'un autre côté, certains scans n'ont pas cette limite, car la valeur du "vide" est égale à la valeur minimale du scan du patient.

Pour normaliser nos données, nous avons utilisé la technique MinMax, qui consiste à décaler les valeurs pour que le minimum soit 0, puis à mettre à l'échelle les valeurs pour qu'elles se situent dans l'intervalle [0, 1]. Chaque valeur x se voit affecter la nouvelle valeur $(x - \min)/(max - \min)$ d'où le nom MinMax.

Dans le cas du jeu de données *Rider* (Figure 14), les valeurs correspondant aux tissus du patient et à l'air ambiant s'étendent sur l'intervalle [0,2047], mais le reste des valeurs sont affectées à -2000. Appliquer aveuglément la normalisation MinMax ici poserait un problème : les points extrêmes à -2000 domineraient le processus de normalisation, et les véritables données seraient écrasées dans un intervalle plus petit, ce qui pourrait entraîner une perte d'information.

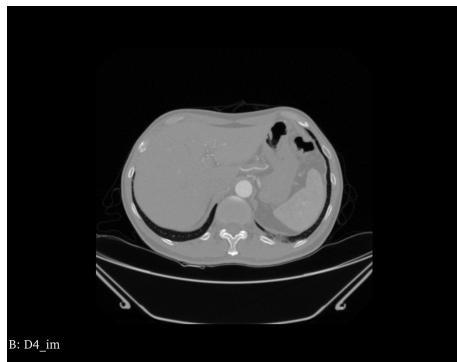


FIGURE 13 – Image D4_im.nrrd

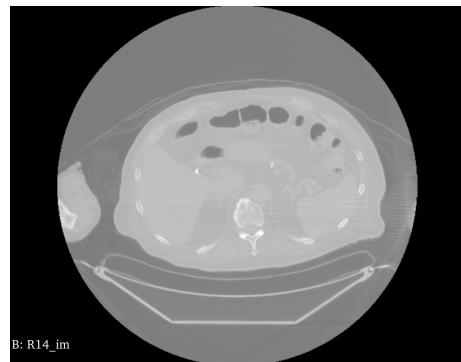


FIGURE 14 – Image R14_im.nrrd

FIGURE 15 – Deux types de bords pour la normalisation [4]

Il existe deux solutions courantes [17, 18] :

1. **Les valeurs extrêmes sont ignorées. Dans notre cas, nous considérerons que le minimum est bien 0. Nous attribuons les valeurs extrêmes à 0 et normalisons ensuite de manière classique.**
2. Nous traitons les deux groupes de manière indépendante. Les données réelles (comprises entre 0 et 2048) sont normalisées dans l'intervalle [0,1], et les valeurs aberrantes (-2000) reçoivent une valeur spéciale, par exemple -1.

La deuxième solution permet d'attribuer une valeur spéciale au "vide" et de différencier les données qui en contiennent de celles qui n'en contiennent pas. Toutefois, cette information n'est pas pertinente pour la segmentation de l'aorte et constituerait un élément superflu. **Nous choisirons donc la première solution.**

3.2.2 Implémentation

En chargeant un à un chaque dataset et en analysant quelques unes des valeurs minimum, on détermine le vrai minimum et le vrai maximum de nos jeux de données. Pour garder ce résultat, nous définissons un dictionnaire (Extrait de code 1) associant à chaque volume sa valeur minimum et maximum. De plus, ce dictionnaire nous permet d'éviter ce processus d'analyse à chaque chargement des données.

```

1 CT_extremas = {
2     'D1': {'min': -1024, 'max': 3071},
3     'D2': {'min': -1024, 'max': 3071},
4     'D3': {'min': -1024, 'max': 6484},
5     # ...
6     'D17': {'min': -1024, 'max': 14542},
7     'D18': {'min': -1024, 'max': 3071},
8
9     'K1': {'min': 0, 'max': 4095},
10    # ...
11    'K18': {'min': -2048, 'max': 5064},
12    'K19': {'min': 0, 'max': 2636},
13    'K20': {'min': 0, 'max': 4094},
14
15    'R3': {'min': 0, 'max': 4095},
16    # ...
17    'R18': {'min': 0, 'max': 2880}
18 }
```

Extrait de code 1 – Dictionnaire associant chaque volume à ses valeurs extrêmes réelles

On dispose maintenant d'une façon de charger et de normaliser automatiquement chaque CT-Scan (Extrait de code 2).

```

1 def load_and_normalize_volume(volume_path:Path)->np.ndarray:
2     """
3         This function inspects the path to determine which CT-Scan it refers to and uses the
4             CT_extremas dictionary to perform an optimal normalization.
5
6         The file targeted by volume_path must be in the format XX.nrrd, where XX corresponds
7             to a key in CT_extremas.
8
9     Args:
10        volume_path (Path): The path of the volume file to load and normalize.
11
12    Returns:
13        np.ndarray: A NumPy array containing the normalized volume data.
14
15    Example:
16        normalized_volume = load_and_normalize_volume(Path('CT-01.nrrd'))
17        """
18
19    ct_name = ct_name_from_path(volume_path)
20    extremas = CT_extremas[ct_name]
21    min, max = extremas['min'], extremas['max']
22
23    vol, _ = nrrd.read(volume_path)
24    vol[vol < min] = min
25    vol[vol > max] = max
26    vol -= min
27    return vol / (max-min)
```

Extrait de code 2 – Chargement des volumes en mémoire

3.3 Extraction de slices

Chaque volume chargé en mémoire est représenté sous la forme d'un tableau NumPy à 3 dimensions. Comme nous l'avons mentionné précédemment, notre objectif est d'extraire des slices 2D de ces volumes pour les fournir en entrée à notre CNN.

Les logiciels de traitement d'images médicales permettent de visualiser et d'enregistrer différentes tranches d'un volume, mais nous avons besoin d'une solution automatique et très rapide. La bibliothèque *NumPy* nous permet d'effectuer cela très simplement par la fonctionnalité de *slicing* (Figure 19).



FIGURE 16 – $vol[k,:,:]$: $k^{\text{ème}}$ couche sur l'axe X



FIGURE 17 – $vol[k,:,:]$: $k^{\text{ème}}$ couche sur l'axe Y

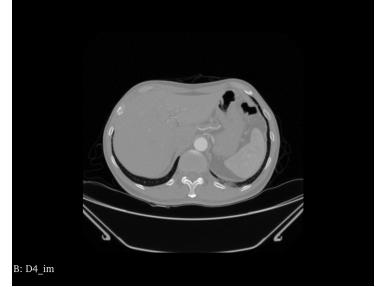


FIGURE 18 – $vol[k,:,:]$: $k^{\text{ème}}$ couche sur l'axe Z

FIGURE 19 – Extraction d'une couche d'un volume sur les trois axes [4]

3.3.1 Choix de traiter uniquement l'axe Z

Les volumes CT-Scan sont généralement composés de multiples coupes 2D empilées le long de trois axes principaux : l'axe axial (ou transversal, correspondant à l'axe Z), l'axe coronal (axe Y), et l'axe sagittal (axe X). Chaque axe offre une perspective différente de l'anatomie humaine.

Pour ce projet de segmentation de l'aorte, nous avons choisi de nous concentrer exclusivement sur l'axe Z. Cette décision est justifiée par la relative simplicité des structures anatomiques observées dans ce plan. En effet, l'aorte et d'autres structures vasculaires apparaissent souvent sous des formes circulaires ou elliptiques sur l'axe Z, ce qui simplifie grandement la tâche de segmentation.

3.3.2 Nombre de slices par volume

Nos volumes ont un axe Z de tailles allant de 96 à 1140, ce qui signifie que des tranches adjacentes peuvent être très similaires. Par conséquent, il n'est pas souhaitable d'extraire toutes les tranches possibles de chaque volume. On se retrouverait avec plus de 16 000 images sur notre dataset AVT. Au lieu de cela, nous pouvons choisir d'extraire un nombre réduit de tranches, comme par exemple 20, 30 ou plus, réparties uniformément sur tout l'axe Z. Ceci nous donnera la liberté d'augmenter le nombre d'images si notre modèle n'est pas assez performant.

Pour notre première session d'entraînement, nous avons prévu d'utiliser 4096 images. Étant donné que nous disposons de 50 volumes, nous devrions théoriquement extraire 81.92 tranches par volume. Cependant, puisque nous ne pouvons pas extraire une fraction de tranche, nous arrondissons ce nombre à 82 tranches par volume. Cela nous donne un total de 4100 images. Bien sûr, l'ensemble de ce processus est automatisé.

L'extraction de n slices réparties uniformément dans un volume est implémentée par la fonction de l'extrait de code 3.

```

1 def make_n_data_points(vol:np.ndarray, n:int=None):
2     """
3         Extracts 'n' uniformly distributed slices along the Z-axis from the given volume.
4
5     Args:

```

```

6     vol (np.ndarray): The input 3D volume represented as a numpy array.
7     n (int, optional): The number of slices to extract. If not specified or too
8     large, all possible slices will be extracted.
9
10    Returns:
11        tuple: A tuple containing two numpy arrays:
12            - indexes (np.ndarray): An array containing the indices of each extracted
13            slice.
14            - slices (np.ndarray): An array with the same dimensions as the input volume
15            , but with only 'n' slices retained.
16
17    Example:
18        If the input volume has 10 slices along the Z-axis, and 'n' is set to 3, the
19        function will return: (np.array([0, 5, 9]), original volume with only the 3
20        specified slices retained)
21
22
23
24
25
26
27
28
29    if not n or n > vol.shape[2]:
30        n = vol.shape[2]
31
32    indexes = np.linspace(0, vol.shape[2]-1, num=n, dtype=np.int32)
33
34    slices = np.empty((vol.shape[0], vol.shape[1], n))
35
36    slices = vol[:, :, indexes].copy()
37    # The copy is important because it ensures that only the extracted slices are kept in
38    # memory, while in the opposite case, the entire volumes would be retained.
39
40
41    return indexes, slices

```

Extrait de code 3 – Fonction permettant l'extraction couches réparties uniformément dans un volume

3.3.3 Gestion des slices

Lors de l'entraînement et du test de notre CNN, nous devons fournir des images 2D extraites des CT-Scans du dataset. Cependant, il est essentiel de minimiser le temps de chargement en mémoire des scans et de limiter le nombre de scans en mémoire pour éviter de dépasser la capacité maximale de mémoire vive.

Charger tous les datasets en une fois dans la mémoire n'est pas possible avec 12 Go de RAM, le forfait de base de Google Colab. Ainsi, lors de l'entraînement, nous pouvons charger le volume en mémoire, extraire les slices concernées, les fournir au modèle puis libérer la mémoire.

Toutefois, durant nos expérimentations, nous avons souvent eu besoin d'accéder aux différentes slices, et le chargement du volume à chaque fois est coûteux en temps. Nous avons donc décidé de profiter des 100 Go de disque dur de Google Colab et d'enregistrer les slices sur le disque dur dans une étape intermédiaire.

La création de toutes les slices sur le disque dur prend environ 5 minutes pour 4100 images, mais cette durée pourrait être considérablement réduite en utilisant plusieurs threads.

Quant au format de sauvegarde des slices, celles-ci étant en 2 dimensions, elles pourraient être enregistrées en .png, en .jpeg, etc. Cependant, cela nécessiterait une conversion supplémentaire et entraînerait probablement une perte d'information. Nous avons donc opté pour l'utilisation de la fonction *np.save()*, qui permet de sauvegarder n'importe quel array sur le disque dur, et récupérer ces données avec *np.load()*. Ces opérations sont directes et optimisées, et il est toujours possible d'exporter les slices au format PNG si nécessaire.

3.4 Organisation des données pour l'entraînement du modèle U-Net

L'entraînement du réseau consiste à ajuster les paramètres du modèle en utilisant un ensemble d'images d'entraînement. Pour évaluer les performances du modèle, il est nécessaire de le tester sur

un ensemble d’images de test, distinct de l’ensemble d’entraînement. Cette étape permet de vérifier la capacité du modèle à généraliser à de nouvelles données.

Pour que les algorithmes que nous avons mis en œuvre sur Google Collab fonctionnent correctement, il faut que les images fournies en paramètre soient "carrées", c'est-à-dire de dimensions $A \times B$ avec $A = B$, car nous effectuons un redimensionnement des images de manière à réduire ou augmenter la taille des images à la taille (512, 512). Sinon, le redimensionnement se fera de manière disproportionnée. Par exemple, une image de 512×666 pixels sera réduite à 512×512 pixels, ce qui entraînera une déformation de l’image. Le modèle apprendrait donc à partir d’images qui ne correspondent pas à la réalité d’une aorte et d’images mal ajustées, ce qui affecterait la qualité des prédictions car elles ne seraient pas toujours bien positionnées.

Choix de la séparation entre entraînement et test

Il existe plusieurs façons de tester notre CNN :

- En utilisant une validation croisée (k-fold cross-validation) : on divise l’ensemble de données en k sous-ensembles. Chacun de ces sous-ensembles est utilisé une fois comme ensemble de test tandis que les $k-1$ autres sous-ensembles sont combinés pour constituer l’ensemble d’entraînement. On répète ce processus k fois et on calcule la moyenne des performances pour obtenir une estimation plus robuste.
- En séparant les données en ensembles d’entraînement, de validation et de test : on utilise l’ensemble de validation pour ajuster les hyperparamètres du modèle et l’ensemble de test pour évaluer les performances finales du modèle. **C’est cette méthode que nous utiliserons.**

Il est important de prendre en compte les problèmes d’**overfitting** et d’**underfitting** lors de l’entraînement du réseau. L’overfitting se produit lorsque le modèle apprend trop bien les données d’entraînement au détriment de sa capacité à généraliser à de nouvelles données. L’underfitting, quant à lui, se produit lorsque le modèle n’apprend pas suffisamment les caractéristiques des données d’entraînement et, par conséquent, ne peut pas bien fonctionner sur les données de test. Une bonne séparation entre entraînement et test permet d’identifier et de gérer ces problèmes de performance⁴.

4. Voir section 9.3.6 *Overfit et curiosités* et références [19, 20] pour plus d’informations.

4 Algorithmes

4.1 Implémentation du modèle U-Net

Le format des données en entrée pour l'algorithme présenté est une image 2D, qui doit être fournie sous forme de tenseur numpy avec des dimensions (largeur, hauteur, canaux). Le nombre de canaux doit être égal à 1 pour les images en niveaux de gris ou à 3 pour les images en couleur (RGB). Comme mentionné à la section antérieure, les entrées doivent être prétraitées avant d'être fournies au modèle.

```
1 def unet_model(input_shape):
2     inputs = tf.keras.Input(shape=input_shape)
3
4     # Contracting path (Encoder)
5     conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
6     conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
7     pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
8
9     conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
10    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
11    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
12
13    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
14    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)
15    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
16
17    # Bottleneck
18    conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool3)
19    conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)
20
21    # Expanding path (Decoder)
22    up5 = concatenate([UpSampling2D(size=(2, 2))(conv4), conv3], axis=-1)
23    conv5 = Conv2D(256, (3, 3), activation='relu', padding='same')(up5)
24    conv5 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv5)
25
26    up6 = concatenate([UpSampling2D(size=(2, 2))(conv5), conv2], axis=-1)
27    conv6 = Conv2D(128, (3, 3), activation='relu', padding='same')(up6)
28    conv6 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv6)
29
30    up7 = concatenate([UpSampling2D(size=(2, 2))(conv6), conv1], axis=-1)
31    conv7 = Conv2D(64, (3, 3), activation='relu', padding='same')(up7)
32    conv7 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv7)
33
34    # Output layer
35    output = Conv2D(1, (1, 1), activation='sigmoid')(conv7)
36
37    return Model(inputs=inputs, outputs=output)
```

Extrait de code 4 – Implémentation de l'architecture U-Net avec Tensorflow

Pour la lecture et la validation des entrées, il est important de s'assurer que les images sont correctement dimensionnées et normalisées. Si les images ont des dimensions différentes, elles doivent être redimensionnées pour avoir les mêmes dimensions que les images d'entraînement utilisées pour le modèle. Les images doivent également être normalisées pour avoir une valeur de pixel comprise entre 0 et 1.

Pour illustrer le fonctionnement de l'algorithme, on peut utiliser un exemple simple : l'image est fournie en entrée au modèle et il prédit une carte de segmentation binaire pour indiquer les pixels de l'aorte. La carte de segmentation est ensuite comparée à la carte de vérité terrain pour évaluer les performances du modèle.

4.2 Reconstruction 3D

Une fois que notre modèle a réussi à segmenter automatiquement l'aorte sur des slices 2D de CT-Scan, il est naturel d'envisager de l'utiliser pour obtenir une représentation 3D de l'aorte. Cette démarche offre plusieurs avantages.

Tout d'abord, une reconstruction 3D fournit une représentation continue et complète de l'aorte, ce qui améliore la précision du diagnostic et la planification du traitement. Ensuite, l'analyse 3D permet une meilleure compréhension des relations spatiales entre l'aorte et les structures adjacentes, ce qui est crucial pour la détection des maladies et la planification des interventions. Enfin, la visualisation 3D aide les cliniciens à mieux comprendre l'anatomie de l'aorte et sa pathologie, facilitant ainsi l'interprétation des résultats et la communication avec les patients.

Notre processus de segmentation d'un volume 3D est orchestré par une fonction automatisée. Cette fonction traite de manière séquentielle chaque slice du volume, prédit un masque 2D correspondant, et assemble ensuite ces masques pour former un volume 3D.

En termes de rapidité, l'opération complète prend environ 10 secondes pour traiter un volume de dimensions $512 \times 666 \times 136$, avec l'aide d'une carte graphique **Nvidia V100** fournie par Google Colab.

4.3 Statistiques

Notre projet a un total de 1553 lignes de code, sans compter les lignes d'explications des fichiers Jupyter Notebook .ipynb. Il a été fait en 2 étapes. Une première étape que nous pourrions considérer de recherche et la première approche de notre groupe aux réseaux de neurones avec les programmes dataPreparation.py et testing.py, et une deuxième étape en utilisant Google Colab :

- Première étape : recherche et collection d'informations et familiarisation du sujet en 2 programmes : 226 lignes de dataPreparation.py + 230 lignes de testing.py = 456 lignes en total.
- Programme final en utilisant Google Colab : 616 lignes de normalisation d'images et d'implémentation U-Net + 481 lignes de reconstruction 3D = 1097 lignes de code en total.

5 Gestion du Projet - Groupe N de Notion

5.1 Présentation générale de Notion

Notion est une application polyvalente de prise de notes et de gestion de projets, adaptée aussi bien à un usage personnel qu'en équipe. Elle offre une grande flexibilité dans la création de pages et de bases de données, permettant de structurer l'information de manière claire et efficace.

5.2 Collaboration et intégrations

Grâce à la collaboration en temps réel, les membres de l'équipe peuvent travailler ensemble sur les pages et les éléments de Notion. Chaque page ou élément peut être référencé n'importe où grâce à un simple lien, offrant ainsi la possibilité d'ajouter du contexte à la documentation du code, par exemple. Notion intègre de nombreuses fonctionnalités, telles que l'insertion de dossiers, fichiers Google Drive, documents, ou projets GitHub.

The screenshot displays the Notion interface for a project titled "Projet de programmation 2".

Project Details:

- Code UE: HAI6061
- # Ects: 5
- + Add a property

Comments:

- Add a comment...

Backlinks:

- 1 backlink from PP2-aorte-segmentation by SuperMuel (Updated 16 days ago)

Links:

- Sujet
- Ressources
- Gestion du projet et Notion
- Compte-rendu

Liens rapides:

- Notebooks:
 - [U-NET_model_with_AVT_dataset.ipynb](#)
 - [predict_and_reconstruct.ipynb](#)
- Compte rendu : [Compte rendu Overleaf](#)
- Storage : [Dossier Google Drive Sam](#)

FIGURE 20 – Aperçu de la page d'accueil du projet [4]

5.3 Organisation des données de CT-Scans

L'une des principales forces de Notion réside dans ses bases de données. Elles permettent une organisation claire et puissante de l'information. Chaque entrée dans la base de données est une page, dotée d'attributs personnalisables.

Pendant notre projet, nous avons utilisé les bases de données pour organiser les données de CT-Scans. Nous avons contacté plusieurs établissements hospitaliers et universitaires en France et en Espagne par courriel pour obtenir des scanners. Cependant, nous n'avons reçu aucune réponse positive ou de réponse tout court pour accéder aux images. Par conséquent, nous avons recherché sur internet des bases de données d'aorte et pris contact avec monsieur Xiaowei Xu [21], professeur associé à l'institut de cardiologie du Guangdong, qui nous a partagé un dossier Google Drive contenant des volumes CT-scan d'aortes.

Chaque demande de CT-Scans adressée à des hôpitaux ou à des individus est représentée par une page Notion, avec des attributs personnalisés tels qu'un nom, une URL et un statut permettant de suivre l'évolution de la demande (débutée, en cours, refusée).

Une fois que nous avons obtenu l'accès aux différents jeux de données, nous avons transformé cette base de données en un répertoire centralisé où nous répertorions tous les datasets, accompagnés de notes descriptives (Figure 21).

Données

Aa Name	🔗 URL	Status demande	Type d'aorte	# Taille totale (GO)	...
Image, geometry and finite element mesh datasets	https://www.sciencedirect.co	Données reçues	Anévrisme	1.3	
"Aortic Vessel Tree"	https://figshare.com/articles/	Données reçues	Saine	5.9	
Données challenge	http://www.sdspeople.fudan	Abandonnée			
XiaoweiXu /Dataset_Type-B-Aortic-Dissection	https://github.com/XiaoweiXu	Données reçues	Dissection	14	
CENTRE DE RADIOLOGIE ET SCANNER CLINIQUE	https://www.radiologie34.cor	En attente de réponse			
CLINIQUE Saint Jean	https://www.radiologie34.cor	En attente de réponse			
CLINIQUE SAINT LOUIS	https://www.radiologie34.cor	Refusée			
CENTRE DE RADIOLOGIE, IRM ET SCANNER CLI	https://www.radiologie34.cor	En attente de réponse			
Imagerie médicale Lapeyronie	https://www.chu-montpellier	En attente de réponse			

FIGURE 21 – Base de données des CT-Scans [4]

5.4 Timeline et organisation des tâches

Pour obtenir une vue d'ensemble des différentes échéances et des tâches à accomplir, nous avons utilisé la fonctionnalité de timeline de Notion. Cette fonctionnalité nous a permis de visualiser clairement les étapes du projet et de mieux planifier notre travail.

La timeline a été un outil précieux pour gérer les délais et répartir les tâches de manière efficace entre les membres de l'équipe. Elle a contribué à améliorer la communication et la coordination, garantissant ainsi que toutes les tâches soient réalisées dans les délais impartis (Figure 22).

Timeline

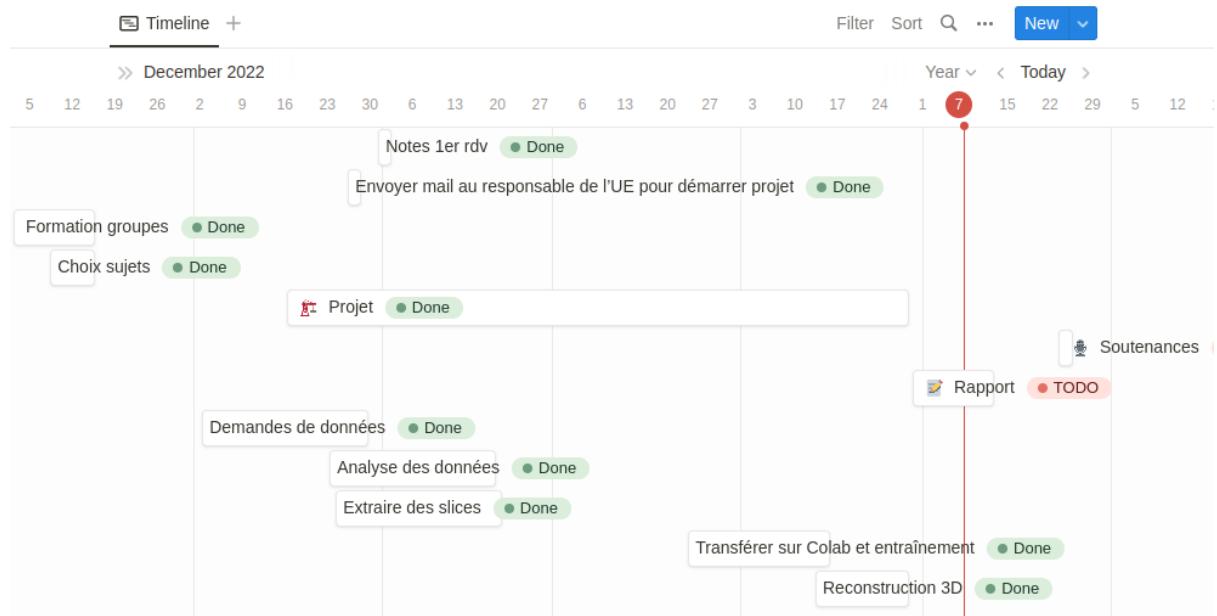


FIGURE 22 – Diagramme de Gantt [4]

6 Bilan et analyse des résultats

6.1 Problème rencontré lors de l'exécution du programme testing.py

Nos ordinateurs portables ne sont pas assez performants pour le training, qui est très lent et qui bloque les epochs. **Solutions envisageables** : utiliser une machine plus puissante avec une meilleure GPU ; ajuster le nombre d'epochs et la taille du batch ; réduire la résolution des images. Pour les images de 512×512 pixels, même en modifiant les paramètres du training, celui-ci reste très lent (il faut laisser l'ordinateur allumé pendant 24h pour atteindre 20 epochs, et ensuite la mémoire de l'ordinateur est saturée). Nous avons donc essayé de diminuer la résolution des images à des valeurs très faibles, ce qui a permis d'accélérer le training sur nos machines personnelles, mais au détriment de la qualité de prédiction (image résultante toute noire ou toute blanche).

Pour réaliser le training avec une meilleure GPU, nous avons dû chercher des ressources alternatives. Nous avons contacté un autre groupe du projet de programmation qui utilisait aussi les réseaux de neurones, et notre camarade de classe Gabriel Combe-Ounkham nous a gentiment prêté son ordinateur équipé d'un **GPU NVIDIA RTX 3080**. Il nous a ensuite envoyé les modèles (fichiers .h5) une fois le training terminé, ce qui nous a permis de les examiner sur notre ordinateur comme expliqué sur la section [9.3 Deuxième programme : testing.py](#).

Comme vous pouvez voir sur les graphes des résultats sur la section [9.3.5 Résultats](#), les modèles commencent à **overfit** vu que les points du graphe *val_loss* ne font qu'augmenter mais il est encore acceptable car l'axe Y est une puissance de 10 inférieure à l'échelle de l'axe Y du graphe *loss*.

Vu ces résultats, nous avons dédié une section [9.3.6 Overfit et curiosités](#). Après cela et de la recommandation de notre encadrante, madame Houda Hammami, nous avons commencé à utiliser Google Colab, donnant lieu à la deuxième étape du projet.

6.2 Programmes finaux : Google Colab

Sur les fichiers .ipynb du projet [22] vous trouverez les programmes finaux utilisés sur Google Colab. Ils incluent :

- Des explications pour chaque cellule de code
- Mise en place de la connexion à Google Drive pour accéder au dataset AVT
- Affichage de certains CT-Scan
- Normalisation des données (MinMax)
- La répartition des images pour le training et pour le test
- L'implémentation d'un réseau de neurones suivant l'architecture U-Net
- La sauvegarde du modèle
- Les graphiques avec les résultats
- Les prédictions du modèle
- Le code pour la reconstruction 3D des segmentations.

Nous avons entraîné notre réseau neuronal sur 10 epochs d'un jeu de données de 3198 couches extraites parmi le jeu de données AVT. Notre dataset d'entraînement a été reparti de cette manière :

- Train :
D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14,
K1, K2, K3, K4, K5, K6, K7, K8, K9, K10, K11, K12, K13, K14, K15, K16,
R3, R4, R9, R10, R11, R12, R13, R14 et R15.
- Test : les autres
D15, D16, D17, D18, K17, K18, K19, K20, R16, R17 et R18.

Voici une prédiction faite par le modèle sur D15.nrrd :

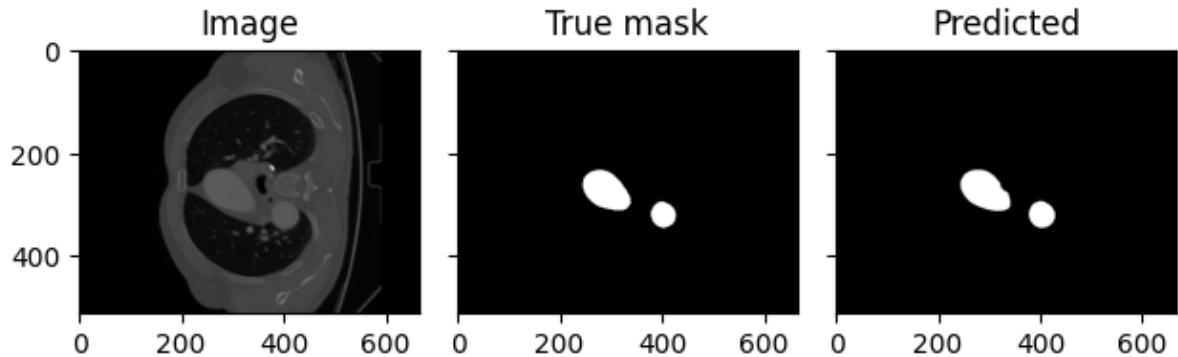


FIGURE 23 – 100ème couche de D15.nrrd prédite par le modèle [4]

6.2.1 Plot training & validation loss values

La valeur de la perte (« loss ») représente la somme des erreurs pour chaque image que nous testons. Elle permet de nous indiquer dans quelle mesure un modèle donné se comporte bien ou mal après chaque epoch. L’objectif est de minimiser cette perte au fur et à mesure des itérations.

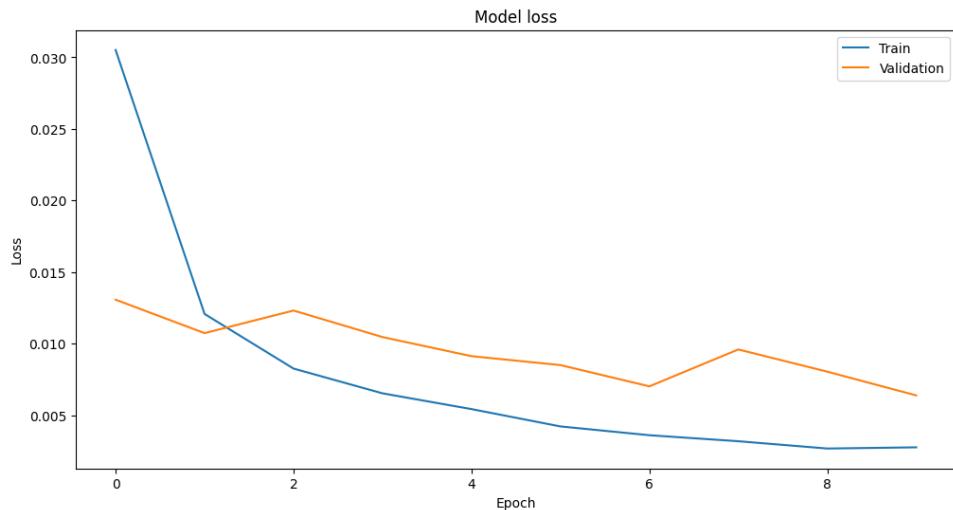


FIGURE 24 – Perte de notre modèle par epoch [4]

6.2.2 Plot training & validation accuracy values

Contrairement à la perte, le taux de précision (« accuracy ») représente le pourcentage d’erreurs de classifications, c’est-à-dire le nombre d’erreurs commises par le modèle comparées avec les vraies cibles. Ce calcul se fait généralement après que les paramètres du modèle soient appris, ici nous regardons la courbe d’évolution de ce taux tout au long de l’apprentissage. L’objectif étant ici de maximiser ce taux, afin d’avoir un modèle précis (non overfit).

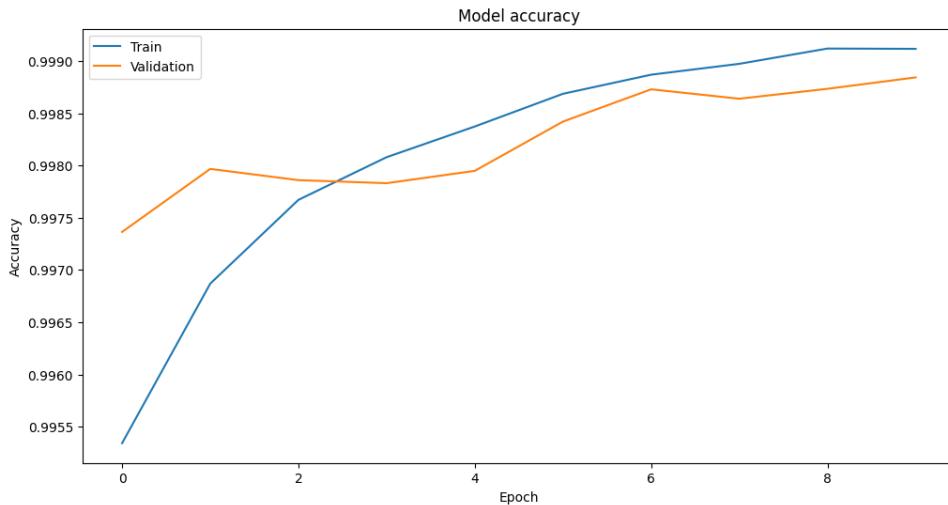


FIGURE 25 – Taux de précision de notre modèle par epoch [4]

Nous pouvons remarquer qu'à chaque cycle dans notre set de données la perte de notre modèle a tendance à diminuer, et que le taux de précision, lui, a tendance à augmenter, nous pouvons en conclure que notre entraînement s'est donc déroulé dans le bon sens et une amélioration en comparaison avec les résultats de testing.py. **Le modèle a atteint une précision finale égale à 0.999124.**

6.3 Reconstruction 3D de l'aorte

Il est toutefois essentiel de souligner que notre approche actuelle traite chaque slice indépendamment des autres. Cette méthode néglige donc potentiellement des informations précieuses contenues dans les différentes couches du volume qui pourraient aider à améliorer l'exactitude de la segmentation.

Ainsi, il serait intéressant de mettre nos résultats en parallèle avec ceux obtenus par d'autres techniques de segmentation qui utilisent l'ensemble du volume 3D en tant qu'entrée. Il existe en effet des solutions de ce type, telles que 3D-Unet [23] et V-Net [24].

Ces modèles exploitent l'information contextuelle 3D à travers l'ensemble du volume, permettant potentiellement une meilleure segmentation. En particulier, 3D-Unet étend l'architecture U-Net classique pour travailler directement avec des données 3D. De même, V-Net est une architecture de réseau neuronal convolutif profond spécifiquement conçue pour la segmentation volumétrique 3D.

L'utilisation de tels modèles pourrait améliorer la qualité de nos segmentations en intégrant des informations contextuelles à travers l'ensemble du volume. Cela pourrait notamment permettre de mieux gérer les zones de l'image où l'anatomie varie rapidement en profondeur. Cependant, ces modèles nécessitent généralement plus de ressources computationnelles et peuvent être plus difficiles à entraîner.

Utiliser des seuils pour la reconstruction

Sur le fichier *predict_and_reconstruct*, la fonction *predict_3D_mask* retourne un tableau numpy 3D représentant le masque prédict. Cette masque contient des valeurs continues comprises entre 0 et 1 représentant la confiance de prédiction du modèle, c'est-à-dire, les valeurs représentent la probabilité que le pixel fasse partie de l'aorte⁵. Nous utilisons ces seuils afin d'obtenir différentes masques et pouvoir faire plusieurs reconstructions.

Vous pouvez trouver les prédictions du modèle au format .nrrd sur le link du Drive de la partie Abstract avec les différents seuils allant de 0.1 à 0.9. Sur les références de la section 9.1 *3DSlicer* vous trouverez des vidéos tutoriel afin de pouvoir les visualiser⁶.

5. Voir section 9.4.1 *Extrait de code de predict_and_reconstruct.ipynb - Explication des seuils*

6. Reconstruction complète de D15.nrrd et plus de reconstructions sur la section 9.4 *PROGRAMMES FINAUX : Google Colab*.

Voici un résultat de la reconstruction faite par notre modèle sur l'image D15.nrrd affiché avec 3DSlicer :

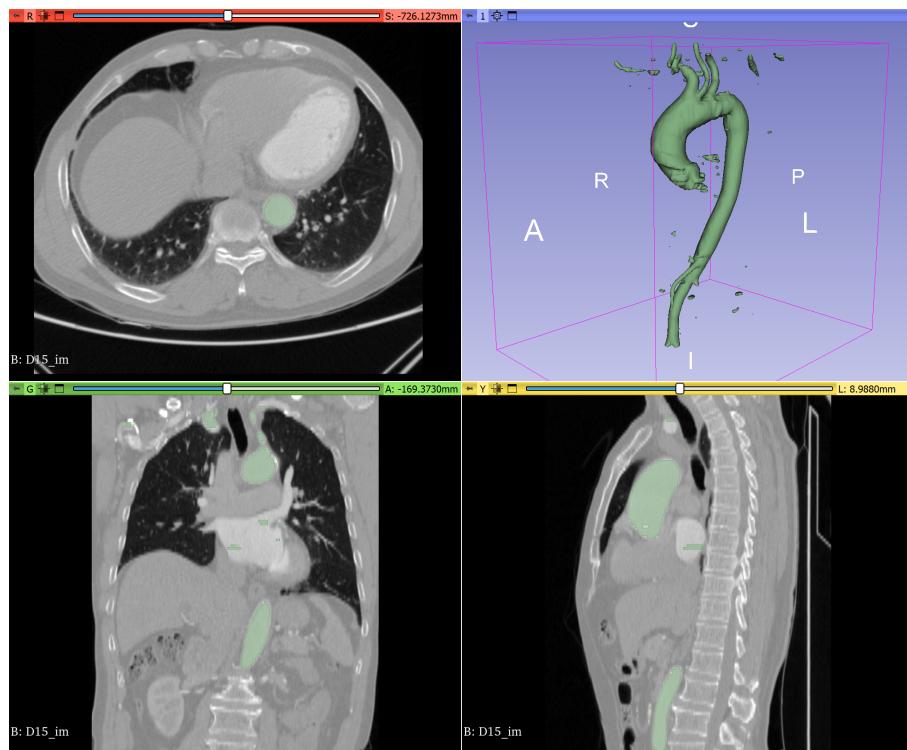


FIGURE 26 – Prédiction - Seuil 0.1 [4]

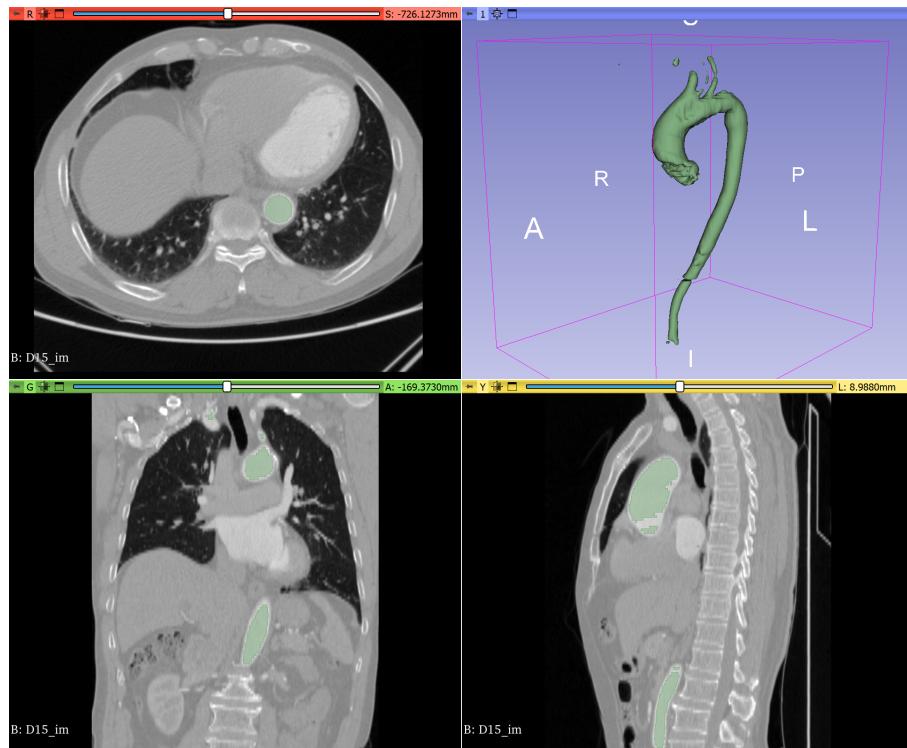


FIGURE 27 – Prédiction - Seuil 0.9 [4]

Nous avons observé que la plupart des reconstructions ont abouti à de bons résultats, sauf R16.nrrd, qui présente des anomalies :

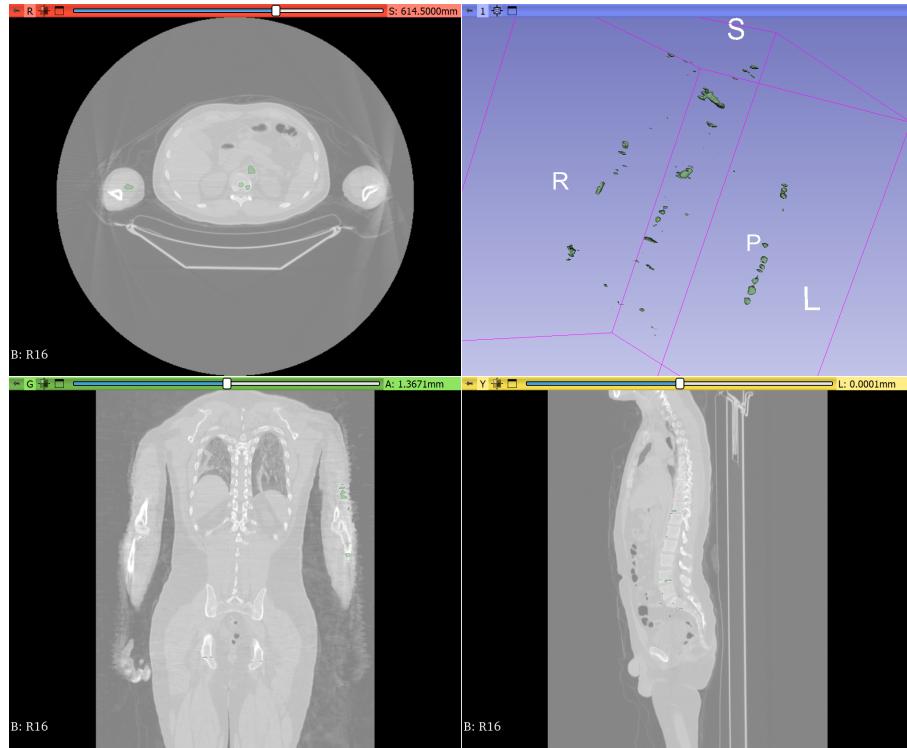


FIGURE 28 – Prédiction - Seuil 0.1 [4]

Par ailleurs, les performances sur le jeu de données de *Rider* sont inférieures à celles obtenues sur les deux autres jeux de données. Cela pourrait s'expliquer par le contraste des images de *Rider*, qui présentent des différences avec celles des jeux de données *Dongyang* et *KiTS*, comme le niveau de zoom ou la qualité des contours. Néanmoins, certains fichiers du jeu de données *Rider* ont été correctement reconstruits, comme R17.nrrd⁷.

7. Voir la section 9.4.3 Reconstruction 3D - R17.nrrd

7 Conclusions

L'objectif principal de notre projet était de créer un outil permettant la segmentation automatique de l'aorte dans les volumes de CT-Scan en utilisant les réseaux de neurones. Après avoir recherché et analysé différents projets répondants à des problématiques similaires, nous avons réussi à aboutir à un outil qui est à la fois simple, modulaire et efficace.

Notre code a été premièrement pensé pour travailler sur l'axe Z avec `testing.py`, mais les images .nrrd fournissaient également les axes X et Y. Ainsi, dans la deuxième étape du projet nous avons implémenté un modèle qui les prend en compte et nous avons obtenu une segmentation sur les 3 axes⁸.

Nous avons non seulement atteint l'objectif du projet, mais nous sommes également allés plus loin en réalisant une reconstruction 3D d'une aorte à partir des segmentations des prédictions faites par notre modèle. Nous avons visualisé cette reconstruction avec 3DSlicer, ce qui nous a permis de vérifier la qualité de la segmentation du modèle : plus la reconstruction ressemble à la segmentation manuelle, meilleur est le modèle.

Nous avons observé que les codes basés sur l'architecture U-Net ont une structure similaire : définition des variables globales en testant différents paramètres, normalisation (si besoin) des images, implémentation d'un modèle U-Net (soit créé par nous-mêmes, soit en réutilisant le code d'un autre projet) et visualisation des résultats. Nous avons documenté tout le code pour qu'il soit facile de modifier les paramètres et les données d'entrée. Ainsi, notre code peut servir de référence à d'autres projets, mais aussi être adapté à d'autres applications de segmentation d'images.

Une perspective d'approfondissement de ce projet serait de développer un outil informatique capable d'évaluer le risque de dissection aortique à partir des images médicales et de déterminer la nécessité d'une intervention chirurgicale. En effet, la prise en charge actuelle repose sur une chirurgie qui n'est pas sans risques ni complications. Pouvoir faciliter le travail des médecins rendrait cette prise en charge moins dangereuse.

Comme mentionné dans les remerciements, nous pourrions utiliser le jeu de données de Xiao Wei Xu, qui contient des images d'aortes dissectionnées, pour le test et vérifier si le modèle est capable de reconnaître une aorte dissectionnée. Nous avons également segmenté manuellement des aortes avec le logiciel 3DSlicer, qui pourraient servir à l'entraînement du modèle en complément des données de monsieur Xu, afin d'obtenir un jeu de données plus large et de comparer la perte de validation et la perte des graphes à la fin de l'entraînement. Par conséquent, en disposant de plus d'images distinctes, nous pourrions bénéficier d'une amélioration du modèle qui éviterait les erreurs de prédiction comme sur le fichier R16.nrrd.

Après la révision du projet par les enseignants de l'université nous pouvons le traduire en anglais et en espagnol. Ce processus nous permettra de diffuser nos résultats à un public plus large et de bénéficier de leurs commentaires et suggestions, ainsi que de soutenir d'autres étudiants non-francophones qui souhaitent réaliser un projet similaire.

Finalement, ce projet a été très enrichissant et nous a apporté beaucoup de bénéfices sur le plan professionnel et personnel. Nous avons pu nous familiariser avec des techniques avancées sur les réseaux de neurones qui sont un domaine en pleine croissance. Il nous a permis de développer nos capacités de travail en équipe, de communication et de gestion du temps. Nous sommes fiers du résultat obtenu.

8. Voir section [6.3 Reconstruction 3D de l'aorte](#)

8 Bibliographie

Références

- [1] Santé Publique France. Anévrisme de l'aorte abdominal. <https://www.santepubliquefrance.fr/maladies-et-traumatismes/maladies-cardiovasculaires-et-accident-vasculaire-cerebral/anevrisme-de-l-aorte-abdominale>, 2019.
- [2] Francisco S. Lozano Sánchez. Treatment of aneurysms of the abdominal aorta, improvements and evidences, 2022.
- [3] Le manuel msd, version pour professionnels de la santé. revue générale des anévrismes de l'aorte. <https://www.msdmanuals.com/fr/professional/troubles-cardiovasculaires/maladies-de-aorte-et-de-ses-branches/revue-g%C3%A9n%C3%A9rale-des-an%C3%A9vrismes-de-aorte>, 2022.
- [4] kancaal1. Deep learning tutorial for beginners. <https://www.kaggle.com/code/kanncaa1/deep-learning-tutorial-for-beginners/notebook>, 2019.
- [5] kancaal1. Convolutional neural network (cnn) tutorial. <https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial>, 2020.
- [6] Maxime Dassen, Andreas de Rijcke, and M. Torians. Deep learning x-ray md. <https://www.kaggle.com/code/maximedassen/deep-learning-x-ray-md>, 2021.
- [7] doxygen. Image processing in opencv. https://docs.opencv.org/3.4/d2/d96/tutorial_py_table_of_contents_imgproc.html, 2023.
- [8] CIBC. Seg3d, segmentation image processing. <https://www.sci.utah.edu/cibc-software/seg3d.html>, 2023.
- [9] Paul A. Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C. Gee, and Guido Gerig. Itk-snap. user-guided 3d active contour segmentation of anatomical structures : Significantly improved efficiency and reliability. <http://www.itksnap.org/pmwiki/pmwiki.php?n>Main.HomePage>, 2006.
- [10] Fedorov A., Beichel R., Kalpathy-Cramer J., Finet J., Fillion-Robin J-C., Pujol S., Bauer C., Jennings D., Fennessy F.M., Buatti J. Sonka M., Aylward S.R., Miller J.V., Pieper S., and Kikinis R. 3d slicer as an image computing platform for the quantitative imaging network. magnetic resonance imaging. <https://www.slicer.org/>, 2012.
- [11] Ángel Díaz Carral. Modelo mecánico para la detección de la pared externa de la aorta en imágenes de rayos x mediante sistemas de partículas para el modelado de aaa. <https://uvadoc.uva.es/handle/10324/18856>, 2016.
- [12] kmader. Aorta as circle segmentation. <https://www.kaggle.com/code/kmader/aorta-as-circle-segmentation/notebook>, 2018.
- [13] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow : Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [14] Marc Päpper. Deep learning on medical images with u-net. <https://www.paepper.com/blog/posts/deep-learning-on-medical-images-with-u-net/>, 2019.
- [15] Wikipédia. U-net — wikipédia. <http://fr.wikipedia.org/w/index.php?title=U-Net&oldid=203589903>, 2023. [En ligne ; Page disponible le 22-avril-2023].
- [16] SOURCEFORGE. Nearly raw raster data. <https://teem.sourceforge.net/nrrd/>, 2023.
- [17] Microsoft. Normalize data component. <https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/normalize-data?view=azureml-api-2>, 2021.

- [18] Google Developers. Normalization - machine learning. <https://tinyurl.com/googleDevelopersNormalisation>, 2022.
- [19] KSV Muralidhar. Learning curve to identify overfitting and underfitting in machine learning. <https://towardsdatascience.com/learning-curve-to-identify-overfitting-underfitting-problems-133177f38df5>, 2021.
- [20] The 365 Team. Overfitting vs. underfitting : What is the difference ? <https://365datascience.com/tutorials/machine-learning-tutorials/overfitting-underfitting/>, 2023.
- [21] Xiao Wei Xu. Github dataset aortas dissectionnées. dataset_type-b-aortic-dissection. https://github.com/Xiaoweixu/Dataset_Type-B-Aortic-Dissection, 2021.
- [22] Ibrahim Harcha, Ilona Lazrak, Samuel Mallet, and Rosa Sabater Rojas. Link github. <https://github.com/SuperMuel/PP2-aorte-segmentation>, 2023. Accès aux codes dataPreparation.py, testing.py et aux fichiers .ipynb utilisés sur Google Collab. Aussi fichiers .h de quelques modèles.
- [23] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net : Learning dense volumetric segmentation from sparse annotation, 2016.
- [24] Zhuotun Zhu, Chenxi Liu, Dong Yang, Alan Yuille, and Daguang Xu. V-nas : Neural architecture search for volumetric medical image segmentation, 2019.
- [25] Fedorov A., Beichel R., Kalpathy-Cramer J., Finet J., Fillion-Robin J-C., Pujol S., Bauer C., Jennings D., Fennessy F.M., Sonka M., Buatti J., Aylward S.R., Miller J.V., Pieper S., and Kikinis R. 3d slicer as an image computing platform for the quantitative imaging network. magnetic resonance imaging. <https://www.slicer.org/>, 2012. PMID : 22770690. PMCID : PMC3466397.
- [26] Anato Imagen. Vidéo 1 youtube pour l'utilisation de 3dslicer. https://www.youtube.com/watch?v=AFBSipzXiXo&ab_channel=AnatoImagen, 2023.
- [27] Jan Egger. Vidéo 2 youtube pour l'utilisation de 3dslicer. https://www.youtube.com/watch?v=5_673cHMBiY&t=6s&ab_channel=JanEgger, 2022.
- [28] Jan Witowski. Vidéo 3 youtube pour l'utilisation de 3dslicer. https://www.youtube.com/watch?v=ZRYMITzwg8g&ab_channel=JanWitowski, 2019.
- [29] DigitalSreeni. Playlist youtube d'introduction à u-net. https://www.youtube.com/watch?v=azM57JuQpQI&list=PLZs0BAyNTZwbR08R959iCvYT3qzhxvGOE&ab_channel=DigitalSreeni, 2020. Playlist de plusieurs vidéos Youtube. Recommandation de regarder jusqu'au vidéo Part 6 (Running the code and understanding results). Cette ressource contient des codes de livre accès sur Github très utiles. "This channel walks you through the entire process of learning to code in Python ; all the way from basics to advanced machine learning and deep learning. The primary emphasis will be on image processing and other relevant functionality.".
- [30] AugmentedStartups. Training u-net on original dataset : A step-by-step tutorial. https://www.youtube.com/watch?v=n4_ZuntLGjg&ab_channel=AugmentedStartups, 2023.
- [31] Elias Rhouzlane. 5 minute teaser presentation of the u-net : Convolutional networks for biomedical image segmentation. https://www.youtube.com/watch?v=81AvQQnpG4Q&ab_channel=EliasRhouzlane, 2017.

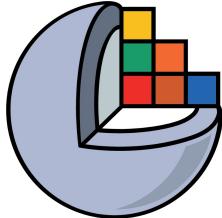
Références des images

- [1] Wikipedia. Anévrisme de l'aorta abdominale. <https://fr.wikipedia.org/wiki/Ane>, 2023.
- [2] médecine et recherche cardiovasculaire The Beat, nouveautés en matière de santé. La tomodensitométrie pour détecter des porteurs asymptomatiques de la covid-19 ?, 2020.
- [3] Scientific Figure on ResearchGate. Ischémie mésentérique et thrombi aortiques. https://www.researchgate.net/figure/CT-scan-abdominal-avec-produit-de-contraste-au-temps-arteriel-on-note-labsence_fig1_242694855.
- [4] Ibrahim Harcha, Ilona Lazrak, Samuel Mallet, and Rosa Sabater Rojas. Images du projet, 2023.

- [5] datafuture. Réseau de neurones.
- [6] Christine Fernandez-Maloigne and Rémy Guillevin. L'intelligence artificielle au service de l'imagerie et de la santé des femmes. <https://www.sciencedirect.com/science/article/abs/pii/S177698171930063X>.
- [7] University of Freiburg. Architecture u-net. <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>, 2015.
- [8] Aditta Das Nishad. Convolution neural network. <https://www.kaggle.com/general/171197>.
- [9] Muhamad Yani, S Irawan, and Casi Setianingsih. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics : Conference Series*, 1201 :012052, 05 2019.
- [10] Adrian Rosebrock. 3 ways to create a keras model with tensorflow 2.0. <https://tinyurl.com/Keras-model>.

9 Annexes

9.1 3DSlicer



3D Slicer⁹ est une plateforme logicielle gratuite et open source pour l'analyse et la visualisation d'images médicales. Une des applications de 3D Slicer est de segmenter des structures anatomiques à partir d'images médicales, comme l'aorte à partir de scanners tomographiques (CT) ¹⁰.

Il existe différentes méthodes pour segmenter l'aorte en utilisant 3D Slicer, en fonction de la qualité, de la résolution et de la précision souhaitée de l'image. Une des méthodes les plus rapides est d'utiliser l'effet Fast marching dans le module Segment Editor. Cette méthode permet à l'utilisateur de placer des graines à l'intérieur de la région aortique et de faire croître le segment en fonction des seuils d'intensité et de distance. Une autre méthode qui peut atteindre une précision plus élevée est d'utiliser l'effet Grow from seeds, qui nécessite de placer des graines à la fois à l'intérieur et à l'extérieur de la région aortique et de laisser l'algorithme optimiser la frontière en fonction des caractéristiques d'intensité et de forme. Les deux méthodes peuvent bénéficier du masquage des régions non pertinentes, comme les os et les poumons, en utilisant d'autres outils de segmentation comme Threshold ou Surface cut.

Après avoir segmenté l'aorte, l'utilisateur peut la visualiser en vue 3D, l'exporter comme un fichier maillé (comme STL), ou effectuer une analyse plus poussée en utilisant d'autres modules ou extensions dans 3D Slicer. Par exemple, on peut utiliser le module Extract Centerline dans l'extension SlicerVMTK pour calculer la ligne centrale et le diamètre de l'aorte le long de sa longueur. Cela peut fournir des informations utiles pour mesurer les anévrismes aortiques, les sténoses ou les tortuosités.

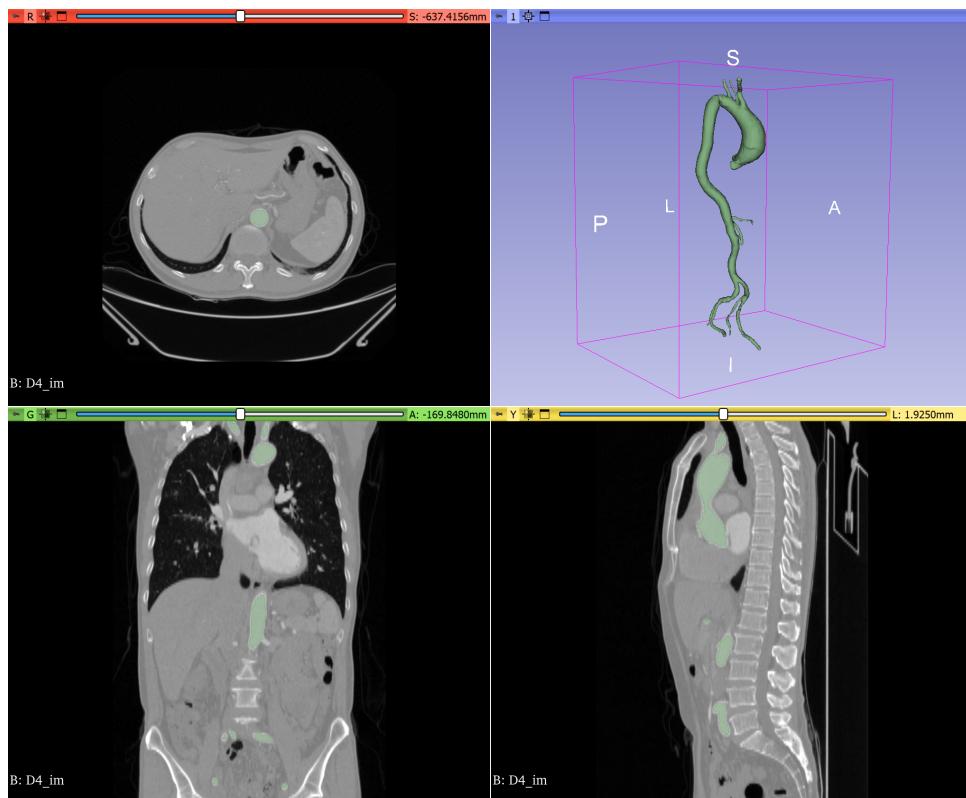


FIGURE 29 – Visualisation de D4_im.nrrd avec D4_seg.nrrd [4]

9. Informations résumées de la référence officielle [25].

10. Vidéos Youtube de 3DSlicer [26, 27, 28].

9.2 Premier programme : dataPreparation.py

9.2.1 Explication et utilisation

Le programme dataPreparation.py a été le premier programme que nous avions codé. Il a été utile pour se familiariser avec tout ce que nous avons utilisé sur le projet final :

- savoir comment ouvrir des fichiers .nrrd et .nii,
- indiquer les paths où se trouvaient les images dans nos répertoires,
- créer des dossiers pour sauvegarder les images au format .png,
- plot les images et indiquer la couche et les coordonnées X, Y, Z.
- premier approche à la normalisation d'images, mais fait avec des valeurs -1000 et 2000 triés complètement au hasard...

Nous avons regardé plusieurs vidéos Youtube pendant les mois de février-mars pour comprendre comment accéder aux données des axes des images avec Python¹¹.

Voici un tutoriel de comment utiliser le code dataPreparation.py en utilisant Visual Studio Code :

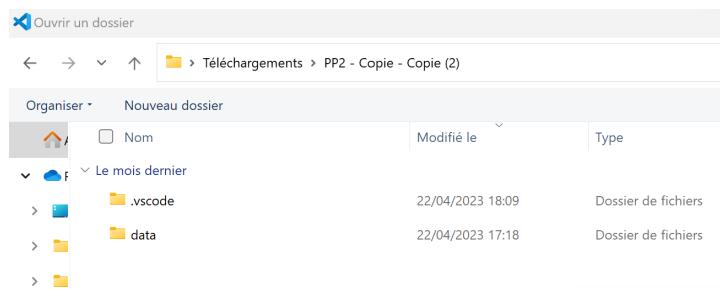


FIGURE 30 – Ouvrir le dossier contenant le répertoire nommée "data" [4]

```

EXPLORATEUR      ...   dataPreparation.py X
PP2 - COPIE - COPIE (3)
  data
  test
  img
    R5_im.nii
    R16_im.nii
    R17_im.nii
    R18_im.nii
  masks
    R5_seg.nii
    R16_seg.nii
    R17_seg.nii
    R18_seg.nii
  train
    img
    masks
    dataPreparation.py
    training.py

```

FIGURE 31 – L'arborescence de la localisation et les noms des images [4]

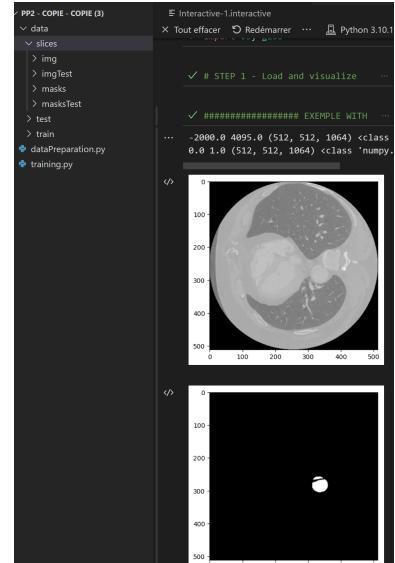


FIGURE 32 – Création du dossier "slices" avec la bonne arborescence et premier plot [4]

11. Vidéos Youtube d'introduction aux réseaux U-Net [29, 30, 28, 31].

```

    ✓ import time ...
    ✓ ##### SAVE SLICES AND MASKS IN THE REPOSITORIES
...
Output exceeds the size limit. Open the full output data in a text
data/train/img\D1_im.nii (512, 666, 251) 1429 0.0 1.0
512 666 251
Slicing Z:
[+] Slice saved: data/slices/img\R0-slice250_z.png
data/train/img\D1_im.nii, 251 slices created

data/train/img\D2_im.nii (512, 666, 186) 1364 0.0 1.0
512 666 186
Slicing Z:
[+] Slice saved: data/slices/img\R1-slice185_z.png
data/train/img\D2_im.nii, 186 slices created

data/train/img\D3_im.nii (512, 666, 175) 1353 0.0 1.0
512 666 175
Slicing Z:
[+] Slice saved: data/slices/img\R2-slice174_z.png
data/train/img\D3_im.nii, 175 slices created

data/train/img\D4_im.nii (512, 666, 180) 1358 0.0 1.0
512 666 180
Slicing Z:
[+] Slice saved: data/slices/img\R3-slice179_z.png
data/train/img\D4_im.nii, 180 slices created

data/train/img\D5_im.nii (512, 666, 176) 1354 0.0 1.0
...
[+] Slice saved: data/slices/masksTest/img\T21-slice1102_z.png
data/test/masks\R5_seg.nii, 1103 slices created

Temps pour extraire toutes les images : 705.7907 s.

```

FIGURE 33 – Extraction des images du format .nii au format .png dans le dossier "slices" avec seulement SLICE_Z=True. Le temps total s'affiche à l'écran [4]

Les datasets séparés sur les dossiers *test* et *train* utilisés dans la section suivante pour le training et le testing sont les suivants, avec 75.22% training (13666 images) et 24.77% testing (4501 images).

Dossier train :
D1,D2,D3,D4,D5
K1,K2,K3,K4,K6,K7,K8,K9,K12,K13,K14,K15,K17,K18
R1,R2,R3,R4,R6,R11,R12,R13,R14,R15

Dossier test : les autres
D6,D7,D8,D9,D10,D11,D12,D13,D14,D15,D16,D17,D18
K10,K11,K16,K19,K20
R5,R16,R17,R18

9.3 Deuxième programme : testing.py

Maintenant, nous ouvrons le fichier nommée *testing.py* et nous l'exécutons. Il faut se placer dans le même répertoire que la section précédente. Nous recommandons ici de regarder le code de la source Github avant de lire cette section pour se familiariser avec les parties du programme.

Ce programme a le but d'utiliser les images au format .png extraites des images .nnrrd ou .nii dans la section précédente avec l'exécution du programme *dataPreparation.py*. Dans la partie antérieure, nous avons séparé les images dans deux dossiers, *test* et *training*, en nous assurant qu'ils ne contiennent pas les mêmes images. Etant donnée que c'est recommandé d'utiliser 30% pour le test et 70% pour le training (cela peut varier, ce n'est pas un pourcentage fixe), nous avons 13666 images CT-Scan de l'axe Z pour le training et 4501 images pour le test.

Tout le programme est commenté et séparé par des titres, en donnant une définition des constantes utilisés et avec des noms pertinents pour les fonctions.

En premier, nous définissons les globales. Cette partie est très importante car indiquent la taille de l'image et le batch_size du training. La taille de l'image fera varier la qualité des images. Nos images au format .png ont une taille de 512x512. Dans le modèle, nous allons les redimensionner afin que le U-NET s'exécute plus vite selon les globales définies et nous ferons varier la taille du batch et des epochs. Ce programme nous permettra :

- créer les fonctions pour afficher les images qu'il trouvera sur les paths indiqués,
- créer les fonctions génératrices des images pour parcourir les dossiers et pour les redimensionner avec les paramètres définis dans les globales,
- compter les images des dossiers et indiquer combien d'images nous utilisons pour le training et pour le test,
- créer le modèle U-NET avec un nombre de niveaux pour paramètre,
- exécuter le modèle et le sauvegarder avec un nom particulier qui définit le nombre d'epochs, le batch_size, l'image height et l'image width utilisés,
- load le modèle si il existe déjà afin de ne pas avoir besoin d'exécuter le programme à nouveau,
- afficher les prédictions du modèle.

A nouveau, nous nous sommes basés en bonne partie des ressources libres Github et vidéos Youtube pour réaliser ce code. Nous avons constaté que les programmes U-Net ont tous une similarité dans leur structure et qu'ils peuvent être réutilisés dans la majorité des réseaux de neurones en choisissant les bonnes paramètres et globales. D'où nous avons dissectionné le code avec des titres et des commentaires en expliquant ce qu'il fait chaque fonction et paramètre pour pouvoir les réutiliser sur un autre projet.

9.3.1 Définitions des paramètres

```
# %%
##### CONSTANTS DEFINITION #####
#####

SEED = 1000
BATCH_SIZE_TRAIN = 16          # tester differente val°
BATCH_SIZE_TEST = 16
NUM_OF_EPOCHS = 30

IMAGE_HEIGHT = 512            # tester differente val°
IMAGE_WIDTH = 512
IMG_SIZE = (IMAGE_HEIGHT, IMAGE_WIDTH)
```

FIGURE 34 – Définition des globales pour les images de taille 512x512 [4]

```
# %%
##### CONSTANTS DEFINITION #####
#####

SEED = 1000
BATCH_SIZE_TRAIN = 10          # tester differente val°
BATCH_SIZE_TEST = 10
NUM_OF_EPOCHS = 30

IMAGE_HEIGHT = 64              # tester differente val°
IMAGE_WIDTH = 64
IMG_SIZE = (IMAGE_HEIGHT, IMAGE_WIDTH)
```

FIGURE 35 – Globales pour redimensionner à taille 64x64 [4]

```
# %%
##### CONSTANTS DEFINITION #####
#####

SEED = 1000
BATCH_SIZE_TRAIN = 32          # tester differente val°
BATCH_SIZE_TEST = 32
NUM_OF_EPOCHS = 50

IMAGE_HEIGHT = 128             # tester differente val°
IMAGE_WIDTH = 128
IMG_SIZE = (IMAGE_HEIGHT, IMAGE_WIDTH)
```

FIGURE 36 – Globales pour redimensionner à taille 128x128 [4]

9.3.2 Modèle

```

Nombre total d'images pour le training: 13666
Nombre total d'images pour le test: 4501

✓ ##### U-NET DEFINITION - CONVOLUTIONS, ...
```
```
↻ ##### U-NET APPLICATION AND SAVING RESULTS ...
Model: "UNET-L3-F32"

Layer (type)          Output Shape       Param #  Connect
=====
input_1 (InputLayer)   [(None, 512, 512, 1  0
                      )]
conv2d (Conv2D)        (None, 512, 512, 32  320      ['input
                  )
conv2d_1 (Conv2D)      (None, 512, 512, 32  9248     ['conv2
                  )
max_pooling2d (MaxPooling2D) (None, 256, 256, 32  0
                               )
conv2d_2 (Conv2D)      (None, 256, 256, 64  18496     ['max_p
                  )
conv2d_3 (Conv2D)      (None, 256, 256, 64  36928     ['conv2
                  )
max_pooling2d_1 (MaxPooling2D) (None, 128, 128, 64  0
                               )
...
Total params: 517,153
Trainable params: 517,153
Non-trainable params: 0
```
```

```

FIGURE 37 – Implémentation fonction unet(3) : U-NET APPLICATION AND SAVING RESULTS [4]

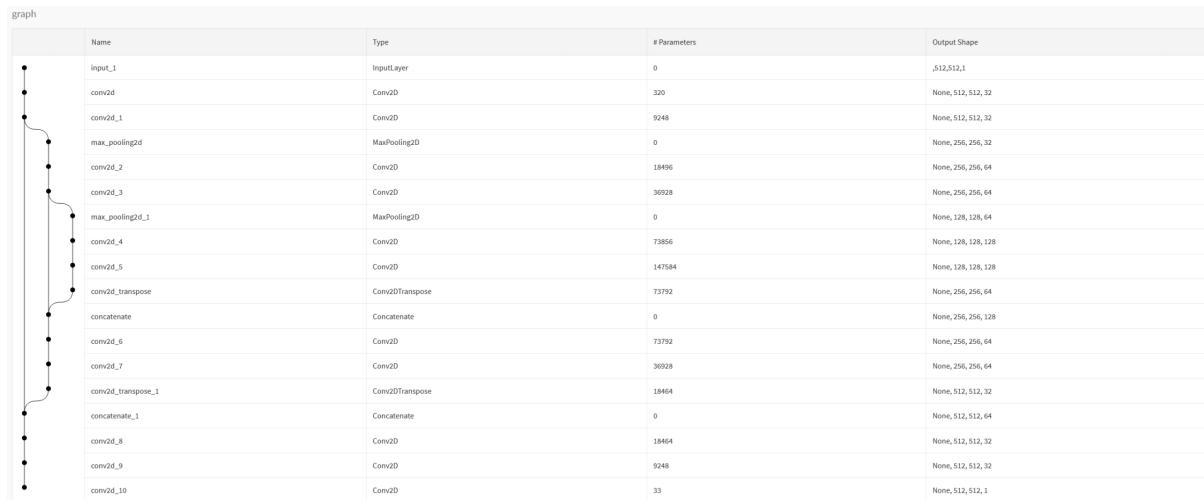


FIGURE 38 – Implémentation fonction unet(3) : U-NET APPLICATION AND SAVING RESULTS [4]

9.3.3 Plots des images CT-Scan

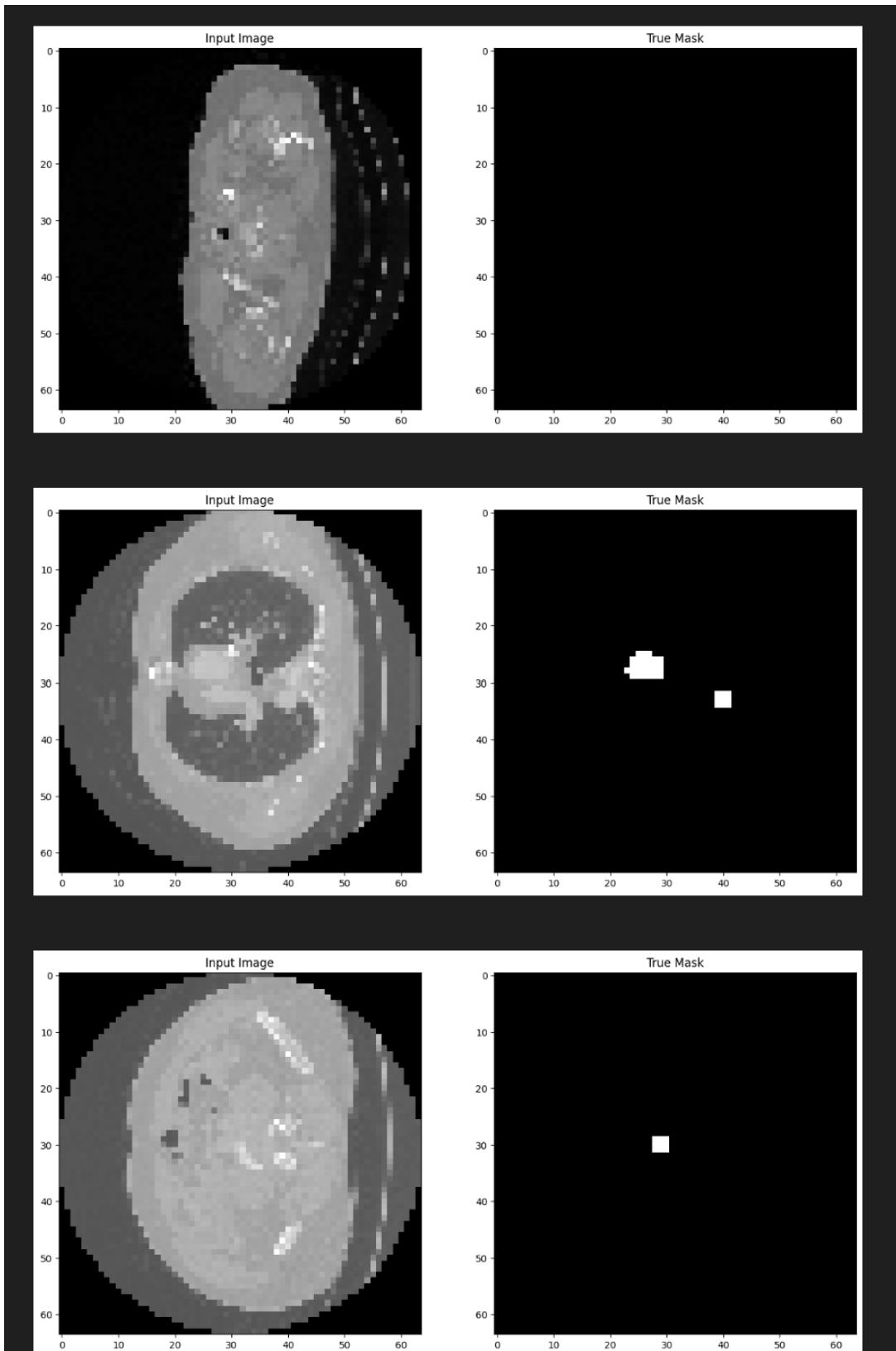


FIGURE 39 – Exemple de plot utilisant le generator avec images 64x64 [4]

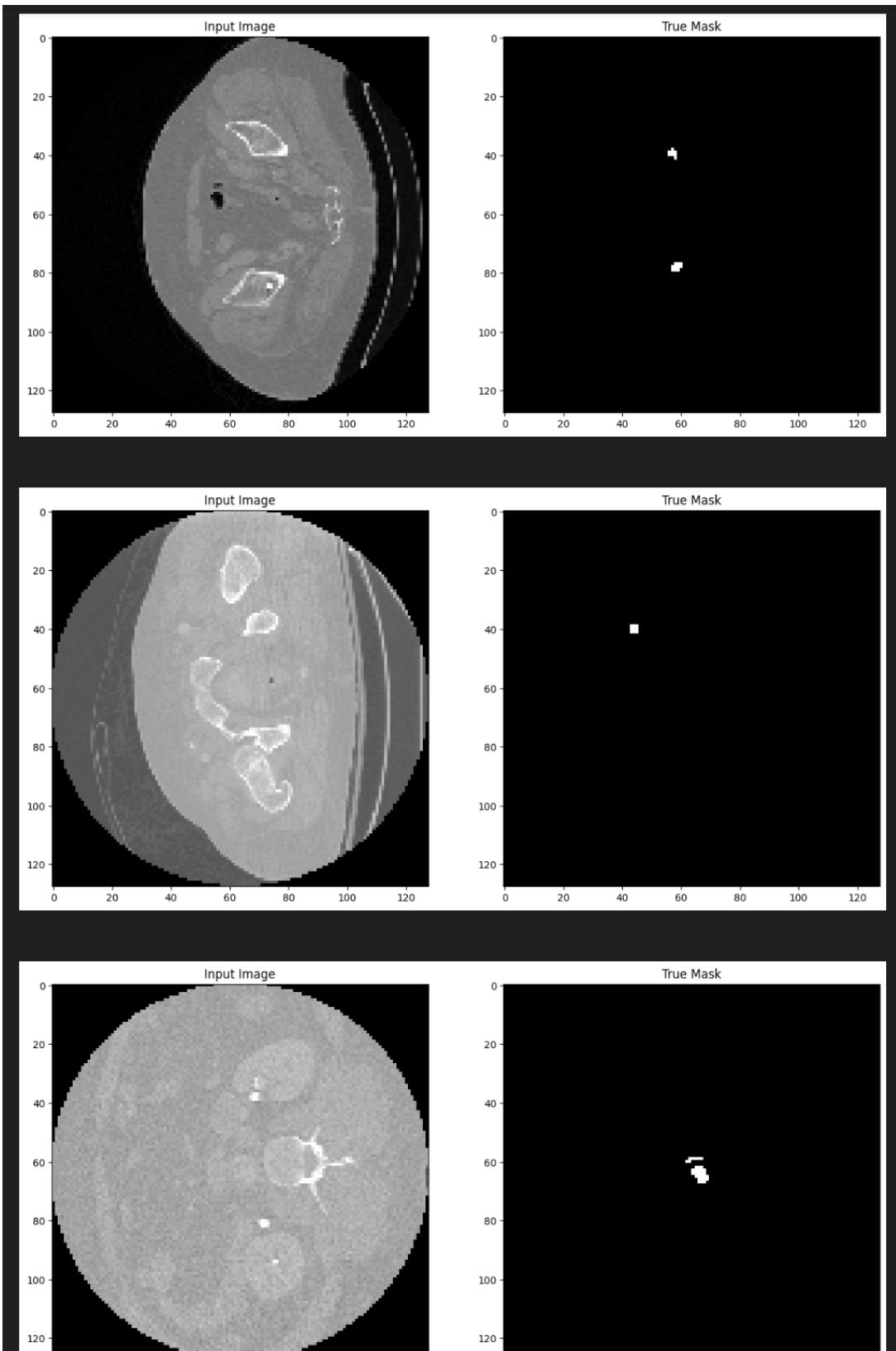


FIGURE 40 – Exemple de plot utilisant le generator avec images 128x128 [4]

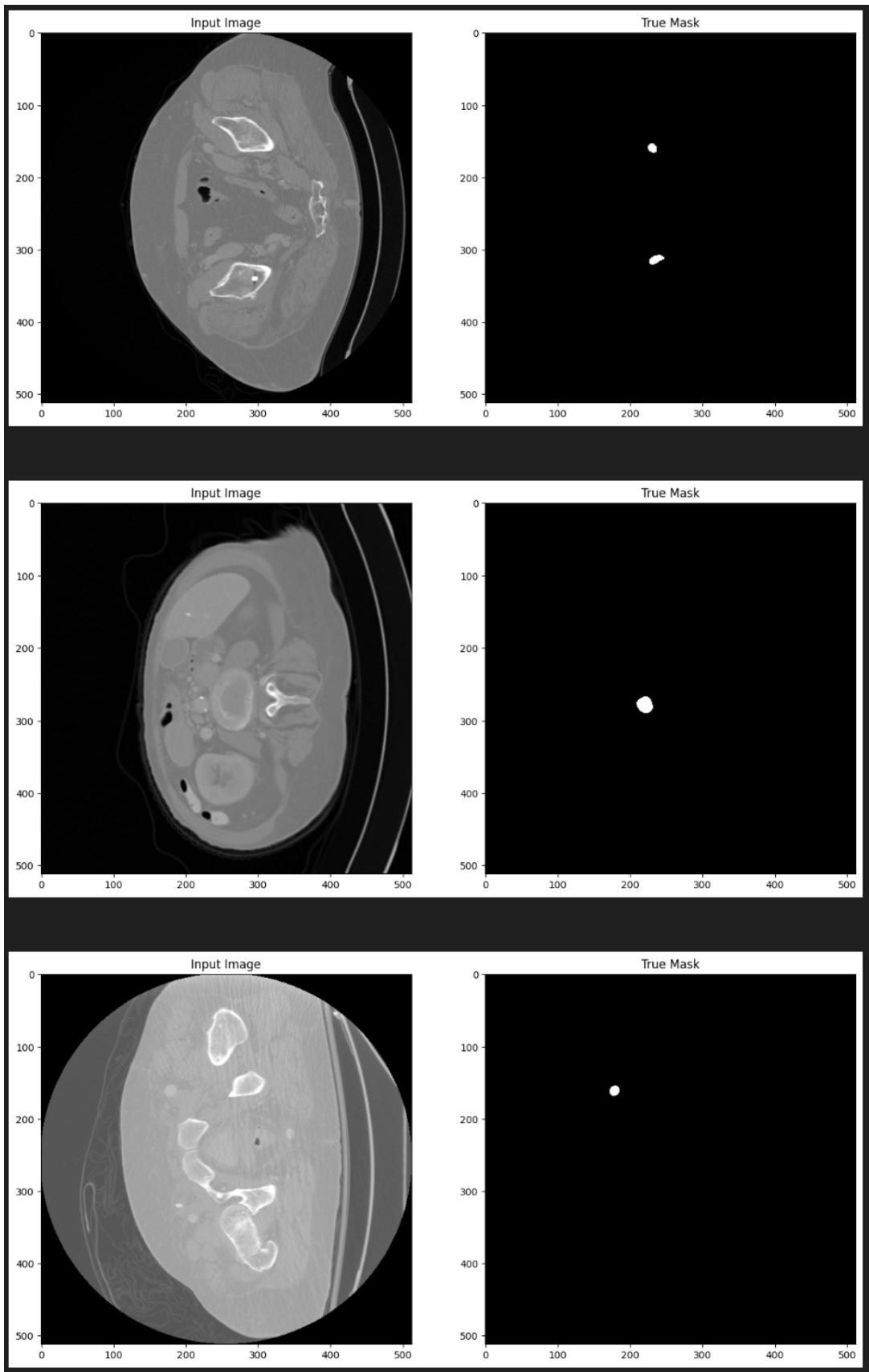


FIGURE 41 – Exemple de plot utilisant le generator avec images 512x512 [4].

Une fois que nous aurons exécuté le code, nous obtiendrons une sauvegarde des modèles au format .h5, nommés correctement afin de pouvoir les réutiliser et en indiquant les epochs et batch utilisés :

```
└─ .vscode
    └─ data
        └─ dataPreparation.py
        └─ donnees512x512.png
        └─ donnees128x128.png
        └─ graphModel128x128.png
        └─ graphModel512x512.png
        └─ training.py
        └─ trainingWithGraphs.py
        └─ UNET-13666totalTrainImages_2016totalTestImages_64height_64width_10batch_30epochs.h5
        └─ UNET-13666totalTrainImages_2016totalTestImages_128height_128width_32batch_50epochs.h5
        └─ UNET-13666totalTrainImages_2016totalTestImages_512height_512width_16batch_30epochs.h5
```

FIGURE 42 – Nom des modèles sauvegardés après l'exécution du code testing.py [4]

De cette manière, ce ne sera pas nécessaire de relancer le training, mais de changer les valeurs des globales pour celles qui apparaissent dans le nom du fichier .h5 et d'exécuter toutes les parties qui ont un titre en commentaire, sauf le titre *U-NET APPLICATION AND SAVING RESULTS*, et de commenter/decommenter le modèle choisi dans la partie *U-NET LOADING RESULTS IF U-NET ALREADY APPLIED BEFORE*.

9.3.4 Prédictions

Une fois que le modèle est chargé (`load_model`), nous pouvons voir les résultats de la prédiction. Pour cela, nous exécutons `show_prediction(test_generator, N)`, avec N étant le nombre de plots/couches que nous voulons observer :

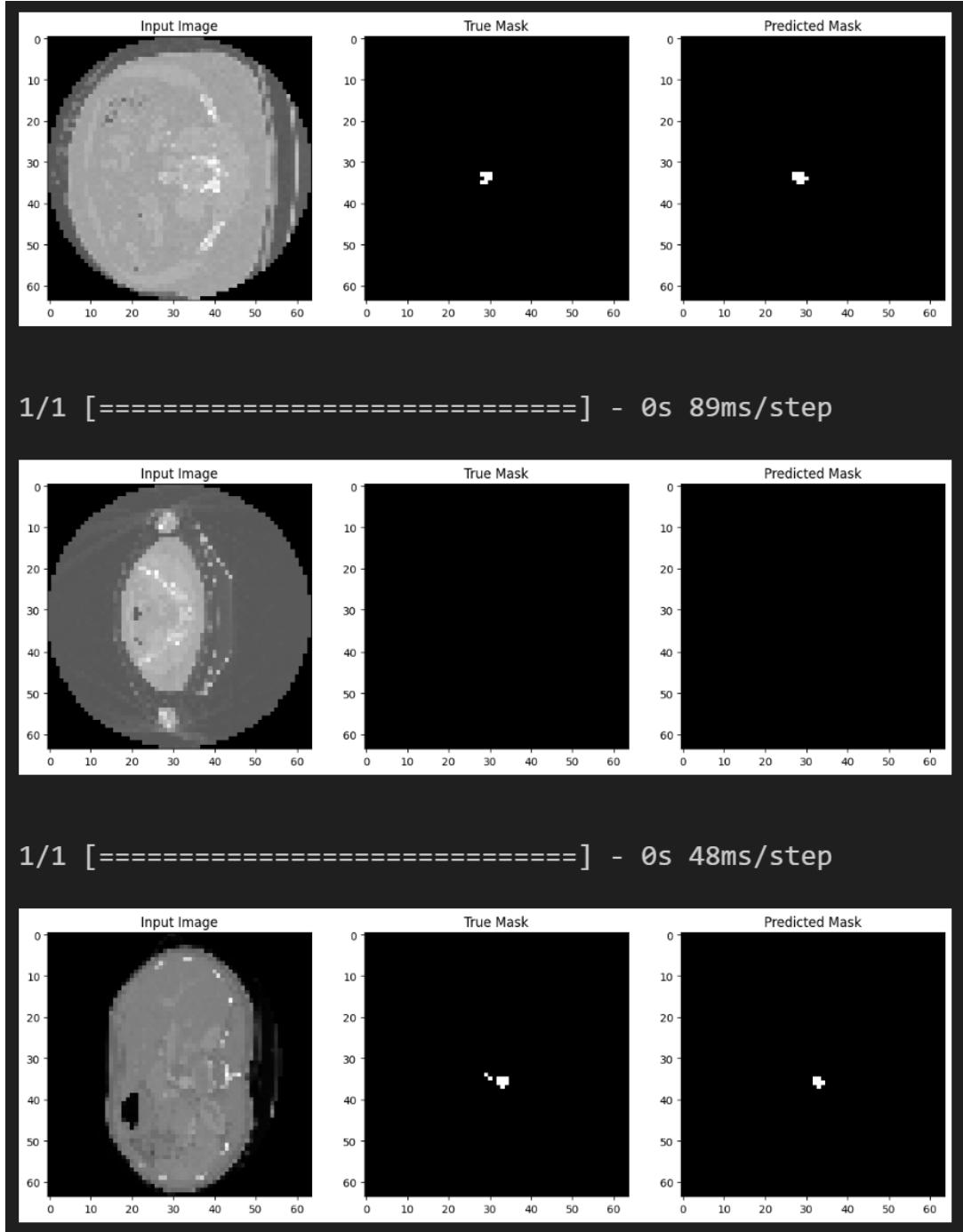


FIGURE 43 – Prédictions de 64x64 [4]

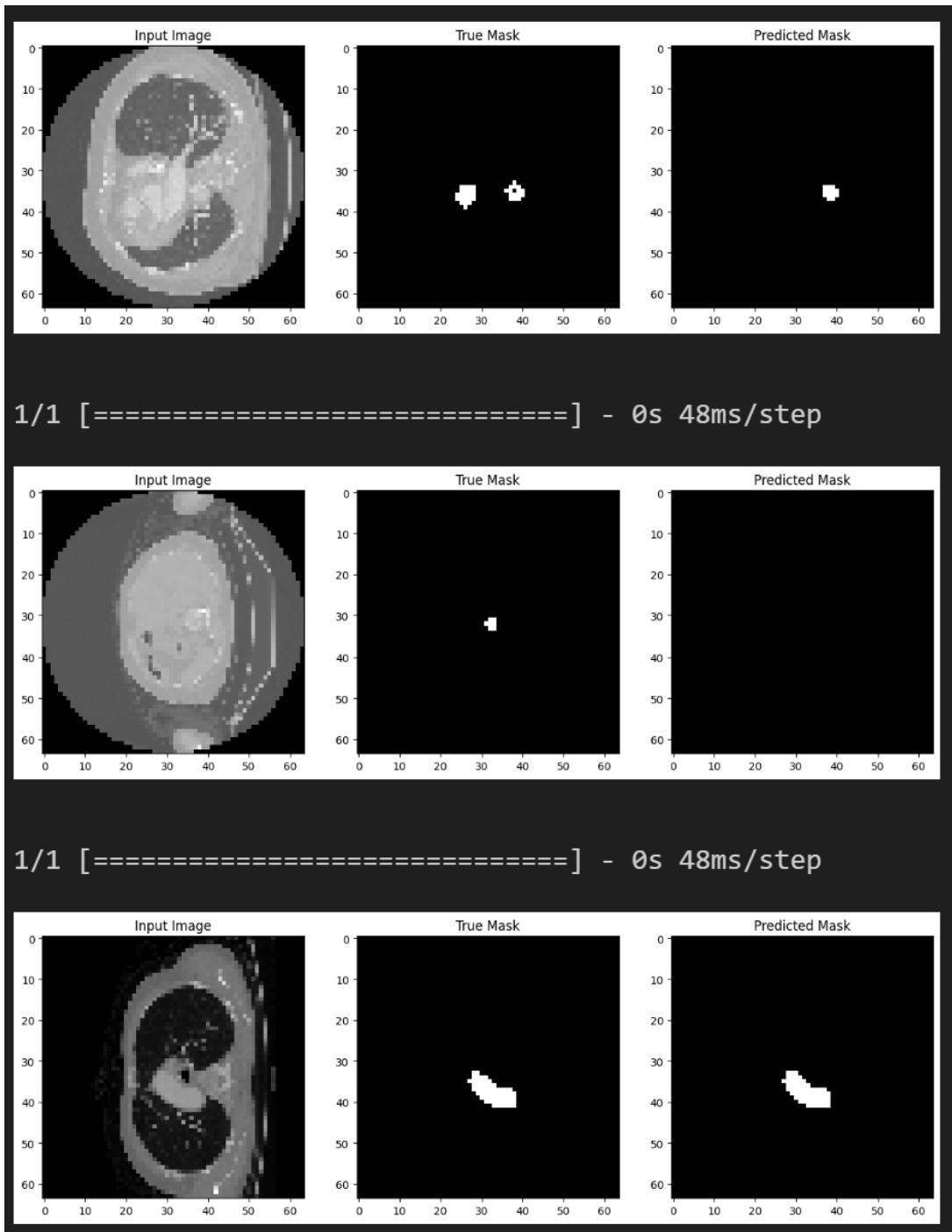
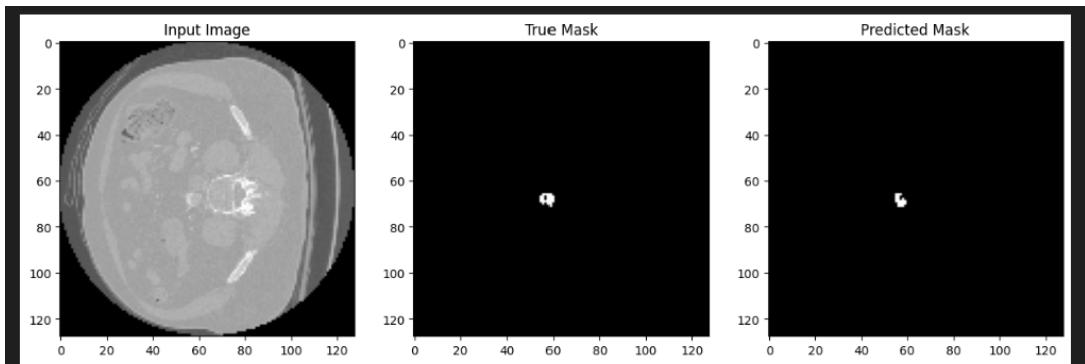
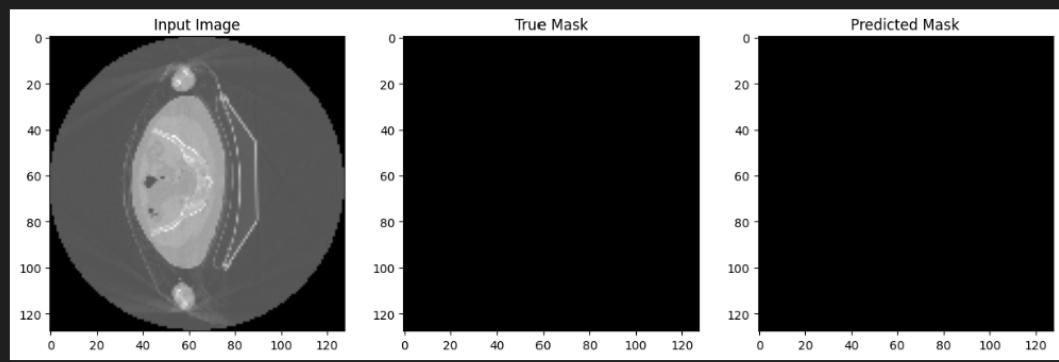


FIGURE 44 – Prédictions de 64x64 [4]



1/1 [=====] - 0s 102ms/step



1/1 [=====] - 0s 72ms/step

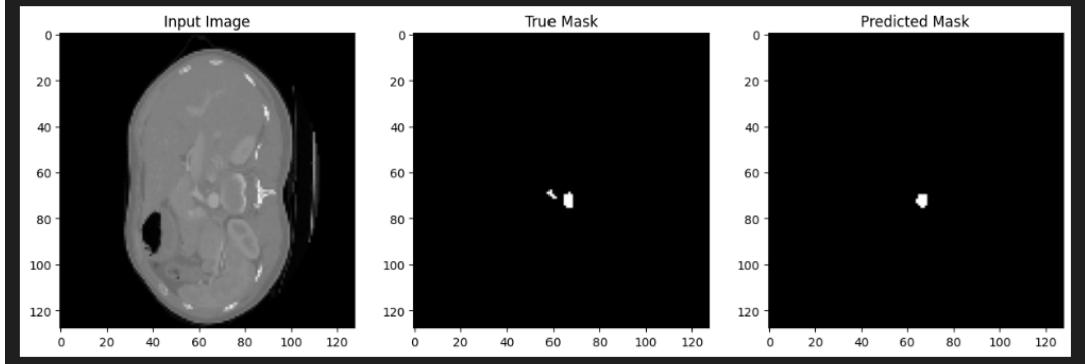
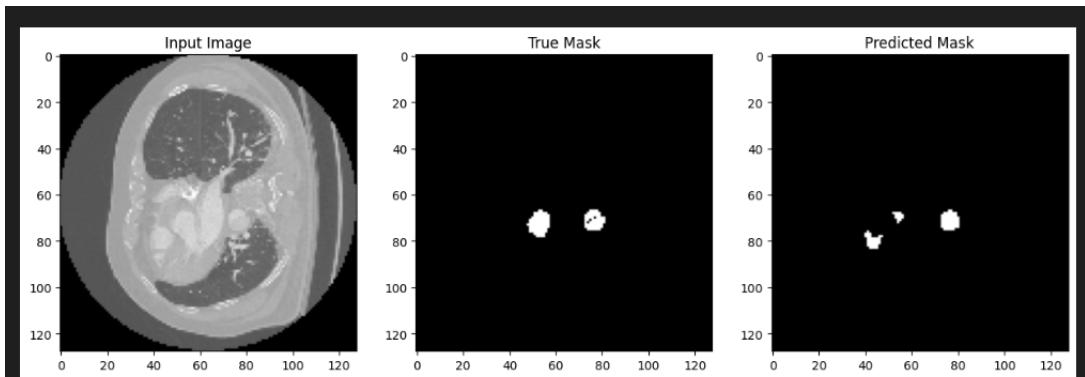
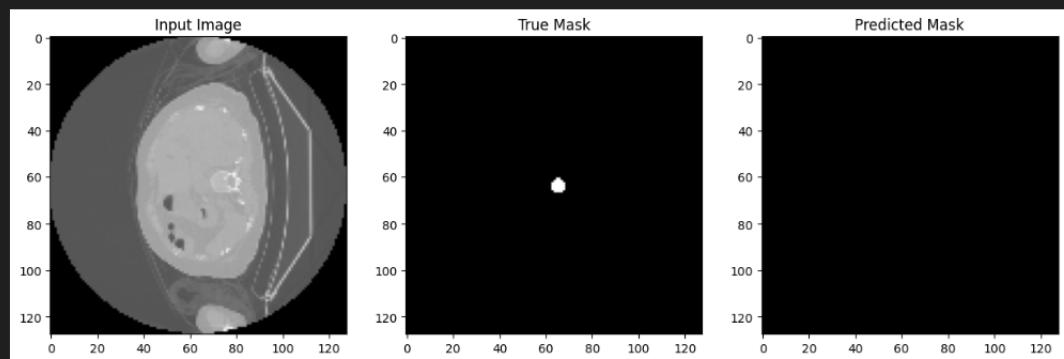


FIGURE 45 – Prédictions de 128x128 [4]



1/1 [=====] - 0s 72ms/step



1/1 [=====] - 0s 73ms/step

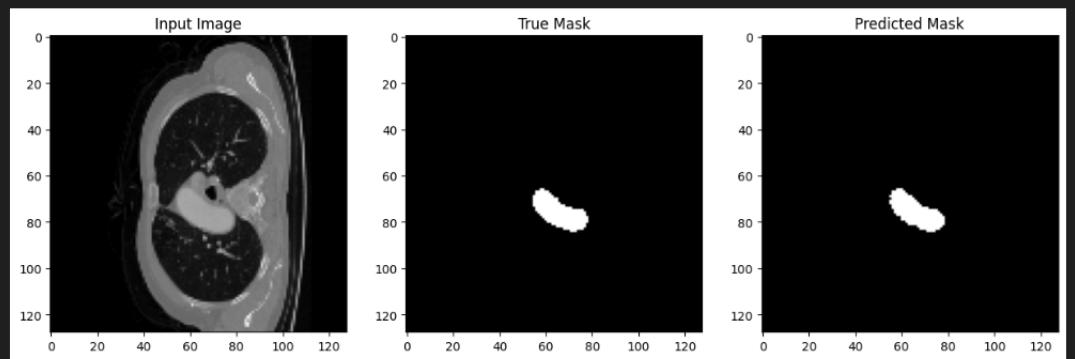
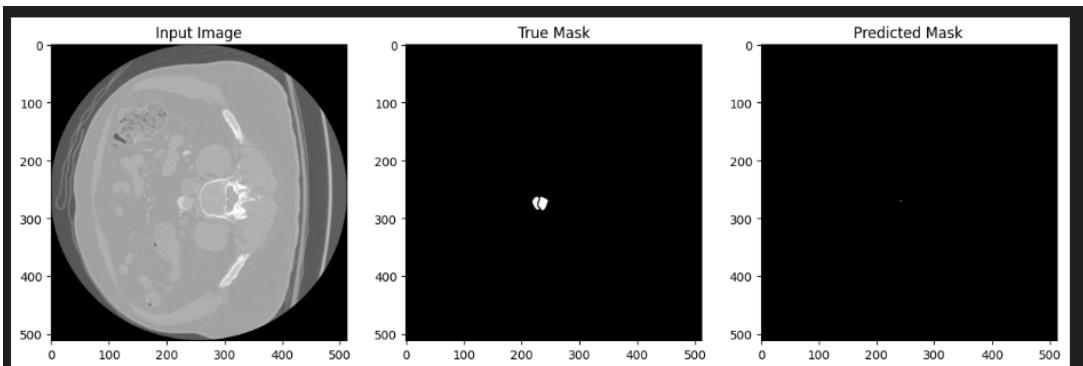
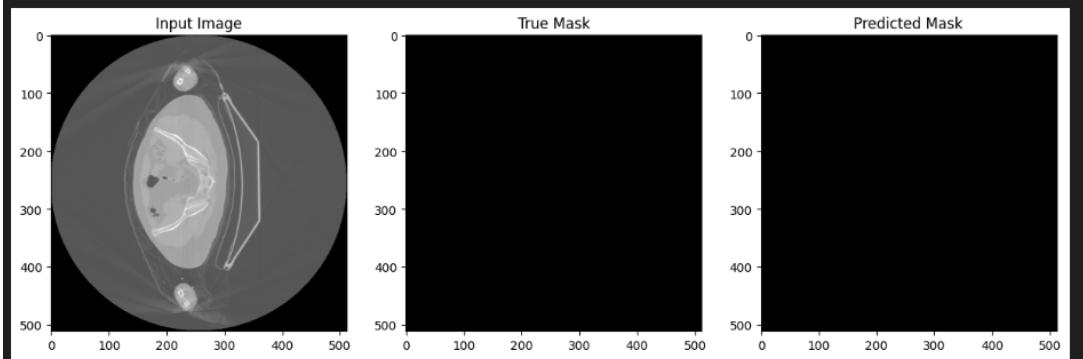


FIGURE 46 – Prédiction de 128x128 [4]



1/1 [=====] - 1s 595ms/step



1/1 [=====] - 1s 669ms/step

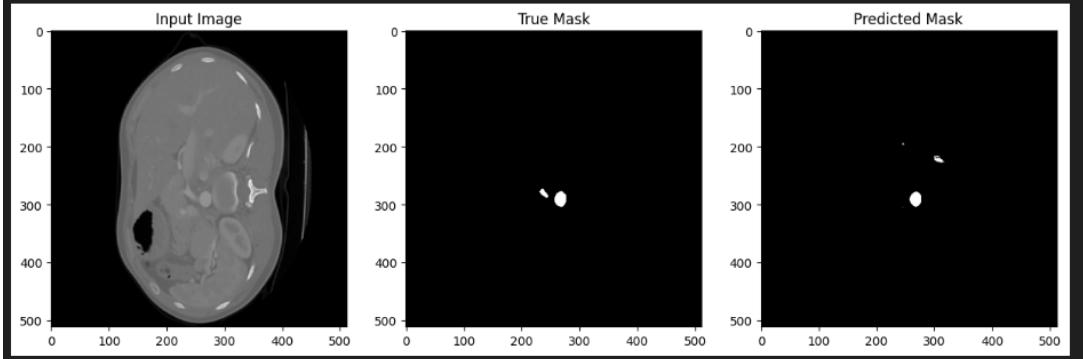
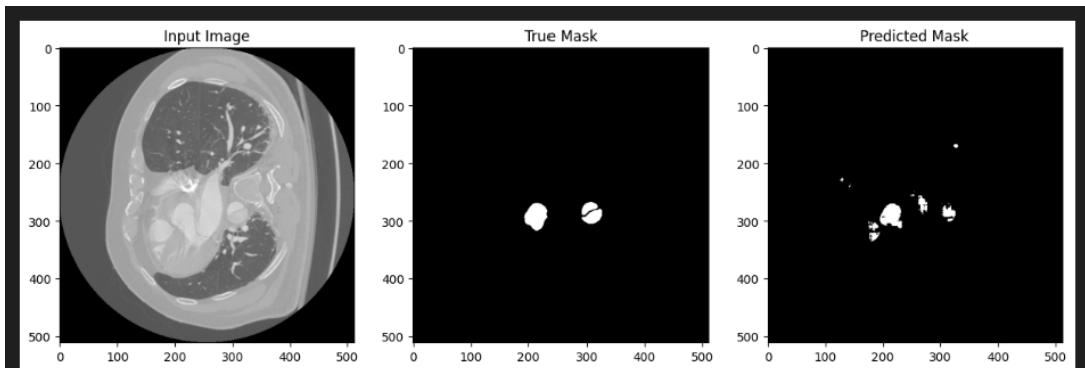
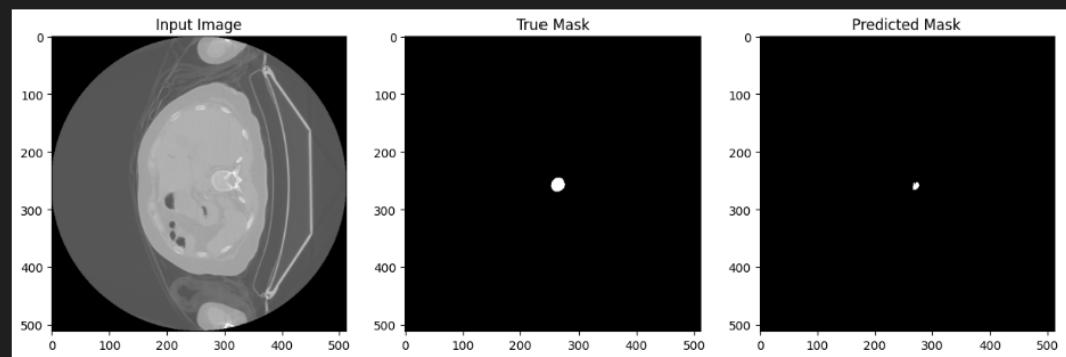


FIGURE 47 – Prédiction de 512x512 [4]



1/1 [=====] - 1s 555ms/step



1/1 [=====] - 1s 592ms/step

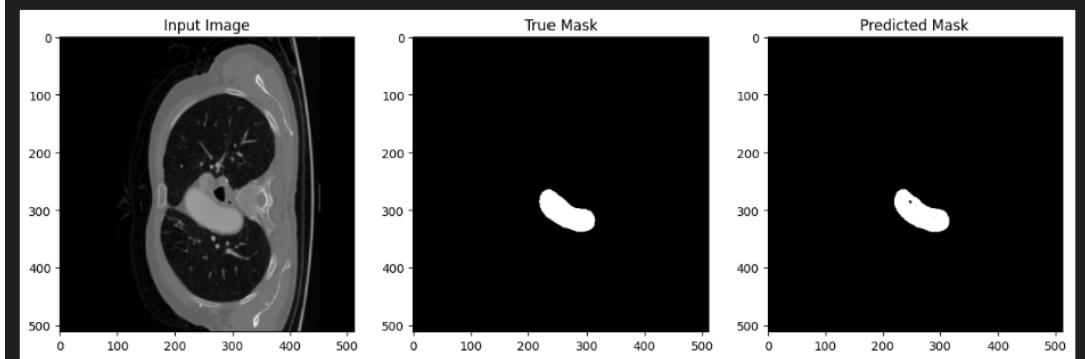


FIGURE 48 – Prédiction de 512x512 [4]

9.3.5 Résultats

Voici les résultats loss-accuracy des modèles de taille 128, 256 et 512 :

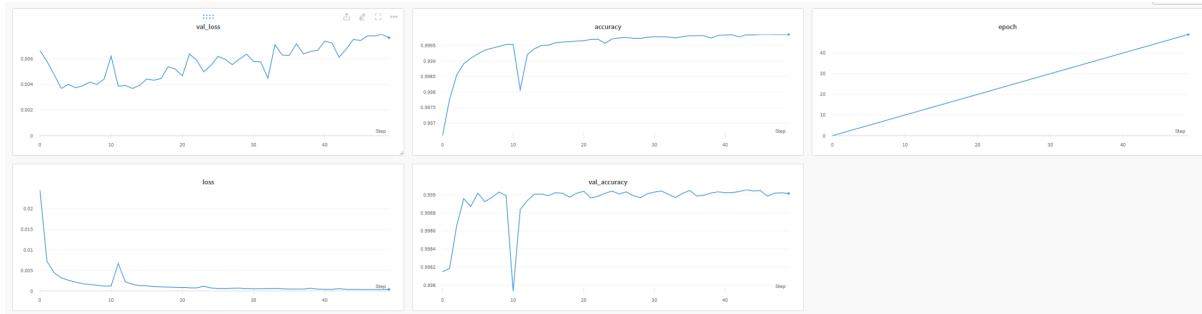


FIGURE 49 – Globales : 32 batch_size, 50 epochs, 128x128 [4]

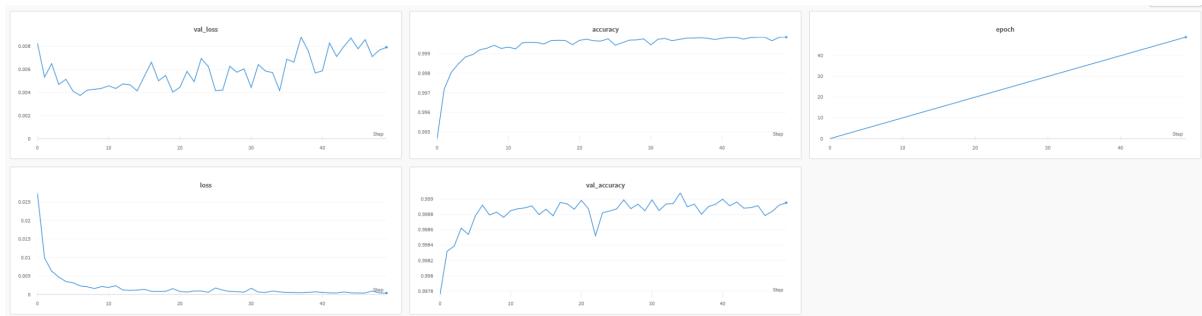


FIGURE 50 – Globales : 32 batch_size, 50 epochs, 256x256 [4]

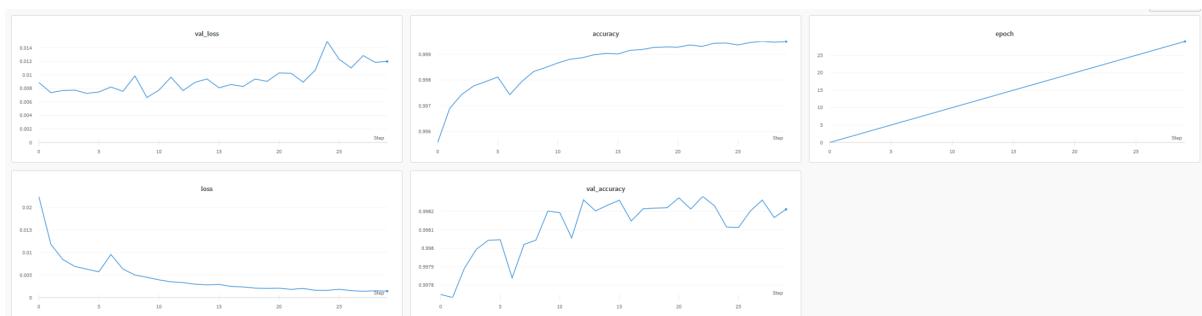


FIGURE 51 – Globales : 16 batch_size, 30 epochs, 512x512 [4]

9.3.6 Overfit et curiosités

L'overfit, c'est le fait que le modèle est tellement entraîné sur le dataset qu'il ne reconnaissse que les images du dataset en question, et qu'il échoue sur tout autre donnée ne provenant pas du dataset sur lequel il a été entraîné.

En essayant différents valeurs pour les paramètres et en pensant que les images de taille 512x512 avaient besoin de plus d'epochs vu la taille des images, nous avons mis 100 epochs. Cela a fait que le modèle soit overfit comme nous pouvons le constater sur les graphes avec la loss du validation set (val_loss) qui n'arrête pas d'augmenter, alors que celle du training set diminue (loss) :

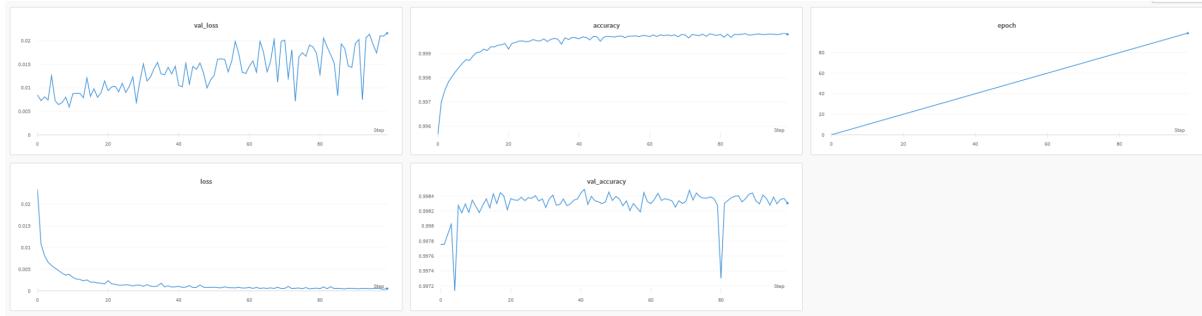


FIGURE 52 – Globales : 16 batch_size, 100 epochs, 512x512 [4]

Si nous utilisons le générateur `train_generator` sur `show_prediction(generator, 10)`, nous allons observer que la prédiction marque exactement où se trouve la vraie segmentation/mask.

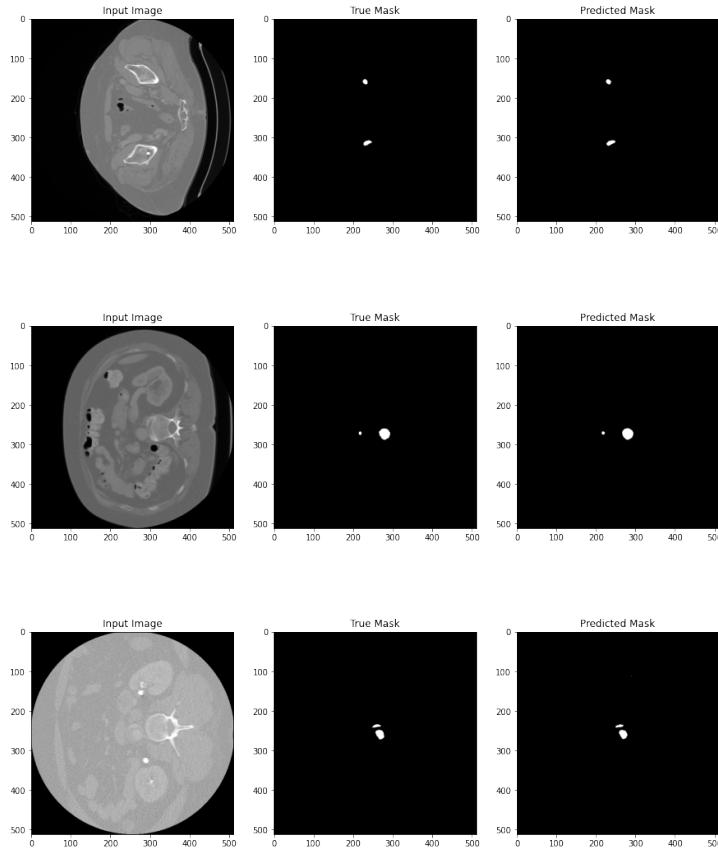


FIGURE 53 – Plot des prédictions utilisant images du training set [4]

Toujours avec les images 512x512 en lançant la fonction `unet(8)` par curiosité et voir si les résultats s'améliorent sans faire un overfit comme avec 100 epochs, l'ordinateur n'arrive pas à le supporter et il a un graphe aussi grand (plus de 512 millions de paramètres) qu'il est illisible sur le format .png et donc il faut le sauvegarder au format .json. On parle de données de 6GB ou plus. Il était si grand que le graph illustrant le loss-accuracy n'a pas de courbe, mais des droites :

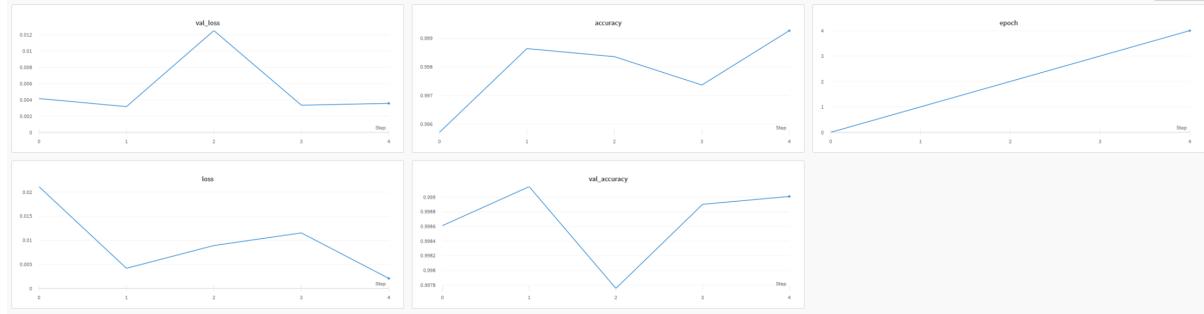


FIGURE 54 – `unet(8)` avec images 512x512 [4]

9.4 PROGRAMMES FINAUX : Google Colab

9.4.1 Extrait de code de predict_and_reconstruct.ipynb - Explication des seuils

```
1 def predict_3D_mask(normalized_vol: np.ndarray, model, model_shape=(512, 512), verbose
2 =0) -> np.ndarray:
3     """
4     Given a normalized 3D volume, predicts a 3D mask using the provided model.
5
6     This function resizes each 2D slice of the input volume to match the model's input
7     shape, predicts the mask using the model,
8     and then resizes the predicted mask back to the original slice shape. Note that the
9     returned mask is not a binary mask; it
10    contains continuous values between 0 and 1 representing the model's prediction
11    confidence.
12
13    Args:
14        normalized_vol (np.ndarray): A 3D numpy array representing the normalized input
15        volume.
16        model : A trained model that takes 2D slices as input and predicts the
17        corresponding mask.
18        model_shape (tuple): The shape (height, width) that the 2D slices should be
19        resized to before being passed to the model.
20
21    Returns:
22        np.ndarray: A 3D numpy array of the same shape as the input volume, representing
23        the predicted mask. Note that the returned mask is not binary and contains
24        continuous values between 0 and 1.
25    """
26
27    mask = np.zeros_like(normalized_vol)
28    H, W, Z = normalized_vol.shape
29
30    for z in range(Z):
31        image = normalized_vol[:, :, z]
32        resized = resize(image, *model_shape) # Resize to fit the model input shape
33        predicted = get_2D_prediction(resized, model, verbose=verbose)
34        resized = resize(predicted, H, W) # Resize back to the original image shape
35        mask[:, :, z] = resized
36
37    return mask
```

Extrait de code 5 – predict_3D_mask

Dans le script à la fin du fichier, nous faisons appel à la fonction précédente :

```
1 # Chargement et prédiction
2 print(f'Chargement de {name}')
3 normalized_vol, vol_header = load_and_normalize_volume(path)
4 predicted_mask = predict_3D_mask(normalized_vol, model)
5
6 #...
7
8 for threshold in THRESHOLDS:
9     thresholded = apply_threshold(predicted_mask, threshold)
10
11 # Chemin de sauvegarde
12 folder = PREDICTIONS_FOLDER / name
13 os.makedirs(folder, exist_ok=True)
14 file = folder / f'{name}-prediction-threshold-{threshold:.4f}.nrrd'
15
16 # Ecriture
17 nrrd.write(str(file), thresholded, header=mask_header)
18 print(f" Enregistrement du masque de {name} avec seuil de {threshold}")
```

Extrait de code 6 – Script - Prédictions avec différents seuils

9.4.2 Reconstruction 3D - D15.nrrd

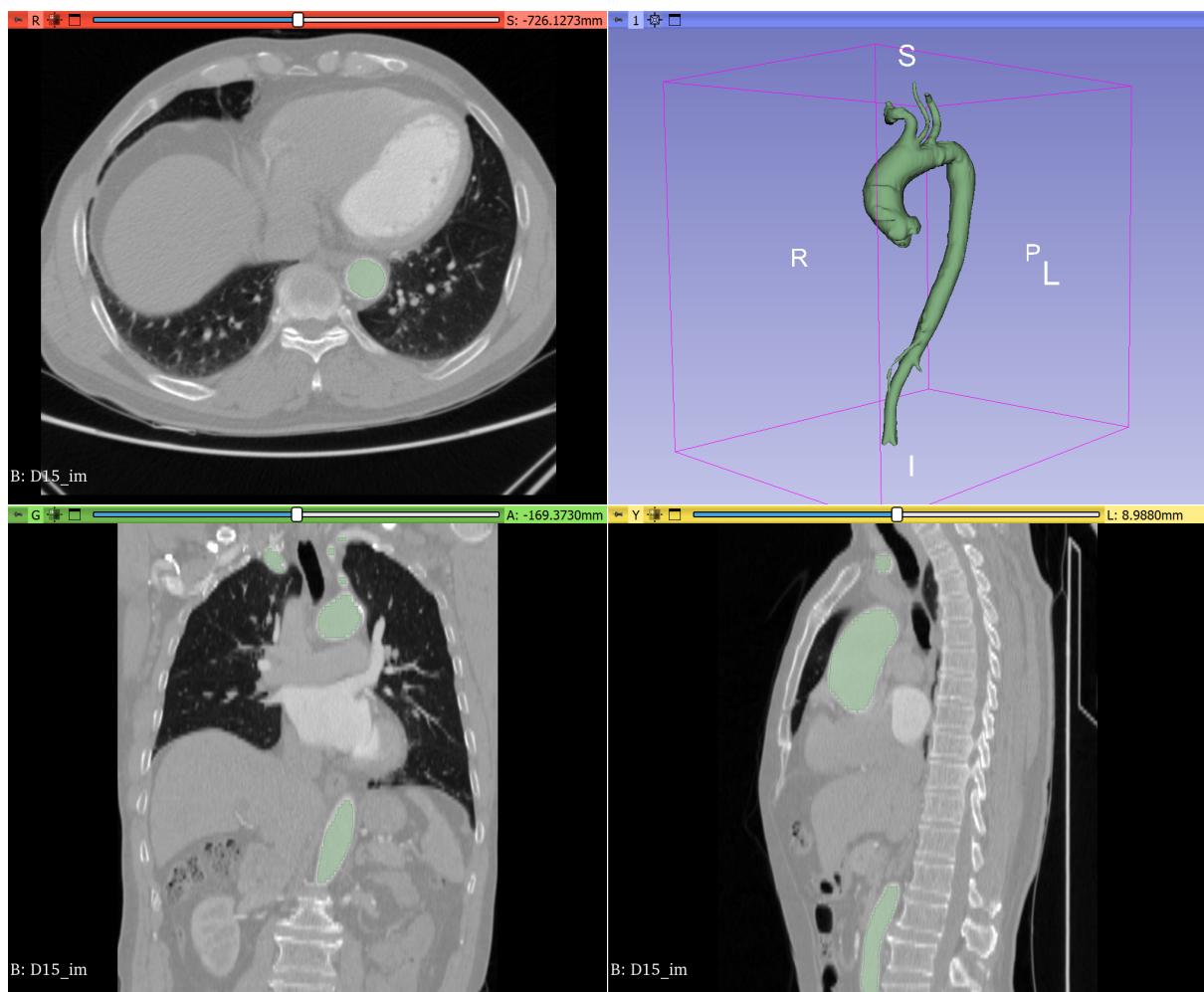


FIGURE 55 – Segmentation manuelle - D15.nrrd [4]

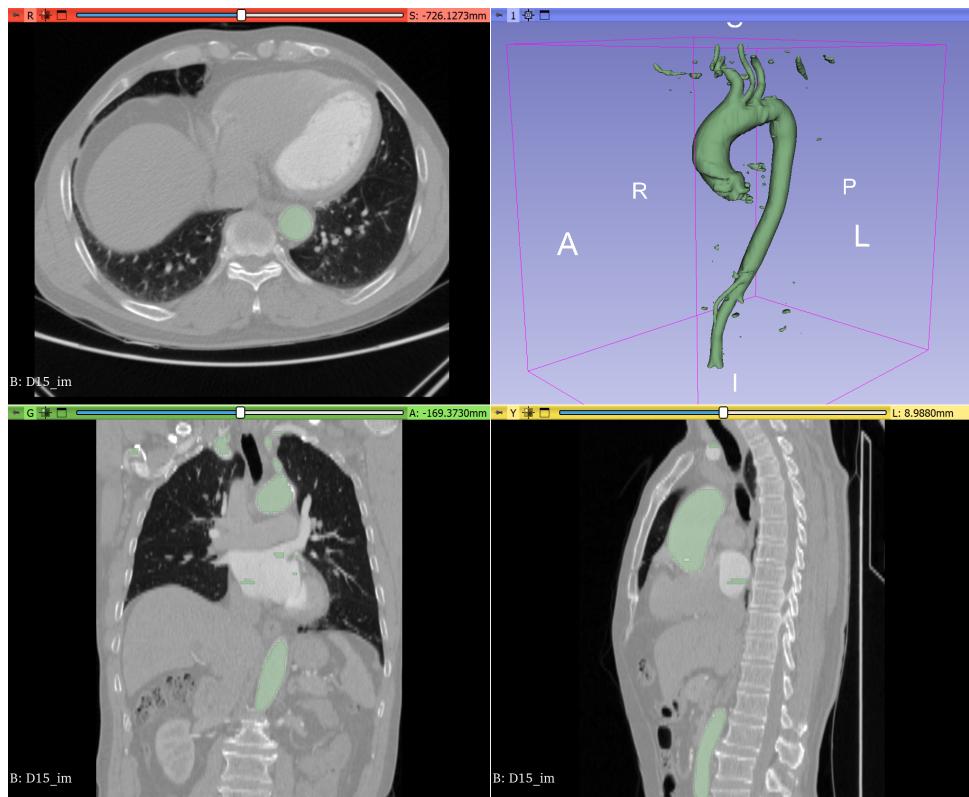


FIGURE 56 – Prédiction - Seuil 0.1 [4]

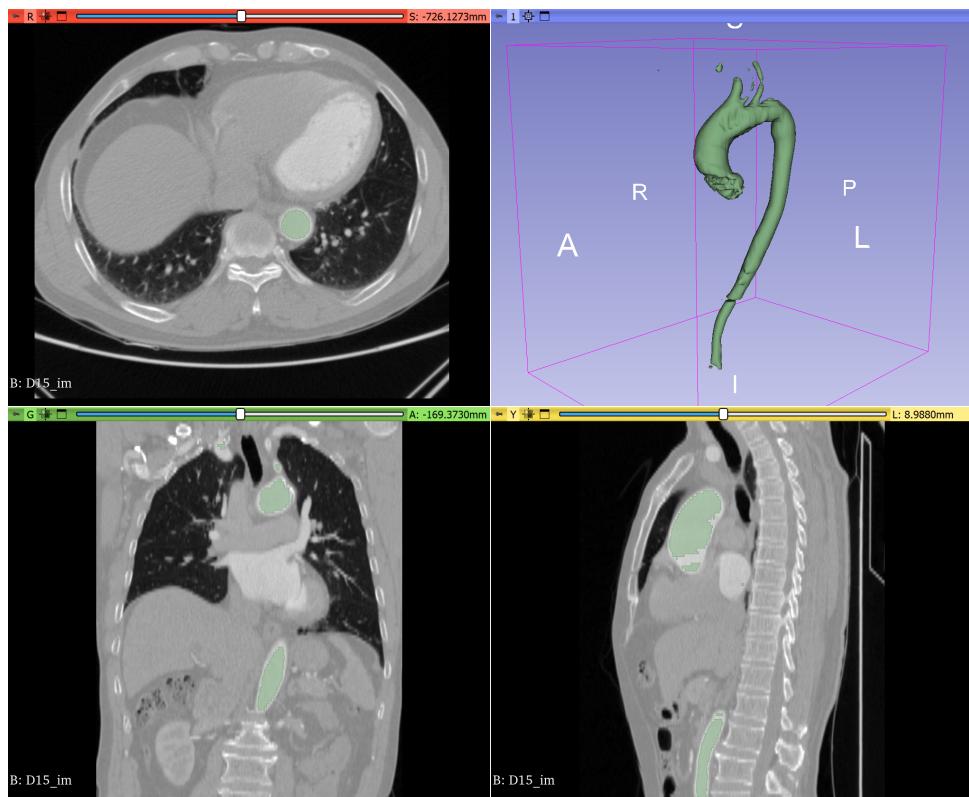


FIGURE 57 – Prédiction - Seuil 0.9 [4]

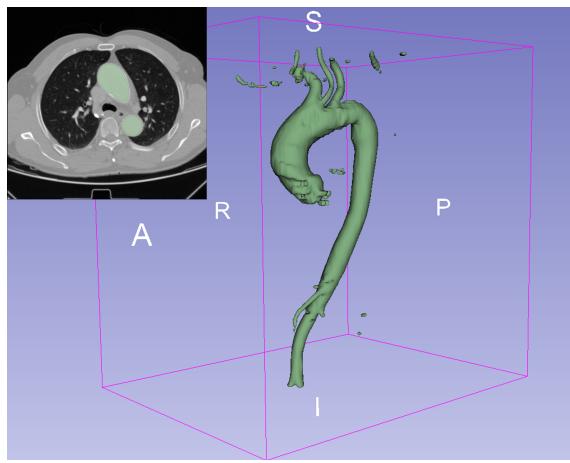


FIGURE 58 – Prédiction - Seuil 0.2

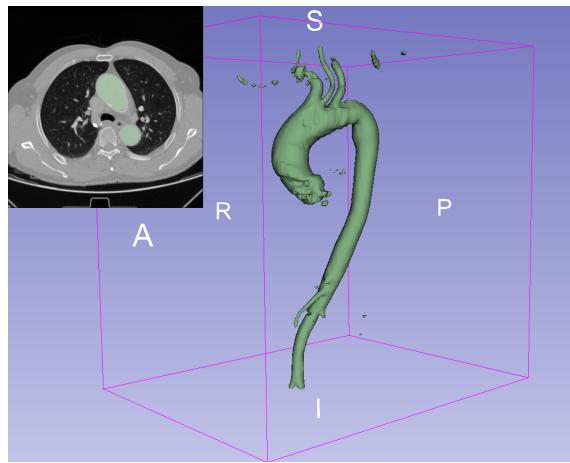


FIGURE 59 – Prédiction - Seuil 0.3

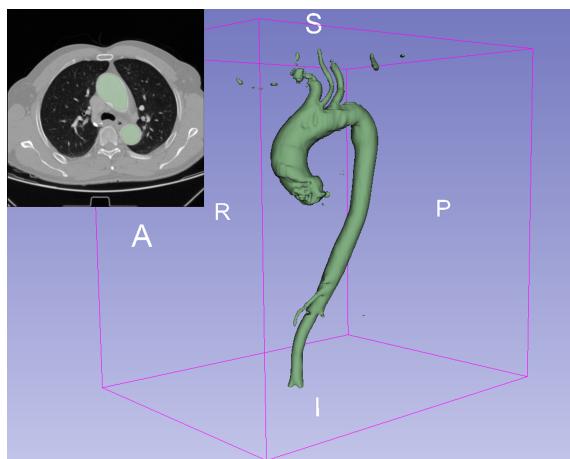


FIGURE 60 – Prédiction - Seuil 0.4

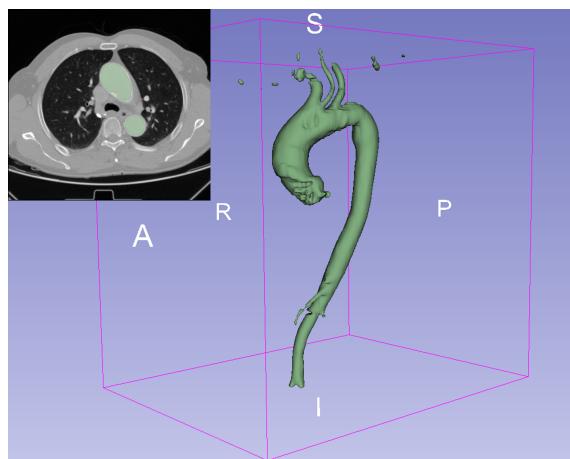


FIGURE 61 – Prédiction - Seuil 0.5

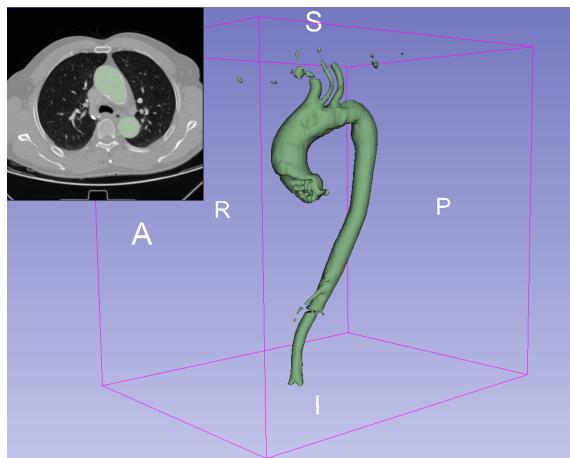


FIGURE 62 – Prédiction - Seuil 0.6

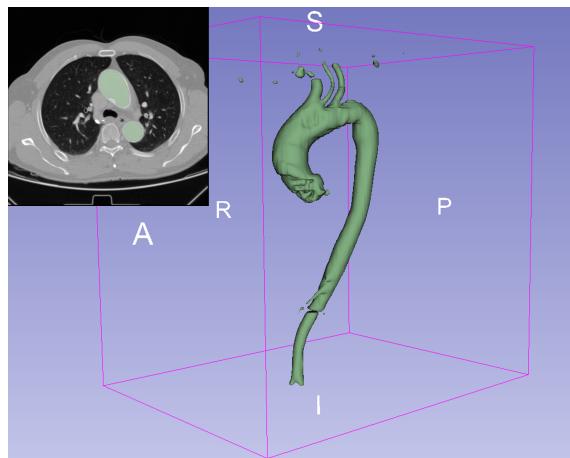


FIGURE 63 – Prédiction - Seuil 0.7

9.4.3 Reconstruction 3D - R17.nrrd

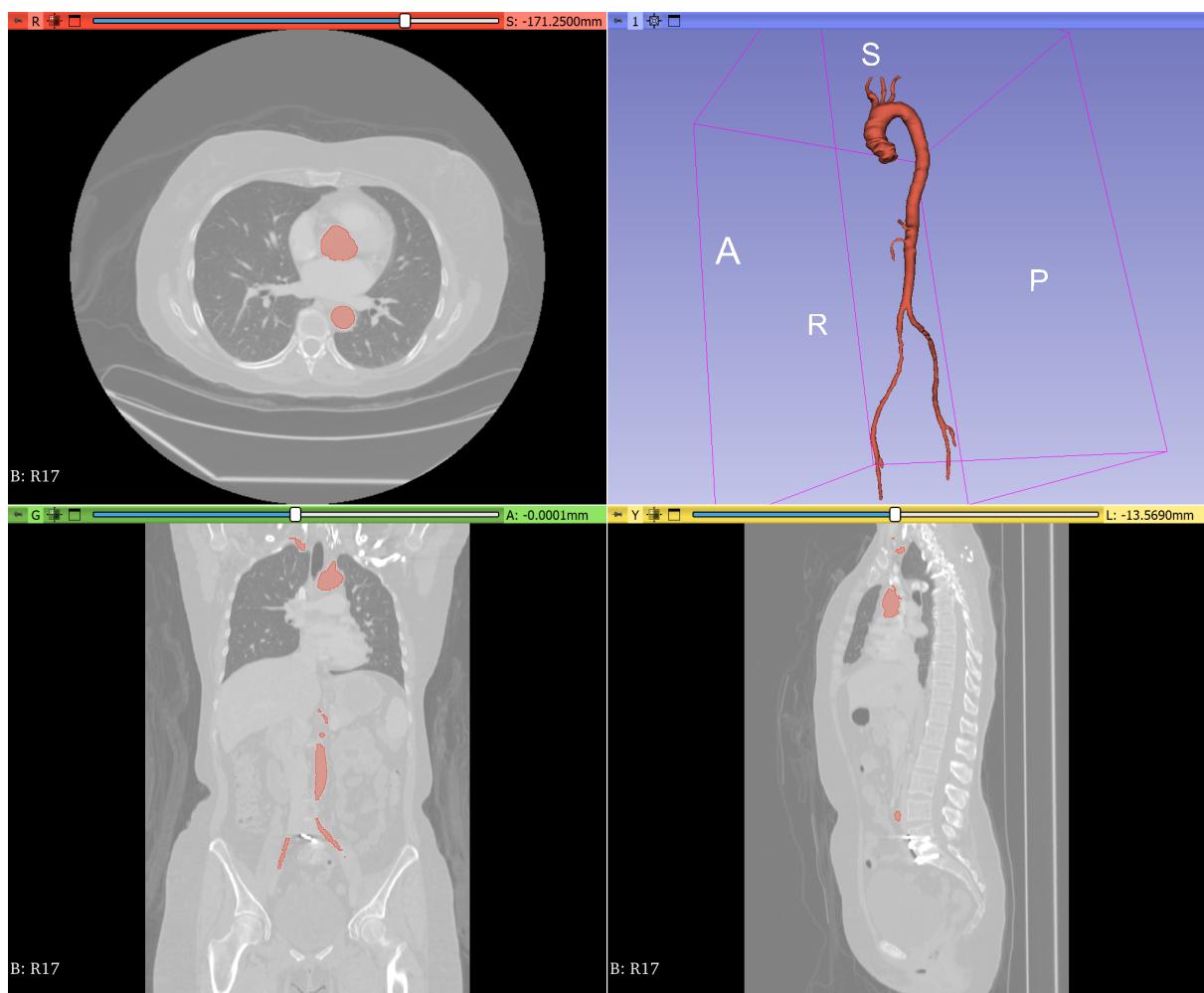


FIGURE 64 – Segmentation manuelle - R17.nrrd [4]

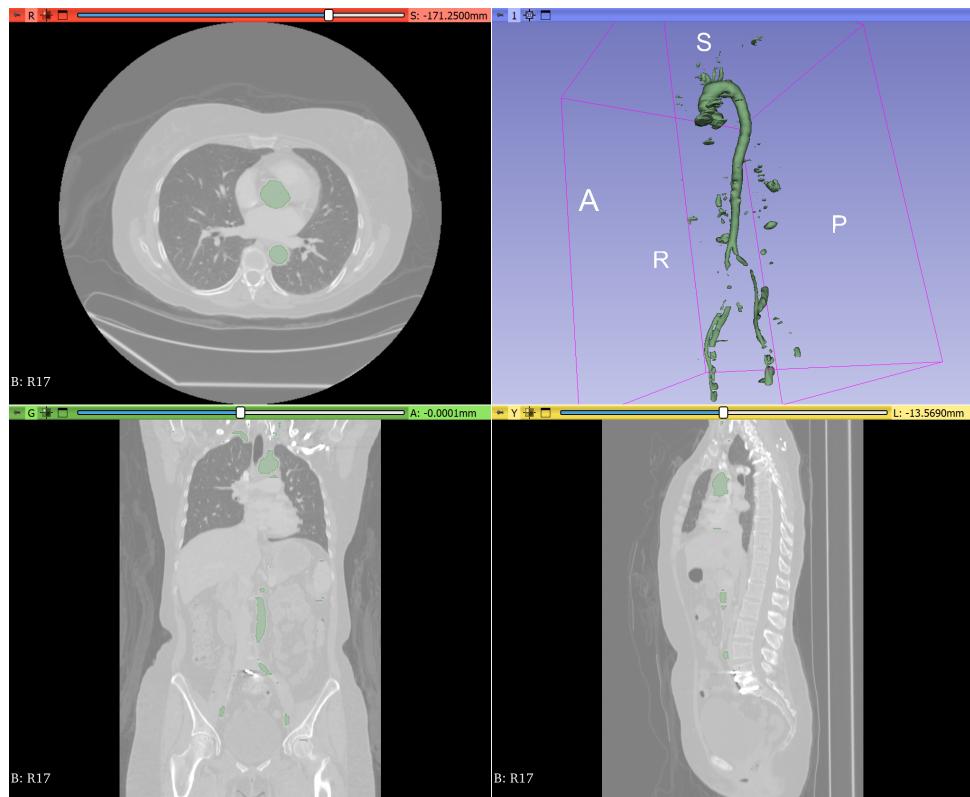


FIGURE 65 – Prédiction - Seuil 0.1 [4]

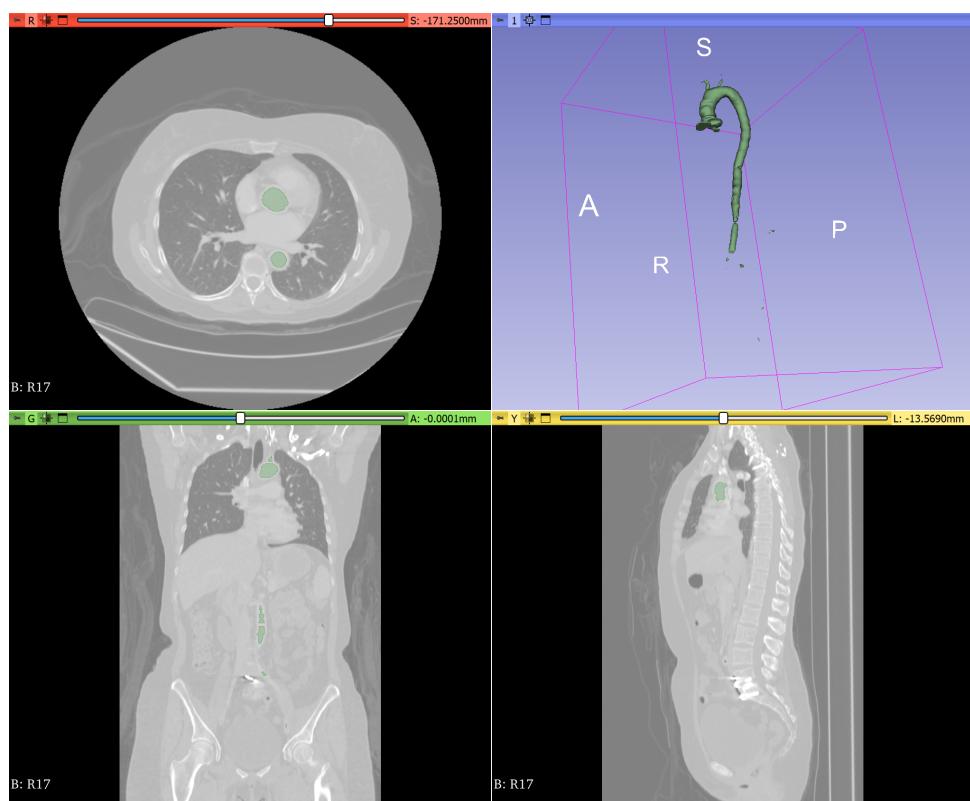


FIGURE 66 – Prédiction - Seuil 0.9 [4]

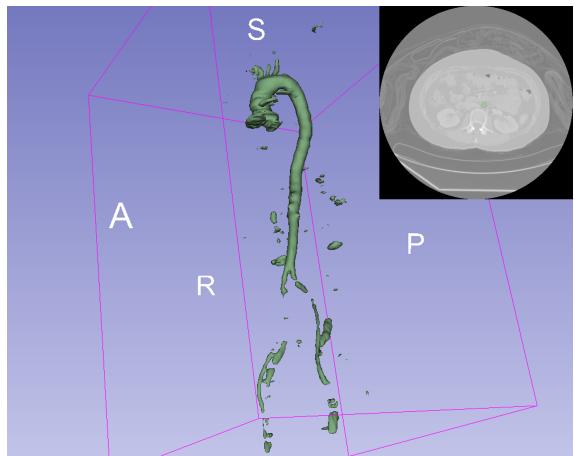


FIGURE 67 – Prédiction - Seuil 0.2

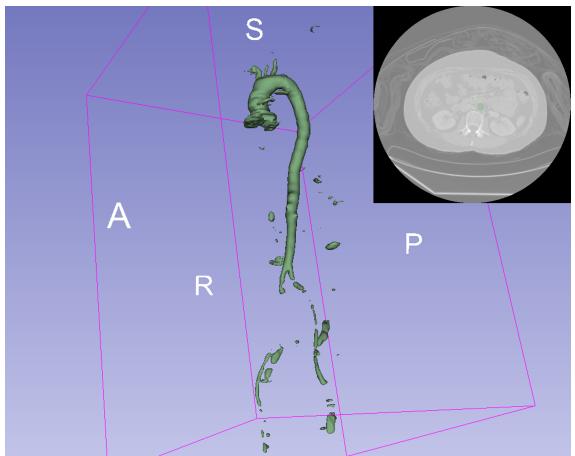


FIGURE 68 – Prédiction - Seuil 0.3

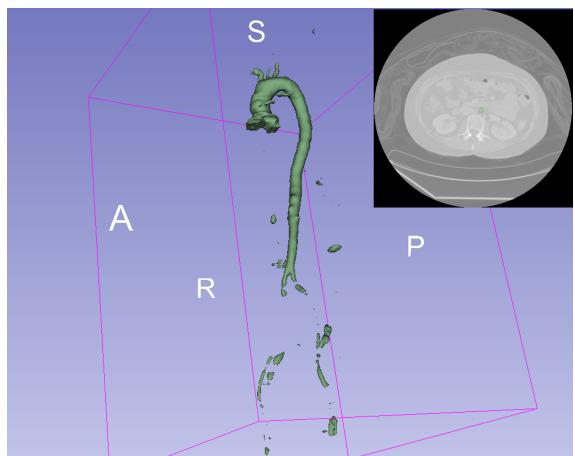


FIGURE 69 – Prédiction - Seuil 0.4

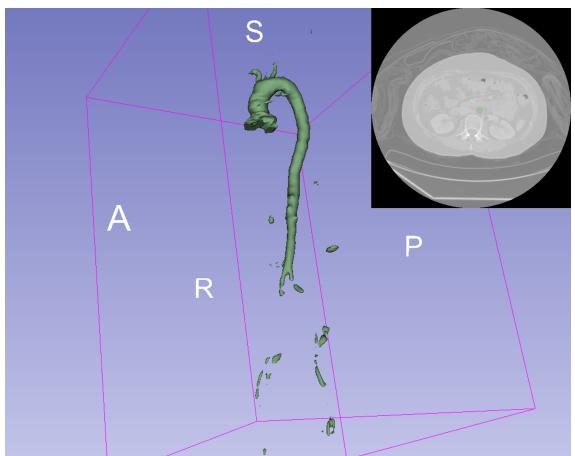


FIGURE 70 – Prédiction - Seuil 0.5

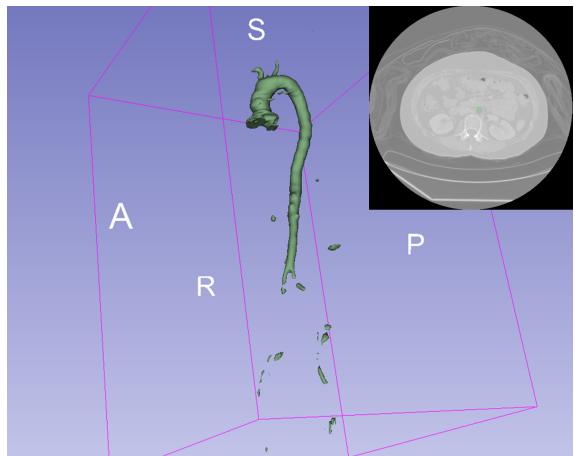


FIGURE 71 – Prédiction - Seuil 0.6

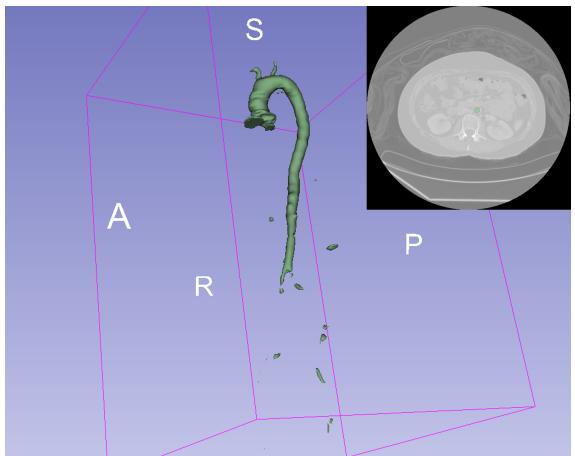


FIGURE 72 – Prédiction - Seuil 0.7

9.4.4 Reconstruction 3D - K17.nrrd



FIGURE 73 – Segmentation manuelle - K17.nrrd [4]

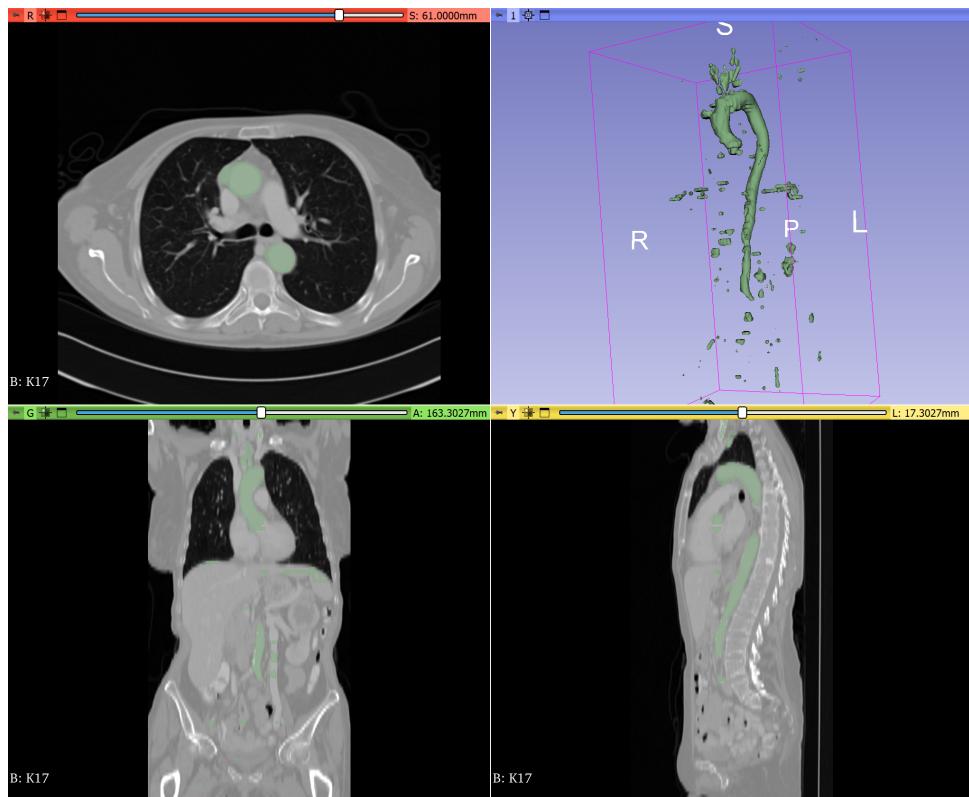


FIGURE 74 – Prédiction - Seuil 0.1 [4]

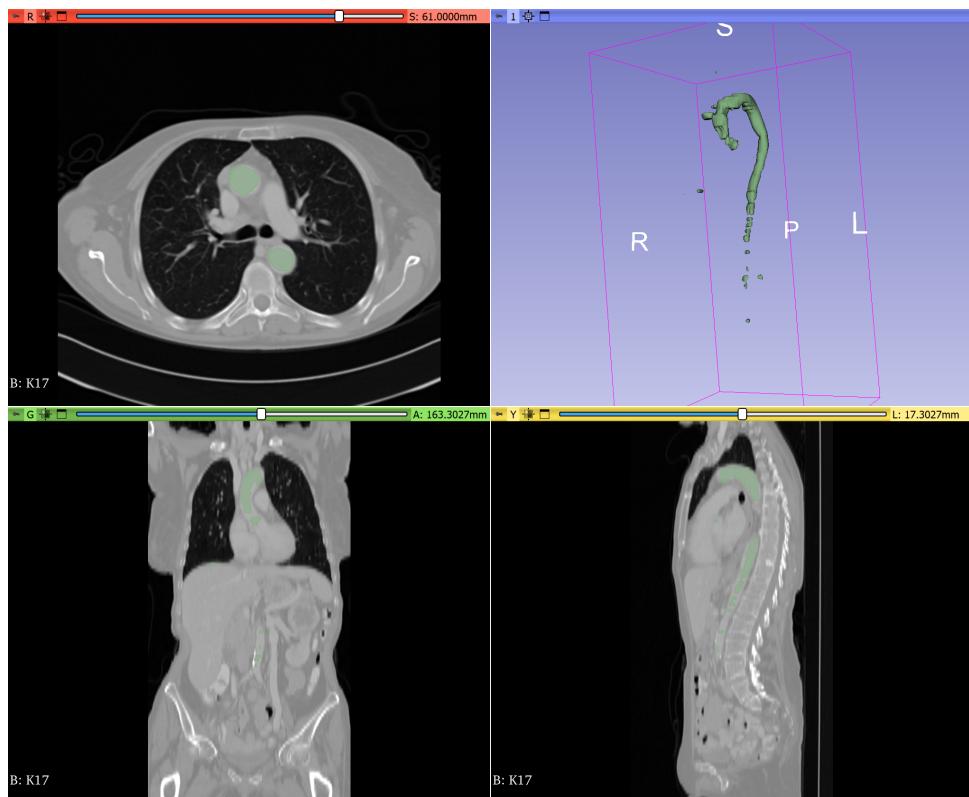


FIGURE 75 – Prédiction - Seuil 0.9 [4]

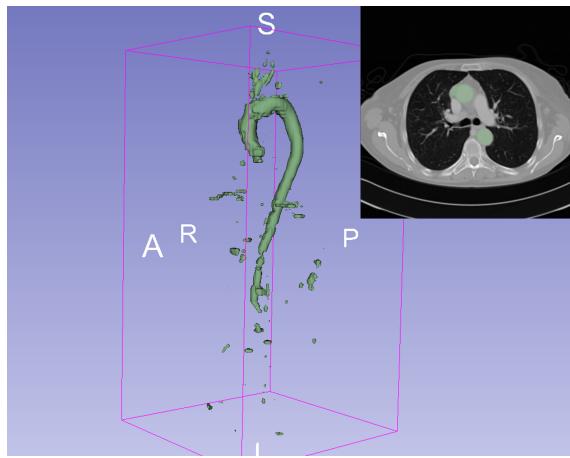


FIGURE 76 – Prédiction - Seuil 0.2

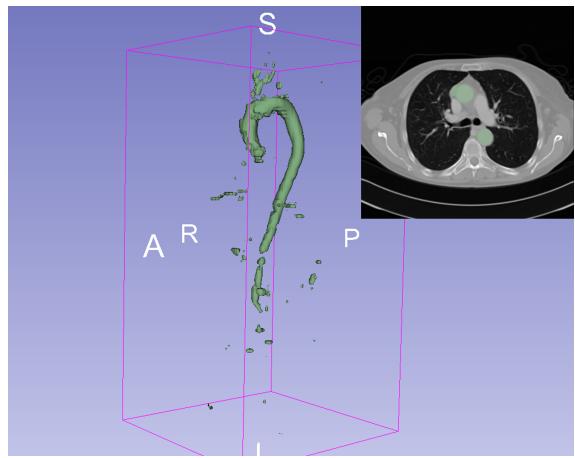


FIGURE 77 – Prédiction - Seuil 0.3

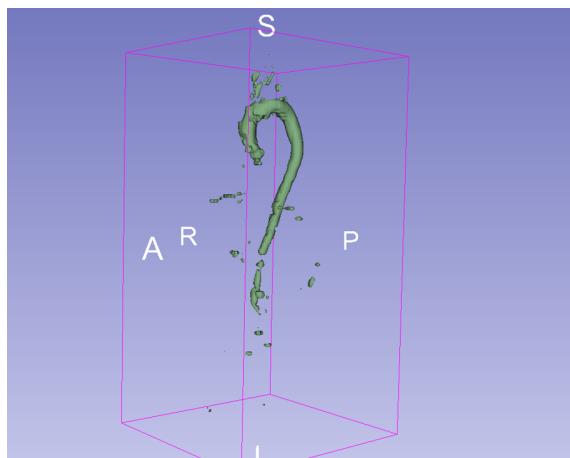


FIGURE 78 – Prédiction - Seuil 0.4

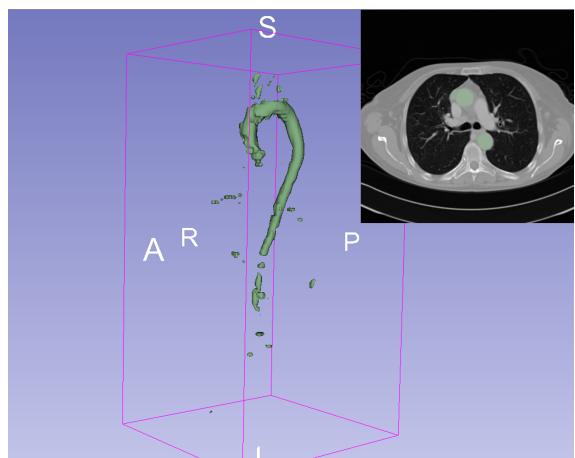


FIGURE 79 – Prédiction - Seuil 0.5

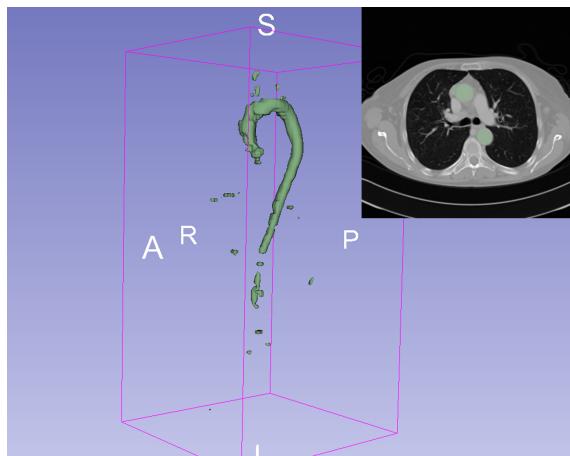


FIGURE 80 – Prédiction - Seuil 0.6

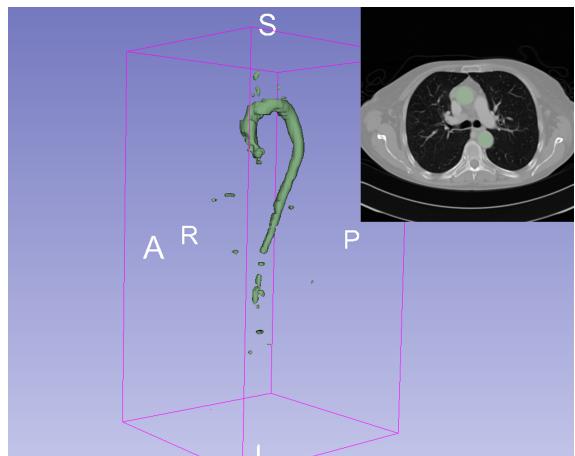


FIGURE 81 – Prédiction - Seuil 0.7

9.4.5 Reconstruction 3D - Exemple qui n'a pas fonctionné R16.nrrd

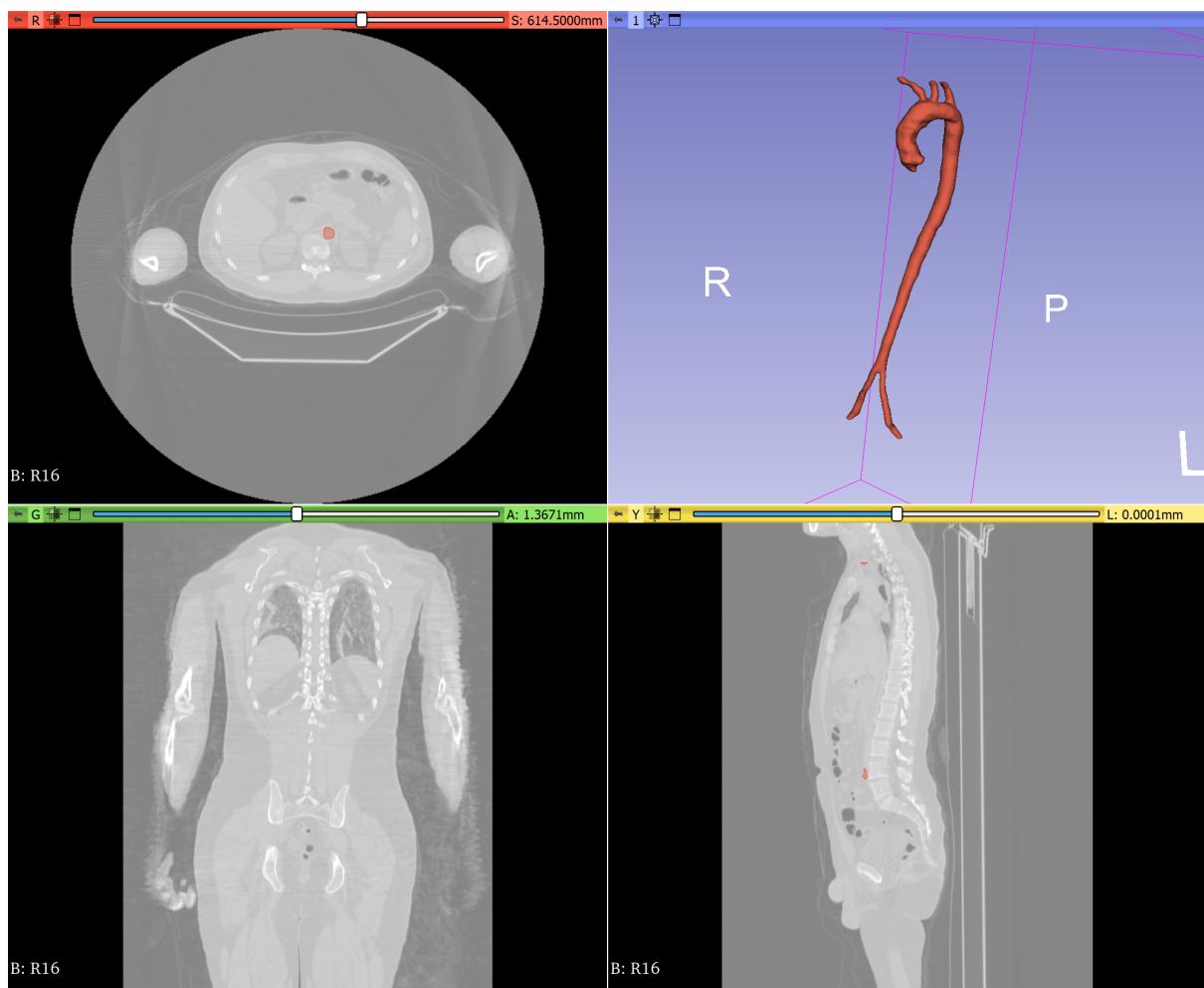


FIGURE 82 – Segmentation manuelle - R16.nrrd [4]

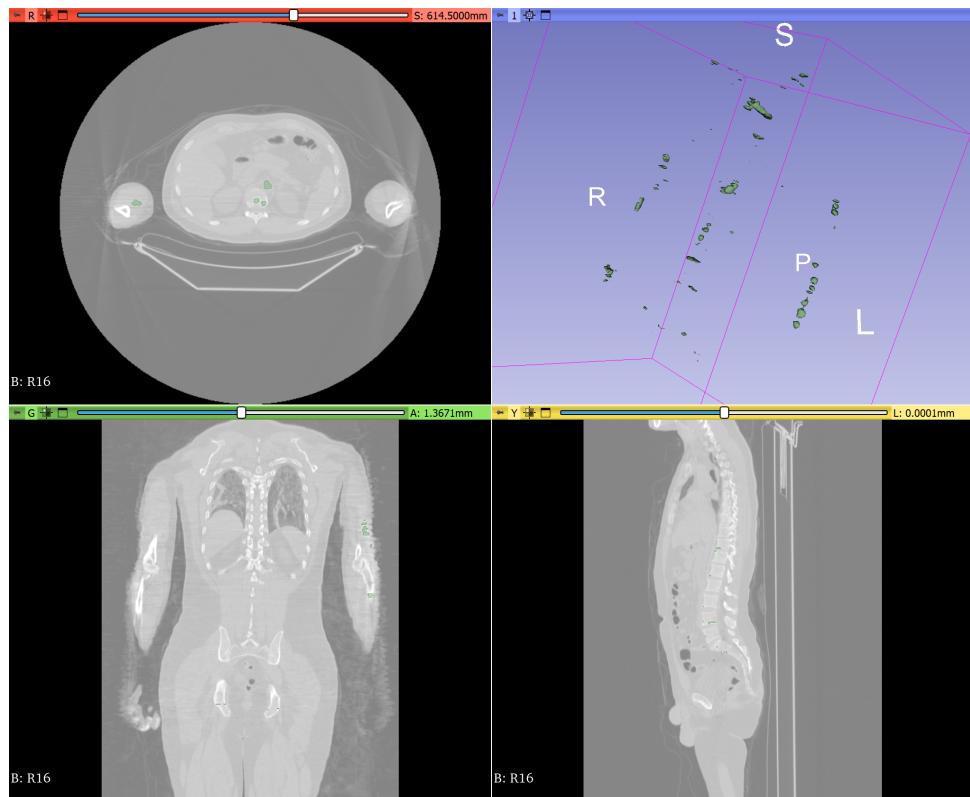


FIGURE 83 – Prédiction - Seuil 0.1 [4]

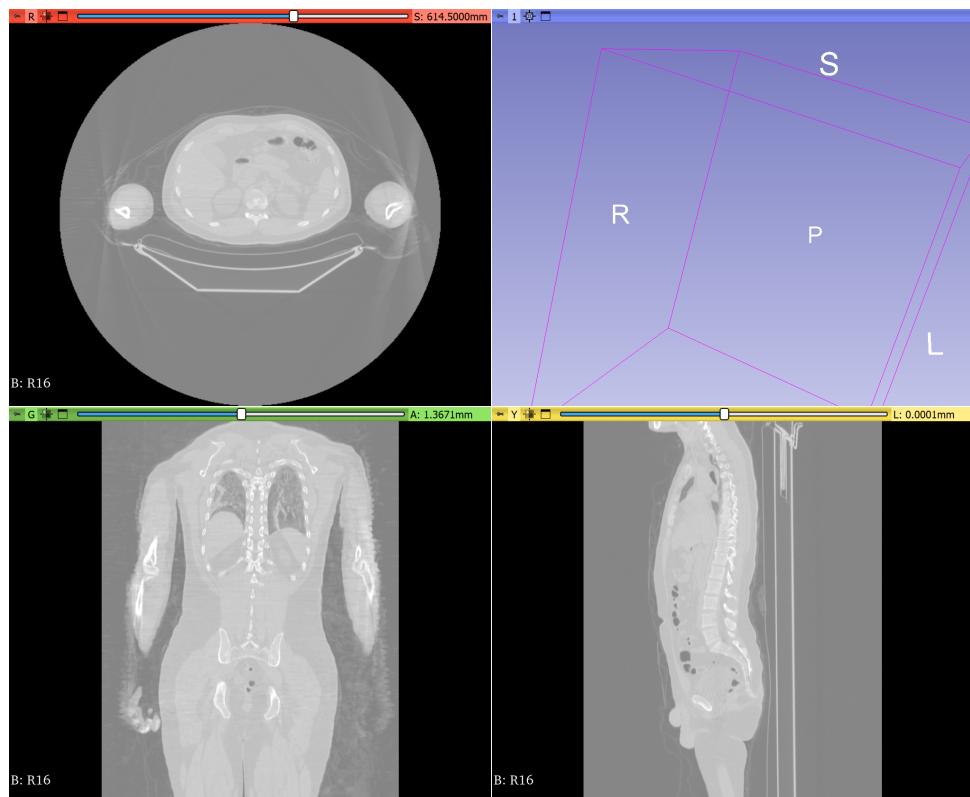


FIGURE 84 – Prédiction - Seuil 0.9 [4]

