

Université de Technologie de Compiègne

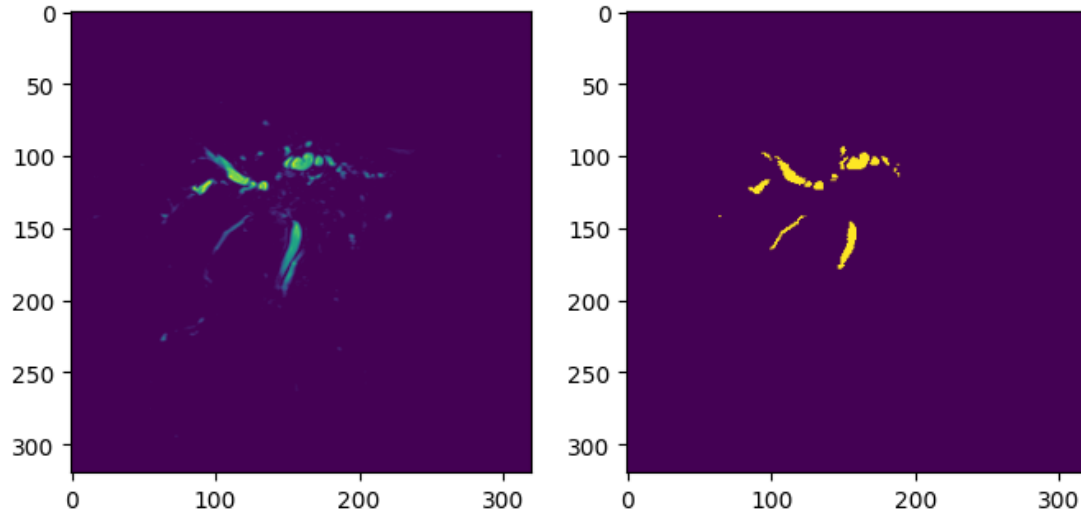
Rapport de stage ST01

Janvier 2025 – Février 2025

Segmentation des voies biliaires à partir d'images de tomodensitométrie dans le cadre de la CPRE.

Ningyuan Zhang

02 AOS



Sous la direction de :

Mme Isabelle Claude et M. Florian De Vuyst, laboratoire BMBI (Tuteurs)

M. Abdelhadi Essamlali, laboratoire BMBI (Doctorant)

Laboratoire de Biomécanique et Bioingénierie BMBI UMR CNRS 7338

Résumé

Le stage ST01 est une composante clé du master à l'Université de Technologie de Compiègne, permettant aux étudiants d'appliquer leurs connaissances théoriques dans un contexte pratique.

J'ai eu la chance de réaliser ce stage du 27 janvier au 21 février 2025 au sein de l'équipe IFSB du Laboratoire de Biomécanique et Bioingénierie (BMBI - UMR 7338). Cette équipe se concentre sur l'étude des écoulements physiologiques et les techniques thérapeutiques cardiovasculaires, ainsi que sur la modélisation des interactions fluides-structures biologiques.

Ce document résume les travaux que j'ai effectués et les résultats obtenus.

Table des matières

1.	Le laboratoire et l'équipe.....	4
1.1	Présentation du laboratoire BMBI	4
1.2	Présentation de l'équipe IFSB.....	5
2.	Contexte général du projet.....	5
2.1	Le projet MAAGIE	5
2.2	Reconstruction 3D des voies biliaires à l'aide de modèles U-net	6
3.	Mon sujet de recherche.....	7
3.1	Logiciels et données utilisés	7
3.2	Configuration des serveurs du laboratoire	8
3.3	Planification et état d'avancement des objectifs	9
3.3.1	Utilisation de vscode en SSH pour accéder aux serveurs et entraîner le code U-net.....	9
3.3.2	Accès aux supercalculateurs Grace et Ada	11
3.3.3	Rédaction du protocole d'accès aux nouveaux serveurs	12
3.4	Méthodes et mise en œuvre.....	12
3.4.1	Utilisation de TensorFlow pour entraîner un modèle sur le serveur Nodal	12
3.4.2	Utilisation de la fonctionnalité SSH de vscode pour accéder aux serveurs Grace et Ada	14
3.4.3	Accès au GPU sur le nouveau serveur.....	14
3.4.4	Conversion de l'architecture tensorflow de U-net en une nouvelle architecture pytorch	15
3.4.5	Ajout de fonctions d'augmentation des données au code sous la nouvelle architecture.....	15
4.	Tests d'efficacité	18
5.	Résultats	19
6.	Perspectives	20
6.1	Défis techniques non résolus.....	20
6.2	Points d'amélioration prévus	21
7.	Remerciements	21
8.	Références	22

1. Le laboratoire et l'équipe

1.1 Présentation du laboratoire BMBI

Le Laboratoire de Biomécanique et Bioingénierie (BMBI - UMR CNRS 7338) de l'Université de Technologie de Compiègne se distingue par son approche pluridisciplinaire, alliant mécanique, physique, biologie et chimie pour répondre à des enjeux de santé publique. Ses recherches couvrent l'étude des systèmes vivants aux échelles moléculaire, cellulaire et organique, avec des applications en diagnostic, traitement et prévention des pathologies musculo-squelettiques, cardio-vasculaires et métaboliques.

Organisé en trois équipes spécialisées (C2B, IFSB, C2MUST), le laboratoire explore des thématiques telles que les interactions fluides-structures biologiques et le développement d'outils biomimétiques pour la santé. En plus de ses avancées scientifiques, BMBI participe activement à la formation des étudiants et contribue au développement des technologies de santé, tout en travaillant en étroite collaboration avec des partenaires académiques, cliniques et industriels.

BMBI - Organigramme scientifique et fonctionnel



1.2 Présentation de l'équipe IFSB

L'équipe IFSB se concentre sur l'étude de la dynamique des fluides dans le système cardiovasculaire et le développement de nouvelles techniques thérapeutiques mini-invasives. En combinant des approches expérimentales, numériques et théoriques, l'équipe examine les variations hémodynamiques à différentes échelles – des molécules et cellules jusqu'aux organes – et leur impact sur le diagnostic et le traitement des maladies.

Les travaux incluent l'utilisation de la microfluidique et des simulations multiphysiques pour étudier le comportement des microcapsules et des cellules, avec des applications dans l'optimisation des stratégies thérapeutiques telles que l'embolisation, la réparation des valves cardiaques et le stenting.

Grâce à des collaborations interdisciplinaires et des partenariats cliniques solides, l'équipe IFSB s'engage à développer des thérapies cardiovasculaires innovantes et des solutions de traitement personnalisées pour les patients.

2. Contexte général du projet

2.1 Le projet MAAGIE

Le projet MAAGIE vise à améliorer la pratique de la CPRE (cholangiopancréatographie rétrograde par voie endoscopique), une procédure utilisée pour traiter des maladies des voies biliaires et pancréatiques. L'objectif est de rendre cette intervention plus précise et efficace en utilisant des technologies récentes comme la robotique, le traitement d'images et l'intelligence artificielle. Le projet cherche à surmonter deux grandes difficultés : la mise en place du cathéter dans les voies biliaires et l'interprétation complexe des images 2D lors de l'intervention. Pour cela, deux solutions principales sont envisagées : un suivi précis des instruments grâce à des capteurs et des algorithmes, et la création d'un guide souple et actif pour une navigation plus précise dans les voies biliaires.

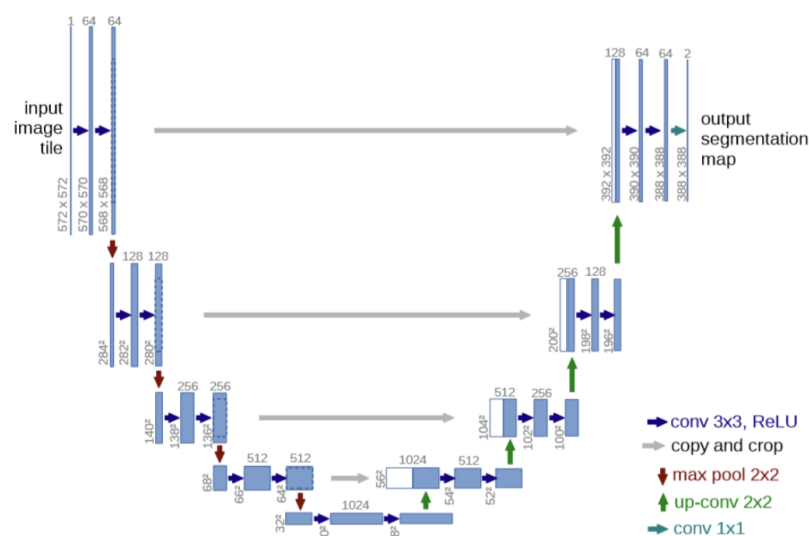
2.2 Reconstruction 3D des voies biliaires à l'aide de modèles U-net

La reconstruction 3D des voies biliaires repose sur une segmentation précise des images de cholangio-IRM, permettant de visualiser l'anatomie complexe de ces structures. L'algorithme U-Net, conçu spécifiquement pour la segmentation sémantique des images biomédicales, est largement utilisé pour cette tâche en raison de son architecture efficace.

L'architecture U-Net se compose de deux parties symétriques : un encodeur et un décodeur. L'encodeur extrait progressivement les caractéristiques essentielles de l'image grâce à des couches de convolution qui appliquent des filtres pour détecter des motifs tels que les contours et les textures. Ces couches sont accompagnées de couches de pooling, souvent par max-pooling, qui réduisent la résolution spatiale tout en conservant les informations clés, ce qui diminue la taille des données et rend le modèle plus efficace.

Le décodeur, quant à lui, reconstruit l'image segmentée en utilisant des couches de déconvolution pour augmenter la résolution. Des connexions skip relient directement les cartes de caractéristiques de l'encodeur à celles du décodeur, permettant au modèle de combiner les détails locaux avec le contexte global. Cette approche améliore la précision de la segmentation, en particulier pour les structures complexes comme les voies biliaires.

Enfin, la couche de sortie de U-Net applique une convolution finale avec une fonction d'activation appropriée, produisant une carte de probabilité pour chaque pixel, indiquant son appartenance ou non aux voies biliaires. En combinant les segmentations 2D des coupes successives, il devient possible de reconstruire un modèle 3D précis, facilitant l'analyse clinique et la planification des interventions médicales.



3. Mon sujet de recherche

Mon sujet de recherche porte sur la segmentation des voies biliaires à partir d'images de tomodensitométrie dans le cadre de la CPRE (Cholangiopancréatographie Rétrograde Endoscopique). Pendant mon stage d'un mois, je travaille sous la supervision de Mme Isabelle et de M. Abdelhadi pour comprendre en détail le code existant du modèle U-Net. Ce modèle, développé dans le cadre du projet ANR MAAGIE, vise à segmenter automatiquement les voies biliaires des patients à partir d'images spécifiques d'IRM (Imagerie par Résonance Magnétique).

Actuellement, les équipements informatiques disponibles à l'UTC nous permettent de mener cette recherche et d'obtenir des résultats satisfaisants. Cependant, ces ressources ne sont pas suffisantes pour exploiter pleinement les techniques d'augmentation des données nécessaires afin d'améliorer les performances du modèle. En effet, les algorithmes d'apprentissage profond exigent une puissance de calcul considérable, notamment durant la phase d'apprentissage.

Dans ce contexte, mon objectif est d'établir un protocole de formation pour utiliser les nouveaux supercalculateurs de l'UTC, Grace et Ada, afin d'entraîner le modèle. De plus, je dois implémenter des techniques d'augmentation des données sur ces nouvelles plateformes pour améliorer la précision, la capacité de généralisation et la robustesse du modèle, rendant ainsi son application clinique plus efficace.

3.1 Logiciels et données utilisés

Pour accéder au serveur du laboratoire, j'ai suivi la méthode recommandée par M. Abdelhadi en utilisant Visual Studio Code (VSCode) comme principale plateforme de connexion. Ces dernières années, VSCode est devenu l'un des environnements de développement les plus populaires pour les langages comme Python, grâce à ses nombreux avantages. Il offre une interface conviviale, une vaste bibliothèque d'extensions et une intégration fluide avec des systèmes de contrôle de version tels que Git. L'extension Remote - SSH permet de se connecter facilement à des serveurs distants, ce qui fait de VSCode un choix idéal pour le développement sur serveur.

Pour gérer les dépendances et l'environnement de travail, j'ai utilisé Conda, un gestionnaire d'environnements virtuels largement adopté dans les domaines de la science des données et de l'apprentissage automatique. Conda permet de créer des

environnements isolés, garantissant ainsi la compatibilité des bibliothèques et la reproductibilité des projets. J'ai choisi Python 3.12, la version stable la plus récente, afin d'assurer la prise en charge des bibliothèques modernes et d'optimiser les performances du modèle.

Le code original de M. Abdelhadi repose sur l'architecture U-Net, implémentée avec TensorFlow et Keras, fonctionnant parfaitement sous la version TensorFlow 2.18.0. Cependant, lors de la transition vers les nouveaux serveurs Grace et Ada, utilisant l'architecture ARM64, il a été nécessaire d'adapter l'environnement, car TensorFlow ne prend pas en charge cette architecture.

Pour surmonter cette limitation, j'ai migré le code vers PyTorch, un autre framework de deep learning reconnu pour sa flexibilité et son efficacité. J'ai choisi la version PyTorch 2.6.0+cu126, optimisée pour CUDA 12.6, ce qui garantit des performances optimales sur les nouveaux serveurs tout en assurant une exécution fluide de l'algorithme U-Net.

En outre, pour la gestion des versions et la collaboration, j'ai utilisé GitHub comme plateforme de dépôt de code. GitHub est l'une des plateformes les plus populaires pour le contrôle de version et la collaboration en ligne. Elle permet non seulement de stocker et de partager le code, mais aussi de suivre l'avancement des tâches via les Issues et d'effectuer des révisions de code grâce aux Pull Requests, garantissant ainsi un flux de travail structuré et efficace.

3.2 Configuration des serveurs du laboratoire

Dans le cadre de ce stage, les serveurs **Pilcam2** ont été utilisés pour les calculs intensifs. Ces serveurs sont accessibles via l'URL suivante : <https://pilcam2.utc.fr>. Les équipements et configurations des serveurs sont détaillés ci-dessous :

NODAL (x86_64, GPGPU)

Processeur : 2 AMD EPYC 7763, 64 cœurs, 128 fils chacun (256 cœurs logiques au total), fréquence de 2,45 GHz avec 256 Mo de cache.

Mémoire vive : 1,5 To RAM.

Carte graphique (GPU) : 1 NVIDIA Ampere A10, 24 Go de RAM GDDR6, 9216 cœurs CUDA, fréquence de 1,70 GHz.

Stockage : 4 SSD de 960 To configurés en RAID 10 pour assurer la sécurité des données et la performance.

GRACE/ADA (ARM64, GPGPU)

Processeur (CPU) : 1 NVIDIA Grace avec 72 cœurs Arm Neoverse V2, fréquence de 3,5 GHz, 1 Mo de cache L2 par cœur et 114 Mo de cache L3, accompagné de 480 Go de RAM LPDDR5X intégrée.

Carte graphique (GPU) : 1 NVIDIA Hopper H100, 96 Go de RAM HBM3, 16896 cœurs CUDA, fréquence de 1,98 GHz.

Stockage :

1 SSD NVMe M.2 22110 de 960 Go.

1 SSD NVMe E1.S de 15 mm avec 1,92 To.

3.3 Planification et état d'avancement des objectifs



L'organisation des tâches pendant le stage

3.3.1 Utilisation de vscode en SSH pour accéder aux serveurs et entraîner le code U-net

Au cours de la première semaine, ma tutrice Isabelle a réussi à me créer un compte de connexion sur le serveur pilcam. Chaque compte personnel dispose de 200 Go d'espace de stockage initial, cet espace étant commun à tous les serveurs de pilcam. J'ai ensuite obtenu le code U-net auprès de M. Abdelhadi, et avec son aide ainsi que celle de ses collègues, j'ai pu utiliser la fonctionnalité SSH de vscode pour accéder au serveur nodal sur pilcam, qui est basé sur une architecture x86. En exécutant la commande `nvidia-smi` dans le terminal, j'ai pu vérifier les configurations de la carte graphique sur ce serveur

(voir figure ci-dessous).

```
(base) bash-5.1$ nvidia-smi
Thu Feb 27 16:05:08 2025
```

NVIDIA-SMI 545.23.08			Driver Version: 545.23.08			CUDA Version: 12.3		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC	
Fan	Temp	Perf	Mem:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.	
0	NVIDIA A10	Off	00000000:21:00.0	Off	0%	Default	0	
0%	50C	P0	61W / 150W	20989MiB / 23028MiB			N/A	

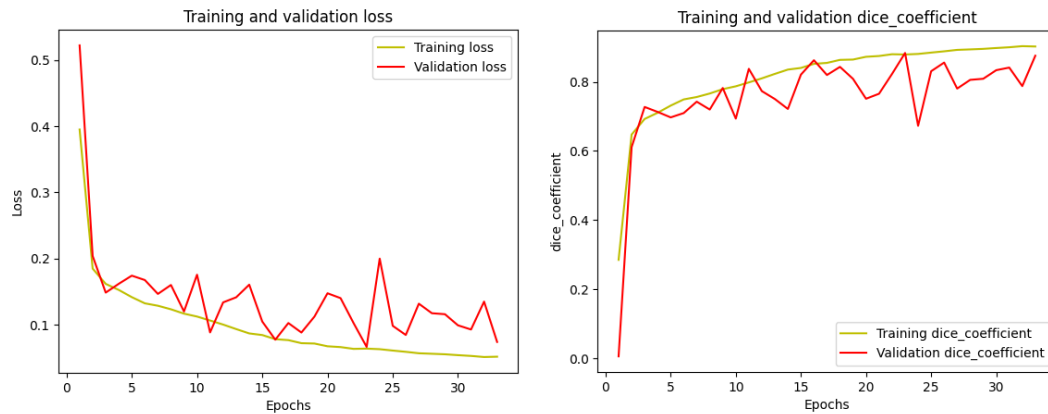
Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	Usage
ID	ID	ID					
0	N/A	N/A	32776	G	...qus/2023/linux_a64/code/bin/ABQcaeK	43MiB	
0	N/A	N/A	2963402	C	/bin/python	20928MiB	

Ensuite, j'ai utilisé la fonctionnalité conda intégrée du serveur nodal. J'ai exécuté la commande `conda create --name my_env python=3.12` dans le terminal pour créer un environnement virtuel spécifique, que j'ai ensuite activé à l'aide de la commande `conda activate my_env`. Une fois l'environnement activé, j'ai installé les différentes bibliothèques requises par le code U-net à l'aide de `pip`.

Enfin, j'ai modifié les chemins du code ainsi que certaines incompatibilités dues aux différences de version des bibliothèques. Après ces ajustements, le code a pu s'exécuter correctement, et les résultats obtenus sont illustrés dans la figure suivante. Pour cette session de tests, les paramètres importants étaient les suivants :

1. **batch_size=16** : taille de lot de 16 ;
2. **patience=10** : déclenchement de l'early stopping après 10 cycles sans amélioration ;
3. **input_shape=(320,320,1)** : dimensions d'entrée de 320×320 avec un seul canal (image en niveaux de gris) ;
4. **n_classes=1** : une seule classe de sortie, correspondant à une tâche de segmentation binaire ;
5. **dice_bce_loss(alpha=0.5)** : une pondération équilibrée entre Dice Loss et Binary Cross Entropy (BCE).

La durée moyenne d'entraînement par cycle était de 79 secondes. L'early stopping s'est déclenché à l'époque 33, ce qui a permis d'obtenir le meilleur score de Dice.



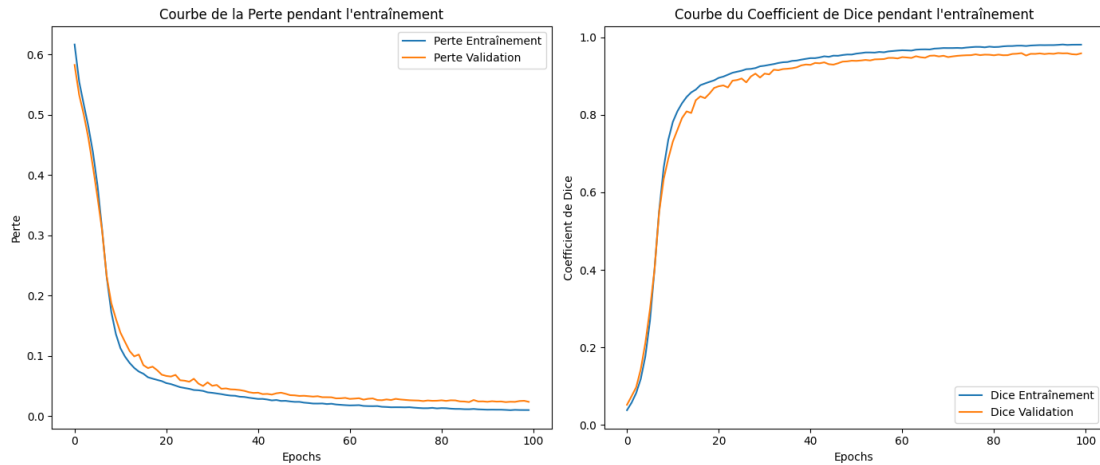
Le graphique de gauche montre la perte (Loss) d'entraînement et de validation du U-Net : la courbe jaune représente la perte d'entraînement et la rouge la perte de validation. La perte diminue globalement, mais celle de validation fluctue, indiquant un possible surapprentissage. Le graphique de droite montre le coefficient de Dice, qui mesure la performance de segmentation (plus il est élevé, mieux c'est). La courbe d'entraînement monte régulièrement, tandis que celle de validation fluctue, suggérant une bonne performance sur l'entraînement mais une généralisation limitée.

3.3.2 Accès aux supercalculateurs Grace et Ada

Puisque Grace et Ada partagent la même carte GPU, il n'y a aucune différence dans la façon d'appeler et d'accéder à ces deux serveurs. En réalité, il est même possible de passer d'un supercalculateur à l'autre sans devoir réinitialiser l'environnement distant de vscode. Par conséquent, dans la suite du texte, je ne parlerai que des travaux réalisés sur Grace, mais soyez assuré que j'ai également effectué les mêmes tests sur Ada.

Grace et Ada sont tous deux basés sur une architecture arm64, ce qui les distingue du serveur nodal qui repose sur une architecture x86. N'ayant pas trouvé de version spécifique de TensorFlow compatible avec cette architecture, j'ai migré le code U-net du framework TensorFlow vers PyTorch. J'ai également intégré un cadre de test simplifié pour l'augmentation des données dans la version PyTorch, et j'ai mené plusieurs séries de tests avec différentes méthodes d'augmentation des données sous ce cadre.

Les résultats montrent que les données originales combinées à des images ayant subi une rotation de 15 degrés offrent les meilleures performances en termes d'augmentation. Après cinq séries de tests, j'ai généré cinq fichiers modèles au format .py, avec une précision sur le jeu de validation comprise entre 87 % et 98 %. Ces résultats démontrent une robustesse et une capacité de généralisation remarquables, comme illustré dans la figure suivante.



3.3.3 Rédaction du protocole d'accès aux nouveaux serveurs

Dans le cadre de mon stage, et afin de faciliter la continuité des travaux par mes futurs collègues, j'ai rédigé deux protocoles pratiques et concis. Ces documents expliquent comment accéder aux serveurs Nodal, Grace et Ada via vscode, ainsi que les étapes nécessaires pour exécuter correctement le code sur ces machines. Grâce à ces protocoles, les utilisateurs peuvent directement entraîner un modèle U-net sans devoir parcourir l'ensemble du rapport.

Étant donné que ce stage n'a duré qu'un mois, il est possible que certaines parties des protocoles soient encore incomplètes ou perfectibles. C'est pourquoi j'ai également conservé ces deux documents en version Word, permettant aux collègues qui leur succéderont de corriger ou d'améliorer plus facilement leur contenu.

3.4 Méthodes et mise en œuvre

3.4.1 Utilisation de TensorFlow pour entraîner un modèle sur le serveur Nodal

Le serveur Nodal utilise une architecture X86 et est équipé d'une carte graphique NVIDIA Ampere A10 offrant jusqu'à 24 Go de mémoire vidéo, ce qui permet des calculs GPU efficaces et stables. Dans le fichier `U-net_Cross_validation_bce-dice.ipynb`, le code utilise l'architecture TensorFlow pour maximiser la capacité de calcul du GPU. Le fragment de code suivant nous permet de vérifier si l'accès au GPU a été réussi :

```
import tensorflow as tf

print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))
```

Cette sortie indique le nombre de GPU détectés ; si elle est de 0, cela signifie que TensorFlow n'a détecté aucun GPU disponible. Dans ce cas, il convient de vérifier les points suivants :

1. Vérifier si le serveur contient un GPU, en entrant nvidia-smi dans le terminal du serveur.
2. Confirmer si l'architecture du serveur est compatible avec TensorFlow. Par exemple, si l'architecture est arm64, elle pourrait ne pas être supportée, nécessitant un ajustement du code ou un changement d'architecture.
3. Assurer que la version GPU de TensorFlow est installée, sinon, il faut l'installer.

La structure du code dans U-net_Cross_validation_bce-dice.ipynb est claire et facile à comprendre. Une fois l'accès au GPU établi, le module d'importation de données peut afficher chaque groupe d'images via matplotlib. Ensuite, le module de calcul de Dice et le module de modèle U-net suivent. Le modèle est construit et son architecture est affichée comme suit :

```
model = build_unet(input_shape=(320, 320, 1), n_classes=1)

model.summary()
```

Ce code affiche l'architecture du modèle et d'autres détails clés après l'entrée des données d'exemple, tels que les dimensions de sortie et le nombre de paramètres de chaque couche.

La section suivante est le module K-fold, qui divise les données d'entrée en 5 groupes, avec 4/5 des données utilisées pour l'ensemble d'entraînement et le 1/5 restant pour l'ensemble de test. La fonction .shape affiche la forme de chaque groupe de données, avec un exemple de 40 ensembles d'images IRM :

train.shape est de 32, ce qui représente 4/5 des données réparties dans l'ensemble d'entraînement pour les 40 groupes.

test.shape est de 8, représentant le 1/5 restant des données réparti dans l'ensemble de test.

La partie suivante est la conversion 2D des données, les aplatissant en un format 2D. Comme les données originales sont sous forme de tableau tridimensionnel (chaque groupe contenant plusieurs coupes d'IRM), nous les aplatissons en tableau bidimensionnel et ajoutons une dimension. Dans la partie formation du modèle, il est nécessaire d'ajuster des paramètres tels que `fold_no` et `batch_size`. Après une période de formation, un fichier modèle .h5 est généré. Après la fin de la formation de chaque groupe, il est nécessaire de modifier manuellement le nombre de cycles de formation, jusqu'à ce que la formation sur les 5 groupes de données soit complète.

3.4.2 Utilisation de la fonctionnalité SSH de vscode pour accéder aux serveurs Grace et Ada

Ensuite, j'ai essayé de me connecter au serveur Grace en utilisant la même méthode que pour le serveur Nodal, mais j'ai rencontré des problèmes. Lorsque j'ai tenté de me connecter via la fonctionnalité SSH de vscode, un message d'erreur s'est affiché. Cependant, en utilisant l'interface graphique de TurboVNC ou la commande SSH dans le terminal Windows, la connexion a été réussie.

Le problème provenait d'une anomalie dans l'environnement distant de vscode (conflits de plugins, versions incompatibles, etc.). À chaque fois que je changeais de serveur, vscode accédait au dossier `.vscode-server/` dans mon répertoire utilisateur. Or, étant donné que cet espace est partagé entre les différents serveurs, le passage d'un serveur à un autre entraînait l'utilisation des mêmes fichiers, provoquant des erreurs de connexion en raison des différences entre serveurs.

Pour résoudre ce problème, j'ai opté pour une méthode de réinitialisation de l'environnement distant de vscode. Avant de changer de serveur, je me connectais d'abord au serveur via SSH dans le terminal et exécutais la commande `rm -rf ~/.vscode-server/` pour supprimer ce dossier. Ainsi, lors de la prochaine connexion SSH via vscode, les fichiers nécessaires étaient réinstallés, garantissant une connexion fonctionnelle.

3.4.3 Accès au GPU sur le nouveau serveur

En raison de l'adoption de l'architecture arm64 sur les nouveaux serveurs "Grace" et "Ada", différente de l'architecture X86 traditionnelle, nous n'avons pas pu trouver une version de TensorFlow pour GPU compatible avec cette architecture et CUDA 12.4 ou supérieur. Nous n'avons également pas réussi à accéder au GPU des serveurs avec la version actuelle 2.18.0 de TensorFlow. Par conséquent, nous avons abandonné les architectures TensorFlow et Keras au profit de l'architecture PyTorch. Ce changement

nous a permis de continuer à exploiter les ressources GPU de ces serveurs, tout en apportant plus de flexibilité et d'efficacité à l'entraînement de nos modèles.

3.4.4 Conversion de l'architecture tensorflow de U-net en une nouvelle architecture pytorch

Nous avons utilisé un autre fichier de code, `U-net_Cross_validation_bce-dice_pytorch.ipynb`, pour réaliser cette conversion, afin de le différencier du fichier de code U-net utilisant l'architecture TensorFlow. Dans cette partie, nous avons tenté de maintenir une cohérence logique avec le code original pour faciliter la compréhension par les futurs utilisateurs. Cependant, en raison des limitations de connaissances de l'auteur et des différences inévitables dans l'utilisation des fonctions et des paramètres lors de la migration de code, ce nouveau fichier semble plus complexe que le précédent. Ainsi, il est conseillé de lire d'abord le fichier `U-net_Cross_validation_bce-dice.ipynb` avant de consulter ce nouveau code. Le code PyTorch diffère de l'original à plusieurs égards :

Premièrement, dans la partie de conversion 2D, la structure des données requise pour l'entraînement des modèles PyTorch est différente, nécessitant une réorganisation des dimensions $(N, H, W, 1)$ en $(N, 1, H, W)$. Deuxièmement, dans les fonctions de calcul de Dice et de construction de la structure U-net, des fonctions natives de PyTorch remplacent celles de l'architecture Keras. De plus, l'utilisation de `optimizer = optim.Adam(model.parameters(), lr=1e-4)` introduit un paramétrage de taux d'apprentissage, ici fixé à 0.0001. Les autres logiques de code restent globalement similaires à celles de la version TensorFlow.

3.4.5 Ajout de fonctions d'augmentation des données au code sous la nouvelle architecture

Dans cette section, nous avons étendu les capacités de traitement des données du modèle U-net en introduisant une série de méthodes d'augmentation de données pour améliorer la capacité de généralisation du modèle sur de nouvelles données et augmenter la diversité des données d'entraînement. Voici les techniques d'augmentation de données mises en œuvre et leurs définitions :

Identité (Identity) : Garde l'image et son étiquette de segmentation inchangées. C'est l'opération de base pour l'augmentation des données, utilisée pour assurer la présence d'images originales non modifiées dans le jeu de données.

Rotation (Rotate) : Tourne l'image et son étiquette de segmentation pour augmenter la robustesse du modèle aux variations d'orientation des images. L'angle de rotation par défaut est de 15 degrés.

Ajout de bruit (Noise) : Introduit un bruit gaussien dans l'image, aidant le modèle à gérer les perturbations aléatoires des données d'entrée dans les applications réelles.

Retournement (Flip) : Effectue un flip horizontal ou vertical de l'image et de son étiquette de segmentation, augmentant la diversité des échantillons.

Ajustement du contraste (Contrast) : Modifie le contraste de l'image pour changer la clarté et l'intensité des détails.

Correction gamma (Gamma) : Ajuste la luminosité de l'image, permettant au modèle de fonctionner efficacement sous différentes conditions d'éclairage.

Transformation affine (Affine) : Applique une transformation affine à l'image et à l'étiquette de segmentation, simulant le zoom, la rotation et la translation de l'image, ce qui aide le modèle à traiter des images provenant de différents angles et distances.

Ces méthodes sont gérées de manière unifiée dans un dictionnaire augmentations, permettant une application facile et systématique de ces transformations pendant l'entraînement. Chaque méthode d'augmentation est conçue pour manipuler simultanément l'image et son étiquette de segmentation, assurant la cohérence des données.

Exemple d'application :

Dans l'implémentation PyTorch, la méthode d'augmentation est sélectionnée comme suit :

```
selected_aug = 'rotate'
```

En appelant les fonctions définies dans le dictionnaire augmentations, nous pouvons facilement traiter le jeu de données d'entraînement. Cette flexibilité permet de tester différentes stratégies de traitement des données au cours du développement du modèle.

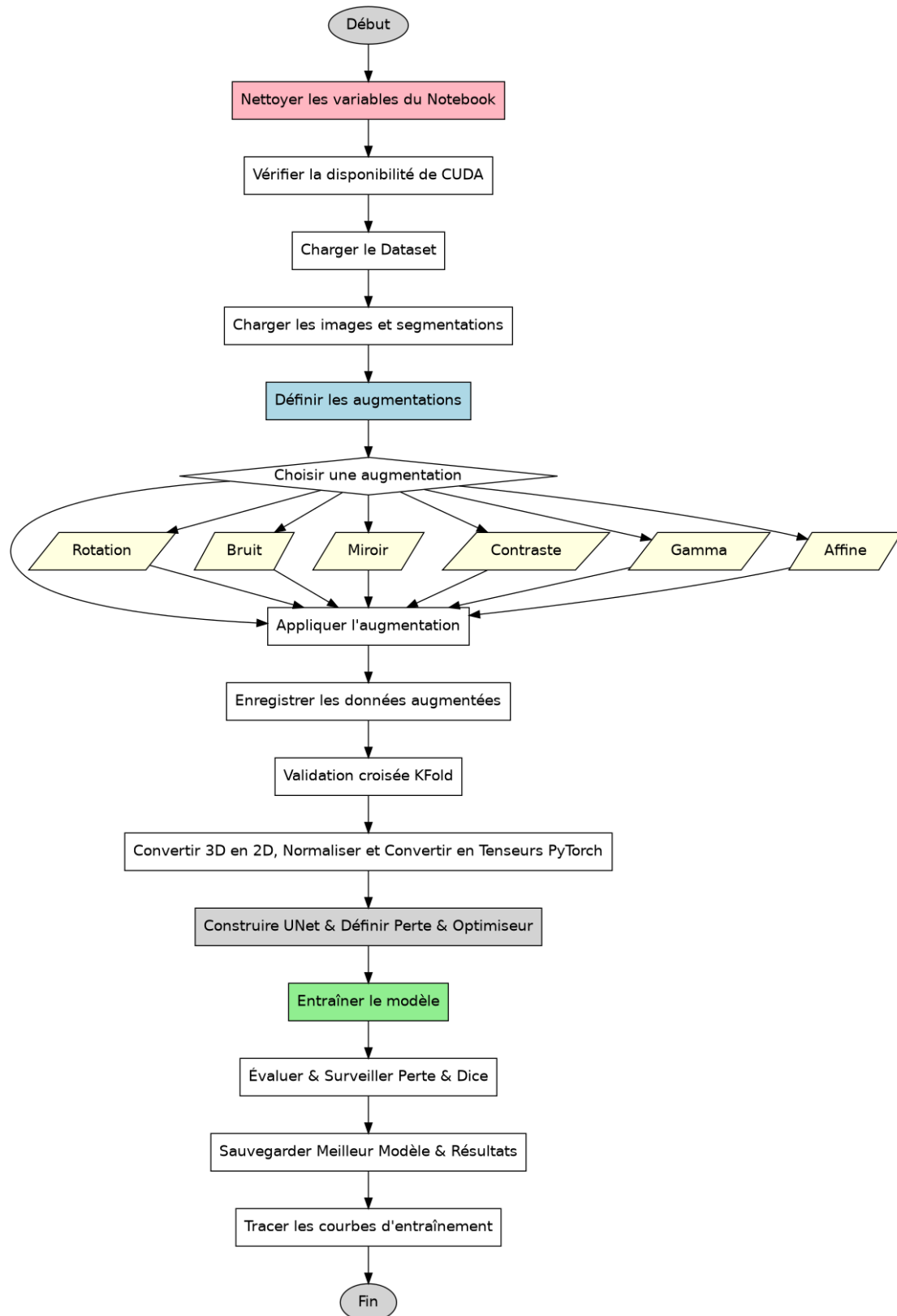


Diagramme de flux du code version Grace

4. Tests d'efficacité

Pour évaluer l'efficacité du modèle, j'ai utilisé le module KFold pour diviser les données en cinq groupes. Chaque groupe a été séparé en un ensemble d'entraînement et un ensemble de test, garantissant ainsi l'indépendance des données de test et évitant toute contamination des données d'entraînement. Les performances du modèle ont été évaluées à l'aide des métriques suivantes : Loss (fonction de perte), Dice coefficient (coefficient de Dice), Val_loss (perte sur l'ensemble de validation) et Val_dice_coefficient (coefficient de Dice sur l'ensemble de validation).

Dans un premier temps, j'ai entraîné le modèle avec TensorFlow en utilisant les données sans augmentation. Les résultats obtenus montrent qu'une surdapprentissage (overfitting) est apparue, ce qui signifie que la capacité de généralisation du modèle n'est pas optimale. De plus, la mémoire GPU de 24 Go du serveur Nodal a limité le batch_size à 16, avec une occupation de mémoire avoisinant 20 Go. Chaque époque d'entraînement prenait en moyenne 79 secondes.

Afin d'optimiser les performances, j'ai effectué plusieurs ajustements des hyperparamètres, notamment la taille du lot (batch_size) et le taux d'apprentissage (learning rate). J'ai constaté que sur le serveur Nodal, la valeur maximale de batch_size était 16, avec un temps moyen d'entraînement de 79 secondes par époque. En revanche, sur le supercalculateur Grace, le batch_size pouvait être augmenté jusqu'à 128, mais le temps moyen d'entraînement par époque était identique à celui obtenu avec batch_size = 32, soit 27 secondes. Cela suggère que l'augmentation du batch size au-delà de 32 n'apporte pas de gain significatif en termes de vitesse d'entraînement.

En parallèle, dans la version PyTorch du code, il est possible d'ajuster le taux d'apprentissage (learning rate), ce qui constitue un avantage par rapport à la version TensorFlow. Après plusieurs essais, j'ai fixé le learning rate à $1e-4$, ce qui a permis d'obtenir les meilleurs résultats en termes de convergence et de stabilité du modèle.

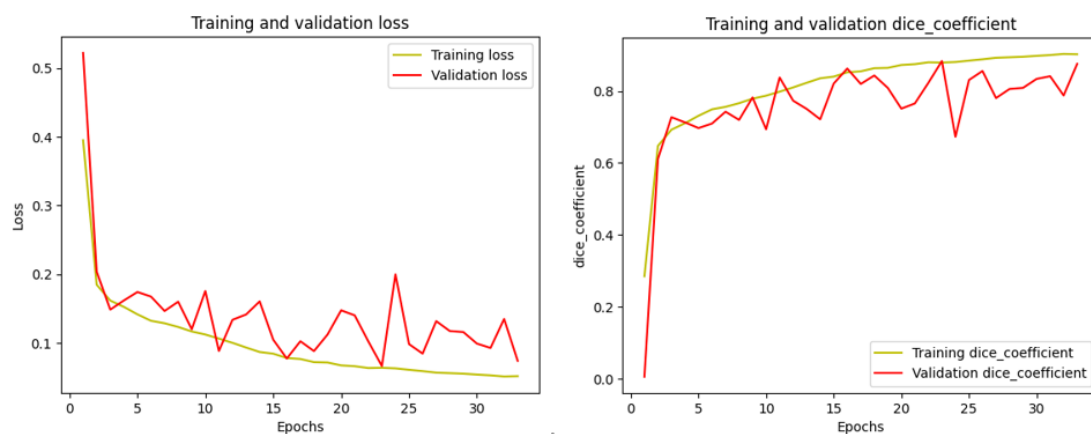
Ensuite, j'ai utilisé la version PyTorch du code et testé différentes stratégies d'augmentation des données afin d'améliorer la robustesse du modèle. Parmi les différentes méthodes d'augmentation testées, la rotation de 15 degrés a montré la plus forte amélioration de l'exactitude et de la capacité de généralisation du modèle. Grâce à l'utilisation du supercalculateur Grace, j'ai pu augmenter la taille du batch à 32, ce qui a permis de réduire le temps d'entraînement à 27 secondes par époque tout en garantissant la stabilité du modèle.

Ces résultats démontrent que le passage à PyTorch couplé à l'utilisation des supercalculateurs Grace/Ada et des techniques d'augmentation des données a permis d'optimiser le temps d'entraînement et d'améliorer les performances du modèle.

5. Résultats

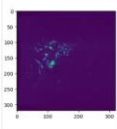
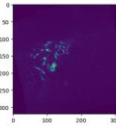
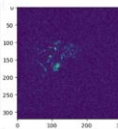
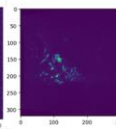
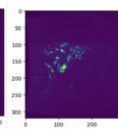
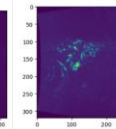
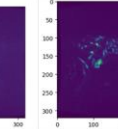
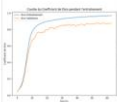
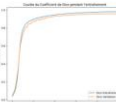
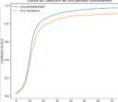
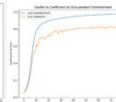
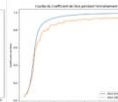
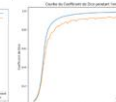
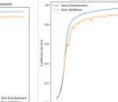
Dans cette section, nous présentons les résultats obtenus après l'entraînement du modèle en appliquant différentes méthodes d'augmentation de données. Nous analysons l'impact de ces transformations sur les performances du modèle en évaluant sa convergence à l'aide des courbes de perte d'entraînement et des courbes du coefficient de Dice. Enfin, nous comparons les coefficients de Dice finaux obtenus avec chaque méthode d'augmentation dans un tableau récapitulatif.

Tout d'abord, la figure suivante illustre l'évolution du coefficient de Dice et courbes de perte d'entraînement lors de l'entraînement du modèle sous TensorFlow.



Nous pouvons observer que la courbe présente d'importantes oscillations, ce qui indique une capacité de généralisation limitée du modèle.

Les images suivantes montrent les résultats obtenus avec différentes techniques d'augmentation des données sous PyTorch, ainsi que les courbes du coefficient de Dice correspondantes. Ces représentations permettent d'analyser l'évolution des performances du modèle au cours de l'entraînement.

							
Mode de l'augmentation Paramètres	Original	Rotate angle = 15	Noise Var = 0.02	Flip axis = 0	Contrast alpha=1.5	Gamma gamma=1.5	Affine
Coefficient de Dice final	87.59%	95.89%	92.85%	82.75%	93.84%	94.23%	92.08%
							

Nous constatons que l'augmentation des données a permis d'améliorer la capacité de généralisation du modèle, aboutissant à une meilleure précision. Parmi toutes les méthodes testées, la rotation de 15° s'est avérée la plus efficace, avec un coefficient de Dice final avoisinant 95%. Ce résultat met en évidence l'importance de l'augmentation des données dans l'amélioration des performances des modèles de segmentation d'images médicales. En particulier, la rotation semble aider le modèle à mieux s'adapter aux variations directionnelles des structures anatomiques, renforçant ainsi sa précision de prédiction.

En conclusion, l'augmentation des données joue un rôle essentiel dans l'optimisation des performances du modèle. Nos expériences montrent que certaines méthodes, comme la rotation de 15°, améliorent significativement les capacités de segmentation du modèle, tandis que d'autres techniques, comme l'ajout de bruit, ont un impact plus limité. Le choix stratégique des méthodes d'augmentation peut donc favoriser une meilleure stabilité et capacité de généralisation du modèle.

6. Perspectives

6.1 Défis techniques non résolus

En raison de la durée limitée du stage, certains problèmes restent à résoudre.

Le principal défi concerne la compatibilité du modèle entraîné avec 3D Slicer. Actuellement, 3D Slicer ne prend en charge que les modèles au format .h5 générés par TensorFlow, tandis que notre modèle PyTorch est sauvegardé sous .pt ou .pth. À ce jour, aucune méthode fiable n'a été trouvée pour convertir ces fichiers sans perte de précision. Une alternative serait d'ajouter une prise en charge native des modèles PyTorch dans un plugin de 3D Slicer, évitant ainsi toute conversion.

Un autre problème concerne l'optimisation du temps d'entraînement en fonction du batch size. Normalement, une augmentation du batch size permet de réduire le temps par époque, mais dans nos expériences, passer de `batch_size = 32` à 128 sur GPU n'a pas réduit le temps d'entraînement. Cela limite l'exploitation optimale des ressources GPU.

Les causes possibles incluent :

Un goulot d'étranglement au niveau du DataLoader, ralentissant le chargement des données vers le GPU.

Un manque d'optimisation de certains calculs, empêchant l'amélioration des performances avec un batch size plus grand.

Une latence accrue dans Batch Normalization (BatchNorm) avec des grands batches.

Une analyse approfondie est nécessaire pour identifier et résoudre ces limitations.

6.2 Points d'amélioration prévus

L'amélioration la plus prioritaire est de permettre à 3D Slicer d'accepter directement les modèles PyTorch, supprimant ainsi la nécessité de conversion.

De plus, le module d'augmentation des données pourrait être optimisé. Actuellement, une seule transformation est appliquée par entraînement, sans combinaison de plusieurs méthodes. Tester des stratégies mixtes pourrait améliorer la robustesse du modèle.

Enfin, nous avons utilisé principalement des méthodes classiques d'augmentation, telles que la rotation et le cutout, sans explorer des techniques plus avancées comme :

Fourier Domain Augmentation,

Self-Supervised Learning-Based Augmentation.

Ces approches ont montré des performances prometteuses en segmentation d'images médicales. Leur implémentation pourrait offrir des améliorations significatives du modèle.

7. Remerciements

Je tiens à exprimer ma sincère gratitude à Mme Isabelle Claude et M. Florian de Vyust. M. de Vyust, en tant que responsable du cours ISC3, m'a offert cette opportunité de stage, et

je lui en suis très reconnaissante. Mme Claude, ma principale encadrante, m'a accompagnée avec patience et bienveillance, m'aidant à comprendre la structure du laboratoire et du projet. Face aux difficultés techniques, elle a mobilisé M. Marc, expert en réseaux neuronaux, pour m'aider à résoudre les problèmes d'accès aux GPU. Son soutien et ses encouragements ont été une grande source de motivation.

Je remercie également M. Abdelhadi et Marine pour leur aide précieuse dans la compréhension du code U-Net et l'utilisation de 3D Slicer. Un grand merci aussi à mes collègues stagiaires, Lyam, Diane et Biyao, avec qui j'ai partagé un mois enrichissant, ainsi qu'à Isabella, doctorante au laboratoire, qui m'a fait découvrir son projet et avec qui j'ai vécu des moments mémorables, comme une séance d'escalade improvisée !

Ce stage a été une expérience très enrichissante. J'ai approfondi mes connaissances en réseaux neuronaux, U-Net et augmentation des données, tout en renforçant ma maîtrise de TensorFlow et PyTorch. J'ai également découvert les aspects matériels des serveurs. En parallèle, mon français s'est amélioré, bien que des progrès restent à faire.

Enfin, cette expérience m'a amenée à réfléchir à mon avenir : envisager un doctorat en imagerie médicale après mon master me semble désormais une possibilité concrète. Je suis reconnaissante pour tout ce que ce stage m'a apporté. Merci à tous !

8. Références

[1] N. Chen, *Intégration logicielle pour la reconstruction 3D des voies biliaires*, Rapport de stage TN09, Université de Technologie de Compiègne, février - juillet 2024.

[2] A. Essamlali, *La cholangiopancréatographie rétrograde par voie endoscopique*, Université de Technologie de Compiègne, 8 décembre 2021.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," *arXiv preprint arXiv:1505.04597*, 2015. [Online]. Available: <https://arxiv.org/abs/1505.04597>.

[4] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J Big Data* **6**, 60 (2019). <https://doi.org/10.1186/s40537-019-0197-0>

[5] CNRS - Formation FIDLE, FIDLE / Principes et concepts des réseaux de neurones convolutifs (CNN), 2024. [Online]. Available: <https://www.youtube.com/watch?v=OZ989EvTIBQ>.