Université de Caen Normandie

Projet Annuel I

M1 - IA, Science des Données et Santé

Année 2023 - 2024

Développement d'une interface pour l'utilisation d'un algorithme de Deep Learning de segmentation de lésions cérébrales sur des images IRM

Projet réalisé par :

ANDRES Romain
CORROLLER Typhaine
LADUREE Luca
OROU-GUIDOU Amirath Fara

Sous la supervison de :

LECHERVY Alexis
CORROYER-DULMONT Aurélien

Second semestre 2023-2024

Table des matières

Introduction

Ce rapport fait la synthèse de la seconde partie de notre projet annuel visant à créer une interface pour l'utilisation d'un algorithme de Deep Learning de segmentation de lésions cérébrales sur des images IRM. Avant de lire ceci, il est recommandé de lire le rapport du premier semestre. Pour rappel, il fait suite au projet de stage de M2 de Raphaëlle Lemaire au centre de recherche François Baclesse de Caen au cours duquel elle a développé l'algorithme de Deep Learning que nous avons manipulé.

1 Rappel des avancées du premier semestre

Au cours du premier semestre, nous avons acquis une base solide pour terminer le projet. Premièrement, nous avons configuré un serveur Dicom Web Orthanc[7] afin de manipuler les images médicales au format DICOM en local sur notre serveur web. Nous avons également confectionné une API permettant d'interagir avec ce serveur et permettre la gestion des données médicales. Cette API a la particularité d'être codée en Python grâce au framework Flask[2], ce qui nous permet de facilement intégrer l'algorithme de Deep Learning. L'interface web provient en grande partie du projet OHIF[6], à laquelle nous avons ajouté une extension pour permettre l'utilisation d'un algorithme de segmentation (Deep Learning) avec notre API, qui, à la fin du premier semestre, était capable de simuler une génération de RT-Struct en envoyant un fichier RT-Struct de vérité terrain.

2 Objectifs du semestre

Le plan de ce second semestre était d'avoir un projet fonctionnel et de présenter un pipeline complet. Pour cela, il était nécessaire d'avancer sur la partie Back-End du projet sur 2 axes : le perfectionnement de l'API et le support de l'algorithme de Deep Learning dans le pipeline final. Nous nous sommes également fixé l'objectif de terminer l'interface web et d'y ajouter une interface de suivi des patients, et pour finir, de confectionner une documentation vis-à-vis des développeurs et une documentation pour les professionnels de santé.

3 Organisation et planification

Nous avons décidé de nous répartir les tâches sur le Front-End et le Back-End du projet comme le semestre précédent. Romain s'est chargé de développer une bonne partie de l'API et de l'intégration d'une vraie preuve de concept que notre projet fonctionne et plus tard du modèle de deep learning, et du suivi des patients (voir section 6.1). Armirath

s'est chargée de l'intégration du modèle de Deep Learning dans l'API ; Luca de travailler sur l'API en intégrant le suivi des patients ; et Typhaine de l'UI de OHIF ainsi qu'à sa documentation technique pour les médecins.

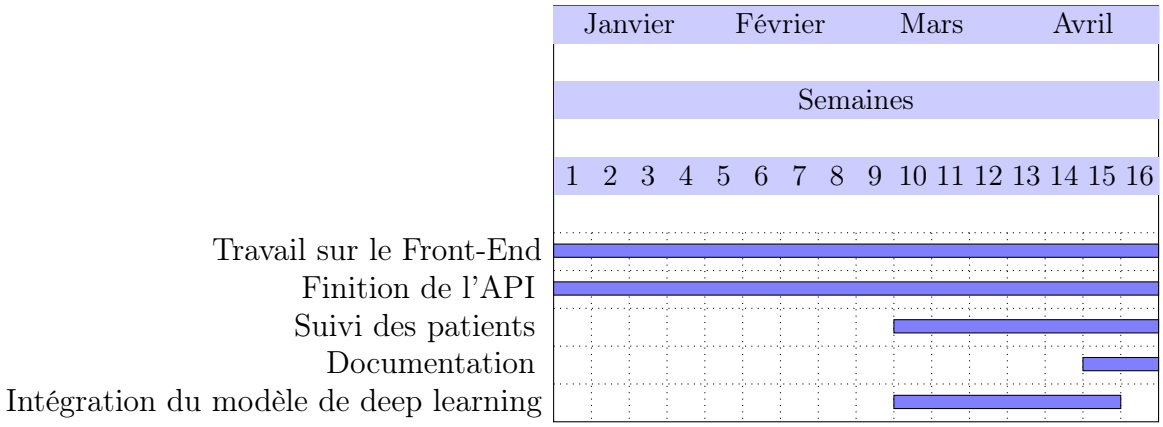

FIGURE 1 – Diagramme de Gantt du projet.

4 L'interface graphique

Tout au long de ce second semestre, nous avons pu ajouter différentes fonctionnalités sur le Front-End.

4.1 Bouton Segmentation

Dans un premier temps, nous avons fait l'ajout d'un bouton sur l'interface de visualisation d'une étude. Ce bouton est en lien avec l'API, il va nous permettre d'appeler le modèle et de générer des labels qui vont être transformés en RT-Struct, qui seront par la suite envoyés sur le DICOM Web Server Orthanc puis visualisables sur la page de l'étude. En appuyant sur ce bouton, un message s'affichera pendant l'exécution de notre code, indiquant également le temps de chargement. Une fois cette étape terminée, le message s'actualisera pour laisser comprendre à l'utilisateur que la tâche demandée s'est réalisée avec succès.

Ce bouton nous permet de lier le Front-End et le Back-End, afin d'afficher directement les RT-Struct pour l'utilisateur sans manipulations parasites, faisant donc gagner du temps et rendant notre interface plus accessible. Il a été placé ici afin de minimiser le nombre de clics et déplacements de souris.

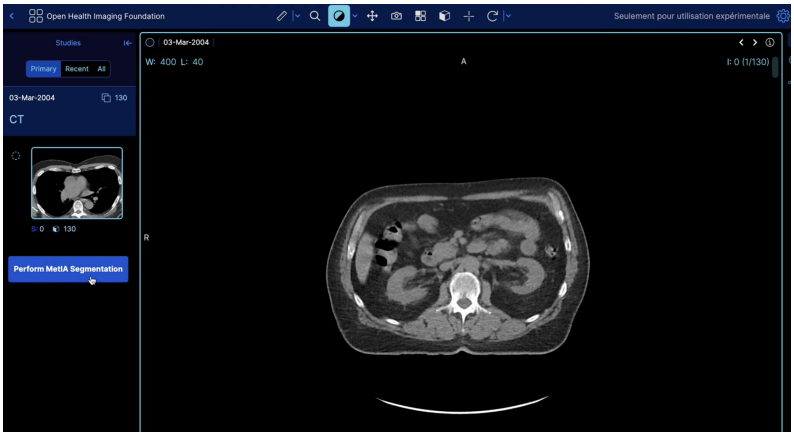

FIGURE 2 – Bouton pour lancer le modèle

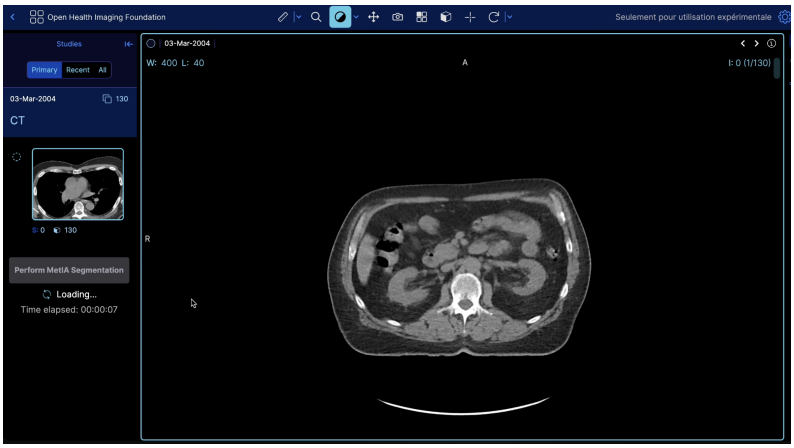

FIGURE 3 – Modèle en exécution

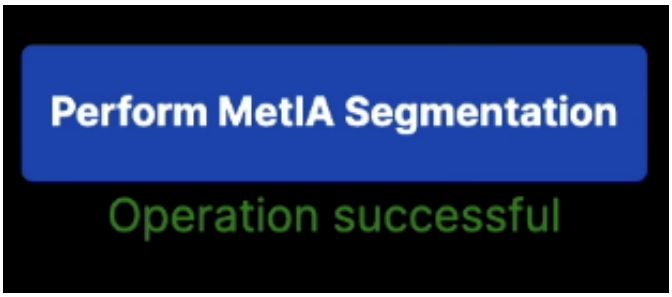

FIGURE 4 – Exécution terminée

4.2 Bouton pour supprimer une étude

En dehors de la visualisation d'études, nous avons également fait des modifications dans la liste des études et ajouté un menu à notre interface. Dans la liste des études, nous avons ajouté un bouton "poubelle" permettant à l'utilisateur de supprimer une étude directement depuis l'interface d'Ohif, ce qui était possible avant uniquement depuis l'interface du DICOM Web Serveur. En appuyant sur cette icône, une fenêtre pop-up va

s'afficher nous demandant une confirmation de suppression. Cela nous permet de tout gérer uniquement sur l'interface OHIF pour simplifier la manipulation par les médecins.

FIGURE 5 – Icône permettant la suppression d'une étude

FIGURE 6 – Confirmation de suppression

4.3 Page d'accueil

De plus, nous avons donc créé un menu reliant plusieurs fonctionnalités.

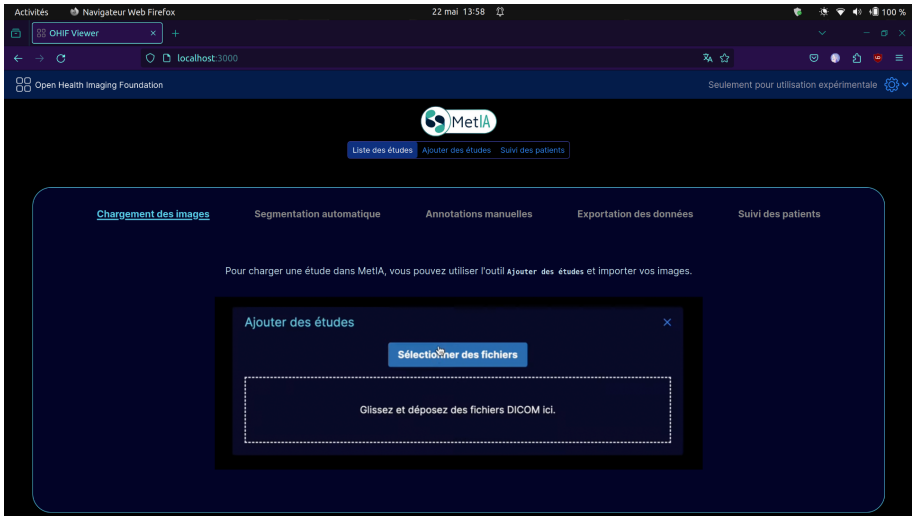

FIGURE 7 – Page d'accueil de notre interface

Dans un premier temps, le menu offre à l'utilisateur un aperçu de ce qu'il peut faire avec l'interface, le tout dans un "carrousel", montrant des images de démonstration des différentes fonctionnalités.

Ensuite, ce menu permet d'ajouter une étude dans la base de données, en ayant le choix de sélectionner soit même les fichiers correspondants, ou en utilisant le "drag and drop" directement. En arrière-plan, cela va envoyer les données au serveur Orthanc automatiquement en affichant le pourcentage de chargement, et l'étude sera ajoutée à la liste sans que l'utilisateur ait besoin de changer de page.

FIGURE 8 – Ajout de DICOM

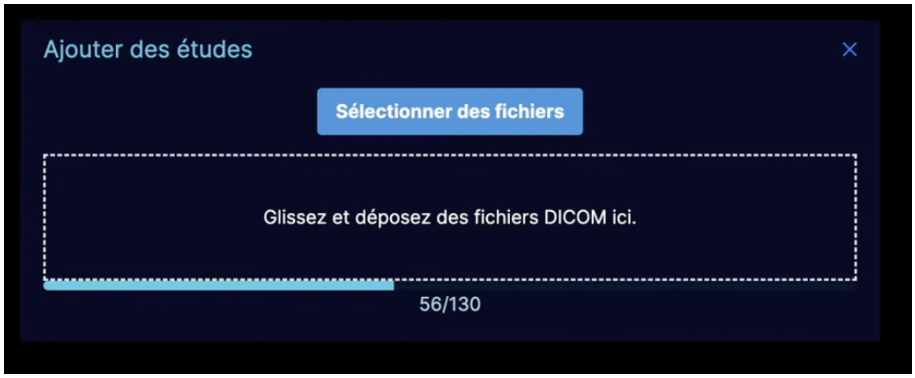

FIGURE 9 – Chargement de l'upload

De plus, le bouton le plus à gauche nous permet d'accéder à la liste des études classique, qui correspond à l'interface sur laquelle nous arrivions avant en lançant le Front-End. Ici sont affichées les études préalablement importées. Nous pouvons voir des informations à leurs propos et également lancer la visualisation des images en cliquant sur "Basic Viewer" (ce qui nous amène sur la page où l'on peut faire la segmentation).

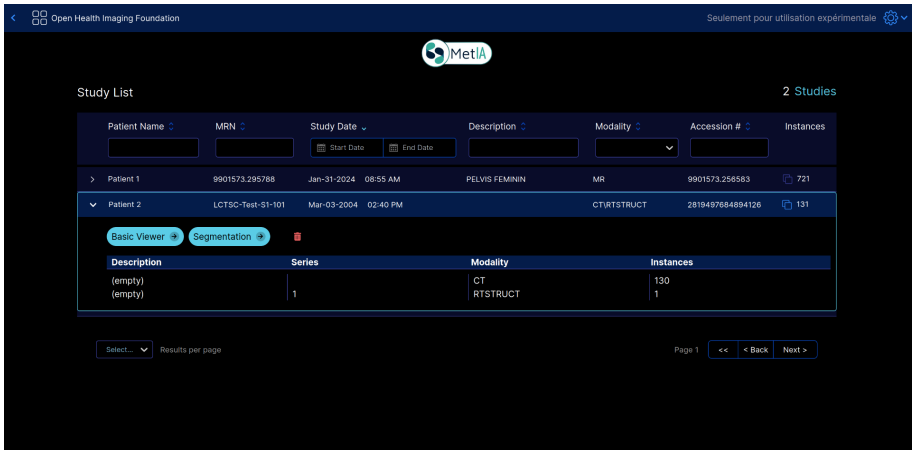

FIGURE 10 – Liste des études

Le dernier bouton, le plus à gauche, va nous permettre d'accéder à la liste des patients.

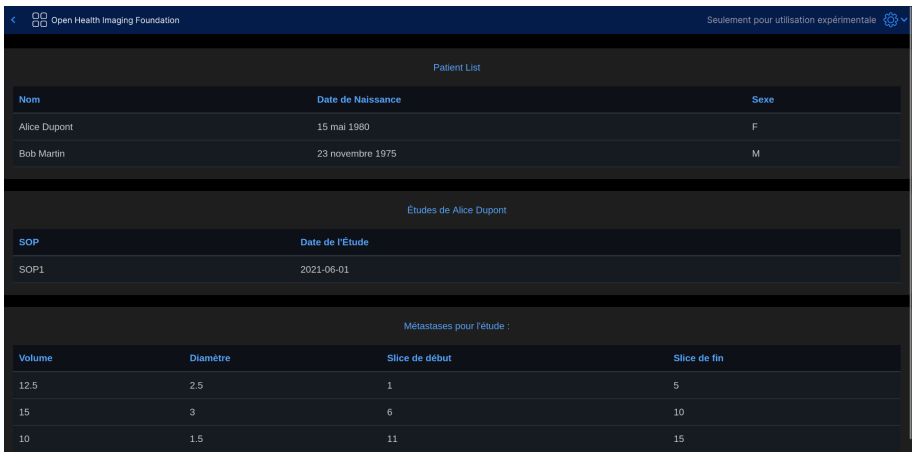

FIGURE 11 – Liste des patients

Cette fonctionnalité va permettre à l'utilisateur de pouvoir observer une liste cette fois-ci triée par patients, affichant toutes les données les concernant, et pouvant grâce à cela, analyser les études dans le temps pour voir si un traitement fonctionne par exemple. Nous avons pu élaborer cette page en créant une nouvelle route. Étant donné que nous avions réussi vers la fin du projet à comprendre comment était configuré le routing au travers de OHIF (c'est à dire les différentes URL et pages associées), nous avons pu mettre la page de suivi des patients sur la route /TRACKING. Cela permet une gestion plus propre sur le web des différentes pages ainsi que de gagner en visibilité.

5 L'API

5.1 Routes et fonctionnalités de l'API

Notre API permet essentiellement au front de manipuler les données DICOM présentes sur le DICOM Web Server Orthanc, de récupérer les données de la base de données SQLite de suivi des patients, ainsi que d'utiliser le modèle de Deep Learning. Voici un récapitulatif des différentes routes dans la table 1.

Méthode	**Route**	**Description**
GET	/getAllStudies	Récupère la liste des études DICOM stockées dans le serveur Orthanc.
GET	/getStudy/{study_id}	Récupère les détails d'une étude DICOM spécifique à partir de son ID Orthanc.
GET	/getStudyDicoms/{study_id}	Récupère les instances DICOM pour une étude spécifique à partir de son ID Orthanc.
POST	/uploadDicom	Permet d'uploader un fichier DICOM vers le serveur Orthanc.
DELETE	/delete-study/{study_instance_uid}	Permet de supprimer une étude DICOM du serveur Orthanc.
POST	/segmentation/{study_instance_uid}	Permet de lancer la segmentation d'une étude DICOM spécifique et de générer un RT-Struct.
GET	/followup-etudes	Récupère les études de la base de donnée de suivi des patients. Paramètre optionnel : {id-Patient}.
GET	/followup-patients	Récupère les patients de la base de donnée de suivi des patients.
GET	/followup-metastases	Récupère les métastases de la base de donnée de suivi des patients. Paramètre optionnel : {idEtude}.

Table 1 – Description des routes et fonctionnalités de l'API

Pour y voir plus clair, voici un aperçu du cheminement pour générer un RTStruct à partir de notre API depuis le front, en partant du principe qu'il n'y a aucun problème.

Vous avez ci-dessous le diagramme de séquence montrant le chemin du Back-End pour une génération de RT-Struct.

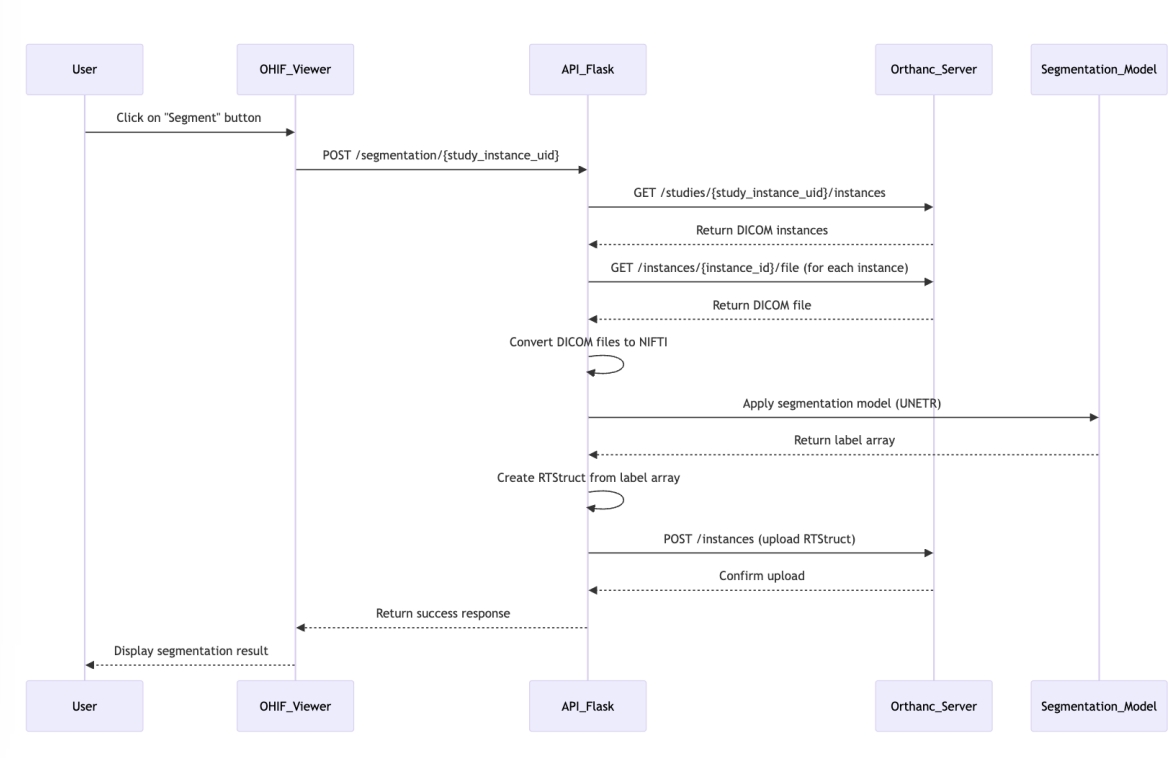

FIGURE 12 – Diagramme de séquence montrant le chemin Back-End pour une génération de RT-Struct

5.2 Une simulation des conditions réelles

Comme nous l'avons discuté dans le premier rapport, l'algorithme de Deep Learning n'étant pas exécutable sur nos machines personnelles en raison des demandes computationnelles excessives de l'algorithme de segmentation utilisé, nous avions conçu un **Mock** nous permettant de nous passer de l'appel à l'algorithme de Deep Learning en retournant uniquement une image RT-Struct de vérité terrain. Ce n'est pas suffisant pour une preuve de concept. Donc nous avons modifié l'API afin qu'elle puisse générer des RT-Struct à partir des images DICOM présentes sur le serveur DICOM Web.

Pour ce faire, nous avons décidé d'utiliser une bibliothèque Python assez répandue dans la manipulation et la création de fichiers DICOM : rtutils[10]. Cependant cette bibliothèque possède un défaut au moment où nous l'avons utilisé : la gestion des images DICOM se fait uniquement sur fichier via un chemin de dossier à entrer ce qui est très contraignant pour une utilisation en API, car chaque appel va créer des opérations sur le disque du serveur.

Ainsi pour palier à ce problème, Romain a décidé de faire un fork du dépôt de la librairie rtutils et de la modifier pour qu'elle inclut une gestion en mémoire des images DICOM. Ainsi avec cette modification, nous pouvons tout faire en mémoire et ne pas user le disque dur grâce à l'utilisation d'un BufferIO. Cela a conduit à un pull request sur GitHub qui pourrait mener à une contribution potentielle au projet rtutils. Pour simplifier le déploiement de cette modification de rtutils, le code modifié a été mis dans le dépôt du back-end.

Pouvant donc créer des fichiers RTStruct en mémoire, nous avons fait une fonction de seuillage qui nous permet de définir les contours des images DICOM à un certain seuil en utilisant un filtre de sobel, une technique de vision par ordinateur apprise dans le cadre de nos études (voir figure 13). Pour développer ceci, tout le pipeline était testé dans des Notebook Jupyter pour faciliter le débogage. Une fois le pipeline correct, il pouvait être intégré dans l'API. Ainsi, cette nouvelle version du Mock sert de vraie démonstration pour les utilisateurs n'ayant pas les ressources computationnelles nécessaires.

FIGURE 13 – Contours calculés en rouge avec notre seuillage sur une image de prostate

5.3 L'implémentation de l'algorithme de Deep Learning

L'algorithme de Deep Learning utilisé pour la segmentation de lésions cérébrales sur des images IRM est le modèle **UNETR**. C'est un modèle du challenge BTCV sur **Monai** [8] utilisé pour la segmentation d'images médicales en trois dimensions. Ce modèle a été choisi et fine-tunné par Raphaëlle LEMAIRE, lors de son stage au centre François Baclesse. Ce modèle donne de meilleurs résultats parmi tout autre modèle (CoTr, TransUNet ou nnUNet). L'intérêt de la fusion de UNet avec les Transformers vient de la capacité de UNet à segmenter des images médicales en trois dimensions et des Transformers, initialement utilisés pour le traitement du langage, à retourner les relations à longue distance entre les éléments d'une même séquence. Ainsi, dans UNETR, le mécanisme des Transfor-

mers est adapté au traitement d'images en les divisant en patch de taille fixe (dans notre projet $96 \times 96 \times 96$ pixels) qui sont ensuite transformés en séquences. Les séquences ainsi créées sont fournies aux Transformers. Une fois passées à ces derniers, leurs mécanismes d'attention vont se concentrer sur les parties pertinentes de l'image en calculant les pondérations pour chaque patch en fonction de sa relation avec les autres patch. Cela permet au modèle d'apprendre la segmentation des données en se basant aussi sur le contexte et les interactions entre les différentes parties d'une image. Une fois les informations capturées par les Transformers, la partie UNet du modèle fusionne ces dernières et crée une carte de segmentation de l'image.

5.3.1 Architecture du modèle

UNETR ou **UNet** basé sur **Transformers**, est une architecture qui modifie l'approche traditionnelle de la segmentation par UNet en intégrant les Transformers pour le traitement des caractéristiques spatiales. Le modèle utilise des blocs de Transformers pour capturer des dépendances à longue distance, améliorant ainsi la capacité du modèle à comprendre les contextes plus larges des images médicales.

Le modèle est configuré avec des paramètres tels que '*in_channels*', '*out_channels*', '*roi_size*' etc., pour personnaliser sa capacité à traiter différentes tailles d'images et de caractéristiques. Les blocs de Transformers sont complétés par des couches de convolution et de résidus lorsque cela est spécifié, permettant une fine personnalisation du traitement des caractéristiques. Un article [3] fut publié en 2021 sur ce modèle par Ali Hatamizadeh.

Son architecture est visible sur la Figure 14.

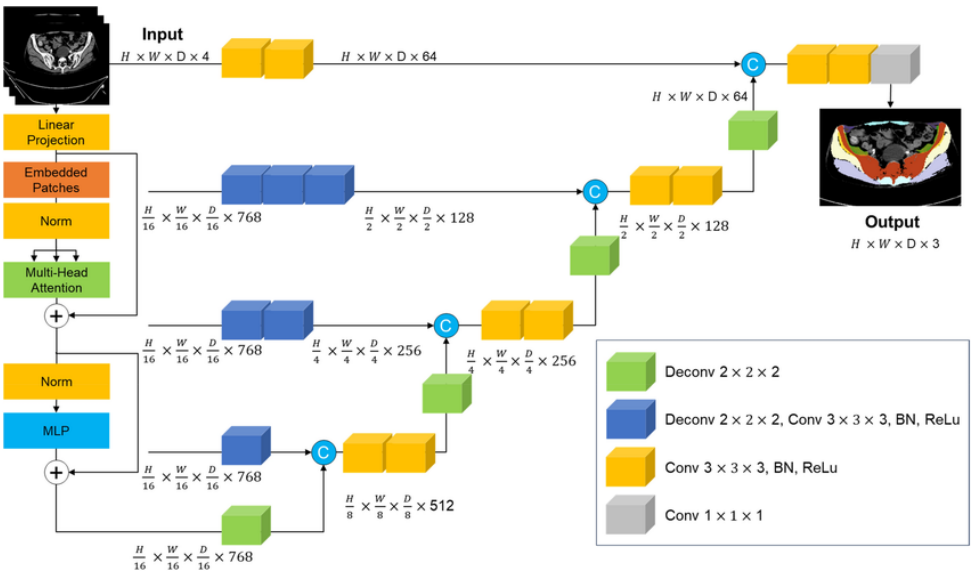

FIGURE 14 – Architecture de UNETR

Cette architecture conçue spécifiquement pour la segmentation médicale, utilise les Transformers pour améliorer la capture des caractéristiques spatiales complexes dans les images médicales. La figure ci-dessus illustre les différentes composantes du modèle détaillant le flux de données à travers le réseau.

Cette architecture est composée de :

- **Image d'entrée** : Une image médicale tridimensionnelle (dans notre cas ce sont des IRM) en entrée avec des dimensions $'\mathbf{H\ x\ W\ x\ D\ x\ 4}'$.
- **Projection linéaire** : Les données sont d'abord projetées linéairement pour augmenter la profondeur de caractéristiques à 64.
- **Patch Incrustés** : L'image est découpées en patch qui sont ensuite incorporés pour former une représentation dense de l'image.
- **Normalisation** : Chaque patch incrusté passe par une couche de normalisation avant d'être traité par le bloc de multi-têtes d'attention.
- **Attention Multi-Têtes** : Les interactions entres les patch sont modélisés à ce stade, permettant au modèle de comprendre les contextes globaux à l'intérieur de l'image.
- **Perceptron Multi-couches (MLP)** : Après l'attention, les caractéristiques passent par un MLP pour une transformation non-linéaire.
- **Convolutions et Dé-convolutions** : Les caractéristiques subissent une série de convolutions et de dé-convolutions pour reconstruire l'image à partir des représentations denses. Ceci comprend les convolutions pour affiner les détails et des dé-convolutions pour reconstruire l'image à partir des représentations denses.
- **Sortie(Segmentation de l'image)** : L'image de sortie est une carte de segmentation avec les dimensions $'HxWxDx3'$, où chaque canal correspond à une classe spécifique dans la segmentation.

5.4 Fine-tuning de UNETR

Le fine-tuning d'un modèle est un procédé permettant de démarrer l'apprentissage du modèle avec des poids adapté à notre objectif au lieu de poids aléatoires. Ce dernier a été fait par Raphaëlle. Cela permet de rendre plus rapide l'apprentissage et d'obtenir de meilleures performances que celles qu'il aurait obtenues sans fine-tuning. Ce procédé se base sur des poids calculés par un modèle préentraîné. Celui qu'elle a utilisé était fourni

dans le code du projet MONAI retrouvable sur github.com [8].

Il existe de nombreuses méthodes pour finetuner un modèle. Dans le projet, notre choix s'est porté sur la technique présente dans l'article de Jason Yosinski[4], où l'objectif est de réinitialiser les poids d'une ou plusieurs des dernières couches du réseau tout en conservant ceux calculés sur les autres couches. Les poids des dernières couches sont plus spécifiques aux données utilisées lors de l'entraînement du modèle pré-entraîné que les poids des premières couches, qui eux sont plus généraux. Les couches non réinitialisées sont gelées durant les premières époques de l'entraînement, pour entraîner uniquement les dernières couches avec nos données. Procéder ainsi permet d'adapter les poids des dernières couches à notre problème tout en conservant les connaissances des caractéristiques communes aux images d'IRM extraites grâce aux poids des premières couches.

Dans ce projet, c'est la dernière couche du modèle qui a été remplacée, permettant ainsi d'ajuster le nombre de classes aux données. Notre problème étant binaire, il y a ou non une lésion cérébrale, la dernière couche du réseau renverra deux canaux de sorties.

5.5 Modèle entraîné

L'entraînement du modèle ainsi que la génération des poids à partir du modèle entraîné ayant déjà été fait par Raphaëlle, nous vous renvoyons au PDF du nom de **RapportStageM2Baclesse.pdf** dans l'archive du code envoyé pour plus d'informations.

De notre côté il suffisait juste d'adapter ce modèle à nos données et le faire fonctionner via notre API Flask. Tout d'abord, le modèle qu'elle a utilisé a été préentraîné sur des données au format NIFTI, or nos données sont au format DICOM, il fallait donc les convertir. Ce passage du format DICOM vers le format Nifti a été réalisé une fois, à l'aide d'un script Python, avant le lancement du modèle sur les données.

```python
def dicom_to_nifti_in_memory(dicom_datasets: List[pydicom.Dataset]) -> nib.Nifti1Image:
    image_slices = [ds.pixel_array for ds in dicom_datasets]
    volume_3d = np.stack(image_slices, axis=-1)
    affine = np.eye(4)
    nifti_image = nib.Nifti1Image(volume_3d, affine)
    return nifti_image
```

Listing 1 – Conversion de DICOM en NIfTI en mémoire

La fonction utilise les bibliothèques `pydicom`, `numpy` et `nibabel` pour lire les fichiers DICOM, manipuler les tableaux de données et créer l'image NIfTI. Elle prend en entrée une liste de datasets DICOM (`dicom_datasets`). Les tableaux de pixels des images DICOM sont extraits et stockés dans une liste (`image_slices`). Les tranches d'image sont

empilées pour former un volume 3D (`volume_3d`), une matrice affine d'identité (`affine`) est utilisée pour indiquer l'orientation spatiale des voxels et une image NIfTI est créée à partir du volume 3D puis de cette matrice. La fonction renvoie le fichier NIFTI en mémoire des DICOM.

5.5.1 Transformations

Avant de passer les images au modèle, des transformations sont appliquées :

- **transforms.Orientationd** : aligne l'orientation de l'image et du label selon un code d'axe spécifié (ici "LAS" [1]).
- **transforms.Spacingd** : modifie les dimensions des voxels de l'image et du label pour qu'elles correspondent aux dimensions spécifiées par `voxel_space`.
- **transforms.ScaleIntensityRanged** : met à l'échelle l'intensité des pixels de l'image de la plage `[a_min, a_max]` à la plage `[b_min, b_max]`.
- **CropBedd** : recadre l'image et le label pour supprimer les parties inutiles, comme par exemple le lit du patient.
- **transforms.CropForegroundd** : recadre l'image et le label pour enlever les parties de l'arrière-plan sans intérêt.
- **transforms.FgBgToIndicesd** : génère des indices de premier plan et d'arrière-plan basés sur le label.
- **transforms.RandFlipd** : applique une mise en miroir aléatoire selon l'axe des Y (une lésion à droite passera à gauche et inversement)
- **transforms.RandRotate90d** : applique une rotation aléatoire à 90 degrés sur l'image et le label selon une probabilité spécifiée.
- **transforms.RandScaleIntensityd** : applique une mise à l'échelle aléatoire de l'intensité sur l'image selon une probabilité spécifiée.
- **transforms.RandShiftIntensityd** : augmente ou diminue l'intensité des pixels de l'image de façon aléatoire.
- **transforms.ToTensord** : convertit l'image et le label en tenseurs Torch.

1. Le code d'axes "LAS" signifie : **L** pour Gauche (*Left*), **A** pour Avant (*Anterior*), et **S** pour Supérieur (*Superior*)

5.5.2 Application du modèle entraîné aux données

Voici au final le code permettant de lancer le modèle, contenant le pré-traitement des données.

```
def getLabelOfIRM_from_nifti(nifti_image: nib.Nifti1Image, pathModelFile:
    str):
    transform = transformation()
    transformed_image = applyTransforms(transform, nifti_image.get_fdata())

    model = loadModel(pathModelFile)
    dico_image = applyUNETR(transformed_image, model)

    label, imageT = disapplyTransforms(transform, dico_image)
    return nifti_image.get_fdata() / 255, label, imageT
```

Avec ce code, on a en entrée une image au format nifti et le chemin du modèle pré-entraîné. Cette fonction nous retourne l'image NIfTI d'origine normalisée par division par 255, les labels de segmentation prédits par le modèle et l'image après les transformations appliquées. La fonction **applyTransforms** permet d'appliquer les transformations décrites ci-dessus sur les images et la fonction **disapplyTransforms** fait le contraire. Ainsi pour le chargement du modèle, on a la fonction **loadModel** qui se charge de ça. Et pour terminer la fonction **applyUnetr** qui applique le modèle de segmentation UNETR sur les images DICOM converties en NifTI pour prédire les labels de segmentation, puis retourner les résultats.

5.5.3 Exécution

Étant donné que nos machines ne disposent pas d'une puissance de calcul suffisante, lorsqu'on clique sur le bouton pour la segmentation, elles plantent au bout de quelques minutes. Nous ne possédions donc pas de résultats concernant la prédiction, ce qui explique pourquoi nous avons dû faire un mock pour la génération des RT-Struct avec une fonction de seuillage décrite dans cette sous-section.

La figure 15 montre bien que le processus de segmentation est lancé, toutefois nous ne sommes pas parvenus au bout de celui-ci pour des raisons matérielles.

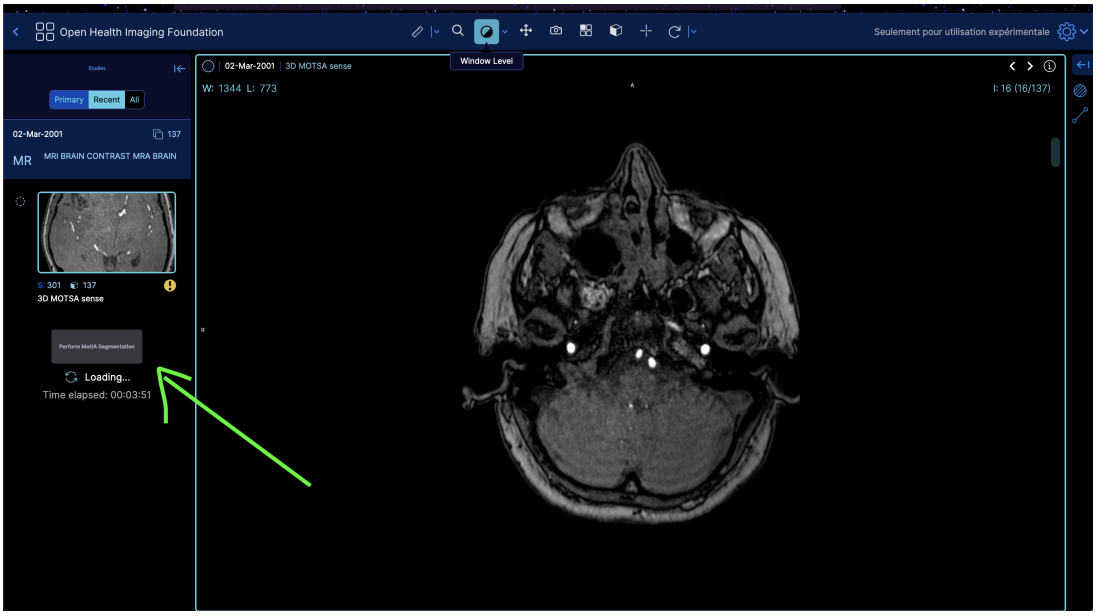

FIGURE 15 – Application du modele

5.6 Le suivi des patients

5.6.1 Idée générale

Mr Corroyer-Dulmont, étant bien au courant des besoins des professionnels de santé, nous a assez rapidement suggéré d'intégrer une interface de suivi des patients. Cette interface permettrait de mieux visualiser l'évolution des métastases chez les patients en y répertoriant les données les plus utiles directement à partir des métadonnées des DICOM transmises à l'application. Cela offrirait en effet aux médecins la possibilité de comparer facilement les volumes et diamètres des différentes métastases en fonction de l'étude et/ou session auxquelles elles appartiennent à différents intervalles de temps.
Pour que le suivi de chaque patient soit optimal, notre objectif était de mettre à jour cette base de données automatiquement à la fin de l'appel à l'API quand le fichier RT-Struct à été généré, étant donné que c'est à cette étape du processus que nous pouvons accéder aux dimensions des régions d'intérêt.

5.6.2 Développement

Ayant travaillé plusieurs fois avec des bases de données SQLite[9] au cours de notre formation universitaire, nous avons donc logiquement décidé d'utiliser cette structure pour stocker nos informations. La table 2 représente la structure de notre base de données et la répartition de celles-ci en 3 tables 'Etude', 'Patient' et 'Metastase'. Cette granularité provient du fait que les métastases sont réparties en fonction des études où elles ont été enregistrées, et que les études concernent un patient précis.

TABLE 2 – Modèle de la base de données SQLite

Table	Champ	Description
etude	idEtude	Clé primaire, incrément automatique
	idPatient	Identifiant du patient
	idSerie	Identifiant de la série
	idSOP	Identifiant SOP
	dateTraitement	Date de traitement
patient	idPatient	Clé primaire, incrément automatique
	nom	Nom du patient
	dateNaissance	Date de naissance du patient
	sexe	Genre du patient (binaire)
metastase	idMetastase	Clé primaire, incrément automatique
	idEtude	Identifiant de l'étude
	volume	Volume de la métastase
	diametre	Diamètre de la métastase (peut être nul)
	slideDebut	Début de la tranche d'image
	slideFin	Fin de la tranche d'image

Le modèle de la table a par ailleurs subi une légère transformation par rapport à celui d'origine. En effet, nous avions initialement choisi de représenter les sessions dans notre base à la place des patients, mais nous nous sommes rendu compte qu'il était plus pertinent de considérer les patients comme une table à part entière, et que le degré de précision qu'apportait la table de session n'était pas réellement utile. La figure 16 vous présente une première version de notre visualisation de la base de données via l'API Flask et des requêtes sur les tables. Chaque étage de la hiérarchie "Études / Sessions / Métastases" était déroulable, permettant à l'utilisateur de cibler les éléments qui l'intéressaient.

Le code de la base de données actuelle se situe dans le fichier **MesuresSQLite.py**. Ce dernier contient donc toute l'architecture de la base ainsi que les méthodes associées pour récupérer, ajouter ou supprimer des données avec notre API. Pour ce faire, nous avons utilisé la bibliothèque python **sqlite3** [9]. Notre choix s'est porté sur celle-ci en raison de son indépendance réseau et de sa prise en charge des transactions ACID [2]. En effet, SQLite3 stocke les données localement dans un fichier de base de données (**mesures.db** dans notre projet) et permet donc à notre fonctionnalité d'être protégée d'éventuels problèmes

2. Le terme "ACID" signifie : **A** pour Atomicité (*Atomicity*), **C** pour Cohérence (*Consistency*), **I** pour Isolation (*Isolation*), et **D** pour Durabilité (*Durability*).

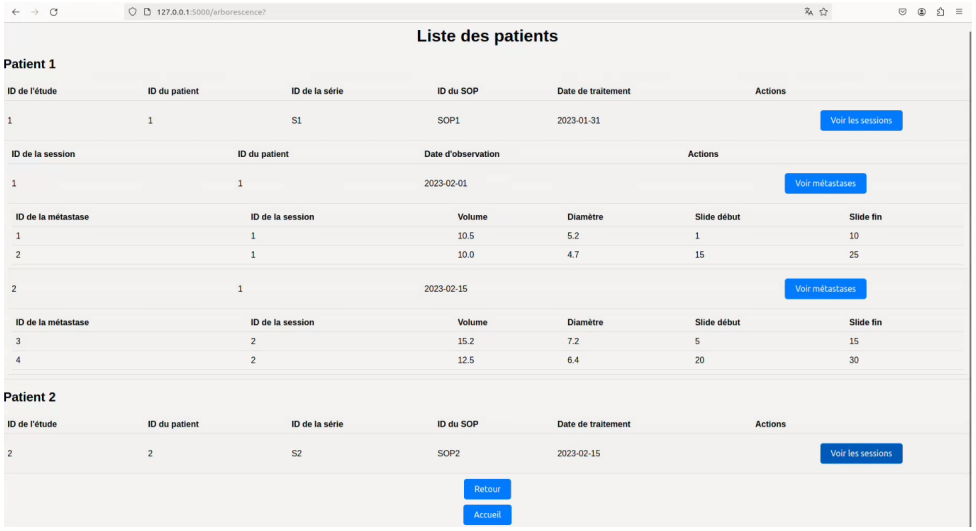

FIGURE 16 – Première version de la visualisation de la BDD

réseaux. Les transactions ACID quant à elles renforcent la robustesse de notre application puisqu'elles garantissent l'intégrité des données même en cas de panne ou d'erreur, ce qui nous paraissait primordial étant donné que l'on travaille avec des données médicales, donc particulièrement sensibles.
L'affichage des données sur le front est géré par notre API Flask (**api.py**). Cette dernière va faire le lien entre ce que voit l'utilisateur sur le Viewer d'OHIF et les données présentes sur dans le fichier **mesures.db**. En effet, la page de tracking (Figure 11) et les données qu'elle contient sont uniquement les réponses des requêtes de l'API sur notre base SQLite.

5.6.3 Illustration

Prenons pour exemple l'affichage de la liste des études. Pour un patient présent dans la base, nous effectuons un appel à la méthode *get_ etudes()* de l'API qui va chercher dans un premier temps si un idPatient a été entré dans les arguments de la requête côté Front. De fait, nous avons prévu le cas d'un affichage global de toutes les études, mais en réalité, c'est la visualisation des études relatives à un seul patient qui va nous intéresser. C'est la méthode *get_ etudes_ from_ patient* prenant en argument un id de patient qui se charge d'interroger la BDD avec une simple requête SQL (Voir ici). Le résultat de cette requête est converti en json avec la méthode *jsonify* puis envoyé au Front afin de l'afficher sur notre page de tracking sur OHIF.

```python
### api.py
@app.route('/followup-etudes', methods=['GET'])
def get_etudes():
    id_patient = request.args.get('idPatient')
    if id_patient is not None:
        etudes = get_db().get_etudes_from_patient(id_patient)
    else:
        etudes = get_db().get_etudes()
    return jsonify(etudes)

### MesuresSQLite.py
# Méthode globale
def get_etudes(self):
        self.curseur.execute("SELECT * FROM etude")
        etudes = self.curseur.fetchall()
        return [{"id_study": etu[0], "id": etu[1], "id_serie": etu[2], "id_SOP": etu[3], "date": etu[4]} for etu in etudes]
# Méthode ciblée sur un patient (ID)
    def get_etudes_from_patient(self, id_patient):
        self.curseur.execute("SELECT * FROM etude WHERE idPatient = ?", (id_patient,))
        etudes = self.curseur.fetchall()
        return [{"id_study": etu[0], "id_serie": etu[2], "id_SOP": etu[3], "date": etu[4]} for etu in etudes]
```

Listing 2 – Pipeline des méthodes pour l'affichage des études

6 La documentation

Pour compléter notre projet, nous avons décidé de créer une documentation à destination professionnelle, pour les possibles futurs utilisateurs.

Dans cette documentation, nous donnons dans un premier temps un contexte au projet actuel, ce qui nous permet de faire un bref résumé du contenu et du but de l'interface.

Dans un second temps, nous donnons les configurations minimales nécessaires pour le lancement et la bonne utilisation du projet.

Ensuite va être développé le contenu de l'interface plus en détail, incluant les différentes étapes de manipulations ainsi que des images pour illustrer celles-ci. Cela va permettre de faire, en quelque sorte, une démonstration des fonctionnalités que nous avons implémentées depuis le début de notre projet, tout comme de celles qui sont déjà présentes sur le Viewer de OHIF comme les annotations et mesures sur images.

Pour plus d'informations sur notre documentation, nous vous invitons à regarder le PDF se nommant **documentationMedecins.pdf** dans l'archive du rendu.

En ce qui concerne les développeurs, tout notre code est bien commenté et les README des dépôts du Front et du Back ont été minutieusement faits afin de permettre une compréhension totale de notre code.

7 Axes d'améliorations

Le rendu de notre projet offre les fonctionnalités principales permettant une première utilisation de l'application. Toutefois, nous avons pu identifier plusieurs pistes d'améliorations qui pourraient être explorées à l'avenir.

7.1 Sécurité

Pour être réellement utilisée dans une routine clinique, il faudrait gérer l'aspect cybersécurité de notre application, notamment via l'inclusion de proxys et de certificats. Notre objectif était d'apporter une preuve de concept sans réellement se soucier de cet aspect, mais nous sommes conscients que la manipulation de données médicales sensibles est un sujet critique qui se doit d'être priorisé en vu d'un déploiement dans un environnement hospitalier.

7.2 Visualisation

Bien que secondaire, une bonne visualisation des données de suivi renforcerait la pertinence de notre application. Pour l'instant, nous affichons simplement les données brutes, mais il est tout à fait envisageable de les représenter sous forme de graphes (avec Matplotlib [5] par exemple), ou bien même d'utiliser des librairies dans le style de D3 [1], qui offrent des visualisations très élaborées.

7.3 Alimenter le suivi

La visualisation de données que nous proposons repose sur des données Mock, écrites en brutes et qui faisaient office de test. Pour exploiter pleinement notre application, il faudrait développer l'alimentation de la base de données qui concerne le suivi des patients. En effet, lorsque nous ajoutons des images DICOM pour effectuer une segmentation, l'idéal serait que les données post-segmentation (volume, diamètre, id en tout genre...) soient directement transmises à nos tables.

7.4 Exportation des données

Nous avons oublié de gérer cette fonctionnalité dans les boutons de OHIF. Il serait bien que l'on puisse exporter les données sur OHIF.

7.5 Perfectionner le visuel

Le Viewer d'OHIF offre une expérience satisfaisante pour l'utilisateur que nous tenions à préserver lors du développement de notre extension. Nous estimons avoir bien accompli cette tâche : notre proposition est visuellement agréable et assez facile à comprendre. Toutefois l'aspect visuel peut sans doute être perfectionné, en rendant l'application plus adaptative par exemple, ou encore en appliquant la charte graphique de OHIF au suivi du patient, etc.

8 Conclusion

Le développement de notre application a représenté une expérience enrichissante et formatrice, et est une réussite globale même si le projet peut encore être amélioré. Quelques détails restent à parfaire comme l'aspect visuel de la page de suivi des patients et son raccordement au pipeline de l'API après une génération de RT-Struct. Néanmoins l'objectif de pouvoir lancer le modèle et d'obtenir les résultats sur le DICOM Web Server puis de pouvoir les visualiser avec une belle interface web a été accompli.

Ce projet a été très intéressant dans le cadre de notre cursus scolaire, car il nous a mis dans un rôle entre chercheurs et médecins. Les retours réguliers de Mr. CORROYER-DULMONT, responsable du pôle intelligence artificielle du centre de recherche Baclesse à Caen, et de Mr. LECHERVY, professeur d'Apprentissage à l'université, nous ont permis d'affiner nos connaissances en informatique et de les mettre en relation avec le monde de la santé et des contraintes qui lui sont associées. Cela nous a fait comprendre une multitude de choses sur le Deep Learning que nous n'aurions pas aussi bien assimilées en ne participant pas à ce projet. Pouvoir manipuler un algorithme de Deep Learning aussi avancé, surtout dans le domaine médical était une véritable chance pour nous. C'était également un véritable challenge, car il n'était vraiment pas évident de comprendre la base du code sur laquelle nous sommes partis, la gestion de l'API n'était pas évidente non plus et le projet OHIF possède une énorme architecture bien complexe sous-divisée en plusieurs sous-projets.

Ainsi nous avons mis au point une API robuste permettant au serveur web de gérer des données médicales issues d'un serveur DICOM web en local, de pouvoir les afficher avec énormément de fonctionnalités issues du projet OHIF, et avons maîtrisé l'intégration d'un modèle de Deep Learning dans notre API avec une génération de données médicales conformes (Format DICOM).

Références

[1] D3 : Documentation de la librairie javascript d3.
Disponible ici : https ://d3js.org/.

[2] Flask : Framework python pour faire des api.
Disponible ici : https ://flask.palletsprojects.com/en/3.0.x/.

[3] Hatamizadeh ali, tang yucheng, nath vishwesh, yang dong, myronenko andriy, landman bennett, roth holger, and xu daguang. unetr : Transformers for 3dmedical image segmentation. 2021.

[4] Jason yosinski, jeff clune, yoshua bengio, and hod lipson. how transferable are features in deep neural networks ? advances in neural information processing systems, 27, 2014.

[5] Matplotlib : Documentation de la librairie python matplotlib.
Disponible ici : https ://matplotlib.org/.

[6] OHIF Viewer : An open source framework for medical imaging. Available at : https ://ohif.org.

[7] Orthanc : Open-source, lightweight dicom server.
Disponible ici : https ://www.orthanc-server.com/.

[8] Project-monai research-contributions : https ://github.com/project-monai/research-contributions.

[9] sqslite3 : Documentation de la librairie python sqlite3.
Disponible ici : https ://docs.python.org/3/library/sqlite3.html.

[10] Aashish Shrestha, Andrew Watkins, Fatemeh Yousefirizi, Arman Rahmim, and Catalina F. Uribe. Rt-utils : A minimal python library for rt-struct manipulation, 2024. arXiv preprint arXiv :2405.06184.