SEGMENTATION MULTIMODALE DES TUMEURS CÉRÉBRALES

Amadou DIA

14 février 2022

1 INTRODUCTION

Challenge 2020 lancé par Brats. Dans le but d'évaluer les méthodes de pointes pour la segmentation des tumeurs cérébrale dans les IRM multimodales.
Il utilise principalement des IRM préopératoires multi-institutionnelles et se concentre principalement sur la segmentation (Tâche 1) des tumeurs cérébrales intrinsèquement hétérogènes (en apparence, forme et histologie) , à savoir les gliomes.De plus, pour cerner la pertinence clinique de cette tâche de segmentation, BraTS'20 se concentre également sur la prédiction de la survie globale des patients (Tâche 2) et entend évaluer l'incertitude algorithmique dans les segmentations tumorales (Tâche 3).

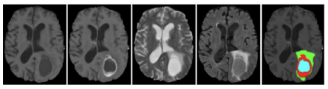

FIGURE 1 – Images IRM de l'ensemble de données BraTS

2 ETUDE DU DATASET

Notre jeu de données est stocké au format NIfTI (.nii.gz) et nous utiliserons la bibliothèque NiBabel pour interagir avec les fichiers.
Pour chaque sujet, on nous donne quatre images IRM, c'est-à-dire quatre volumes tridimensionnels :

- **Natif (T1)**
- **Pondération T1 post-contraste (T1CE)**
- **Pondéré en T2 (T2)**
- **fluide T2 volumes de récupération d'inversion atténuée (FLAIR)**

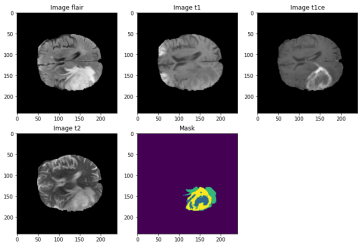

FIGURE 2 – les differentes images de chaque sujet : T1, , T2, T1C, FLAIR, Mask.

Tous les jeux de données d'imagerie ont été segmentés manuellement et ont été approuvés par des neuro-radiologues expérimentés. Ainsi, Chaque exemple a été segmenté et étiqueté par 5 évaluateurs :

- **Étiquette 0 : volume non étiqueté**
- **Étiquette 1 : noyau tumoral nécrotique et non rehaussé (NCR/NET)**
- **Étiquette 2 : œdème péri-tumoral (ED) (un gonflement)**
- **Étiquette 3 : manquante (aucun pixel dans tous les volumes contenant l'étiquette 3)**
- **Étiquette 4 : tumeur rehaussée de GD (ET)**

La tâche d'apprentissage automatique consiste alors à étiqueter chaque voxel avec l'une de ces 5 étiquettes.
Par ailleurs, Lors de la comparaison des performances dans le cadre du défi BRATS, l'évaluation est basée sur trois combinaisons différentes d'étiquettes : complète, de base et d'amélioration. Donc l'algorithme sera formé sur les cinq étiquettes, mais lorsqu'on l'évalue, on l'évalue comme s'il s'agissait de trois tâches de classification binaire différentes.
Nous avons accès à un total de 369 images d'entraînement que nous allons diviser en un ensemble de données d'entraînement (80%) et de validation (20%).

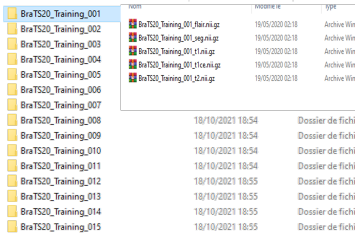

FIGURE 3 – les fichiers du dataset

3 DEROULEMENT

Pour la réalisation du projet, nous allons la scinder en plusieurs étapes à savoir :

- **Etape 1 : Préparer les données**
- **Etape 2 : Définir le générateur de données personnalisé**
- **Etape 3 : Définir un modèle**
- **Etape 4 : Entraînement et prédiction**

Cependant, Parmi les outils et language de developpement qui seront utilisés ; il y'a le **python** comme language et **jupyter** comme outils de developpement

3.1 Préparation des Données

D'abord nous allons télécharger le jeu de donnée puis le décompresser (lien du dataset).
Cependant, Le nom du fichier segmenté dans le dossier **355** a un nom bizarre. il faut le Renommer pour qu'il corresponde aux autres.
Pour une bonne gestion des fichiers **nii**, nous allons utiliser la librairie **nibabel** (lien de nibabel). qui est trés utilisé pour un accés en lecture ou en écriture de certain format de fichiers médicaux et de neuroimagerie courants, notamment **NIfTI1** dans notre cas.
Aprés ces étapes, nous allons effectuer l'ensemble des taches ci desous afin de mieux préparer nos données avant de les entrainer :

- **Mettez à l'échelle tous les volumes (en utilisant MinMaxScaler).**
- **Combinez les trois volumes non-natifs (T2, T1CE et Flair) en un seul volume multicanal car il contienne beaucoup plus d'informations**

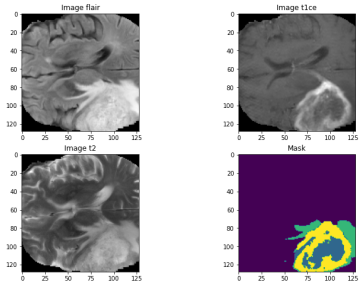

FIGURE 4 – Apres combinaison T2, T1CE et Flair en un seul volume multicanal

- **Réaffecter les pixels de valeur 4 à la valeur 3 (car 3 est absent des étiquettes originales).**
- **Recadrage des volumes pour supprimer les régions vides inutiles autour du volume d'intérêt réel (Recadrage à 128x128x128).**
- **Abandonner tous les volumes où la quantité de données annotées est inférieure à un certain pourcentage. (Pour maximiser la formation sur des volumes réels étiquetés).**
- **Enregistrez tous les volumes utiles sur le disque local en tant que tableaux numpy (npy).**
- **Divisez les volumes d'images et de masques en ensembles de données de formation et de validation.**

Cependant, Le fichier **get_data.py** sera dedié a cette étape de pré-processing.

3.2 Définir le générateur de données personnalisé

Le générateur de données d'image Keras ne fonctionne qu'avec des images **jpg**, **png** et **tif**. Il ne reconnaît pas les fichiers npy. Nous devons définir un générateur personnalisé pour charger nos données depuis le disque.

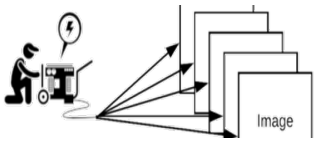

FIGURE 5 – générateur d'images

Cependant, Le fichier **data_generation.py** sera dedié a cette étape de générateur de données.

3.3 Définir un modèle

Nous avons plusieurs choix concernant un modèle de traitement pour la segmentation semantique des images :

- **Modèle U-NET**
- **Modèle FCNl**

Parmi, ces modèles, nous allons faire une ètude dessus puis faire une synthese afin de voir lequel donne de meilleur résultat

3.3.1 Modèle U-NET

U-NET est un modèle de réseau de neurones dédié aux taches de Vision par Ordinateur (Computer Vision) et plus particulièrement aux problèmes de Segmentation Sémantique.

C'est l'un des réseaux de neurones les plus utilisés pour la segmentation d'image. Il s'agit d'un Modèle de Réseau de Neurones Entièrement Convolutif. Ce modèle fut initialement développé par Olaf Ronneberger, Phillip Fischer, et Thomas Brox en 2015 pour la segmentation d'images médicales.

3.3.1.1 Achitecture U-NET

L'architecture de U-NET est composée de deux " chemins ". Le premier est le chemin de contraction, aussi appelé encodeur. Il est utilisé pour capturer le contexte d'une image.
Il s'agit en fait d'un assemblage de couches de convolution et de couches de " max pooling " permettant de créer une carte de

caractéristiques d'une image et de réduire sa taille pour diminuer le nombre de paramètres du réseau.
Le second chemin est celui de l'expansion symétrique, aussi appelé décodeur. Il permet aussi une localisation précise grâce à la convolution transposée.

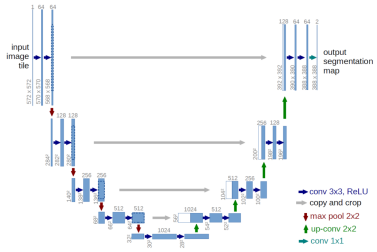

Figure 6 – Architecture U-NET.

3.3.1.2 Avantages

Dans le domaine du Deep Learning, il est nécessaire d'utiliser de larges ensembles de données pour entraîner les modèles. Il peut être difficile d'assembler de tels volumes de données pour résoudre un problème de classification d'images, en termes de temps, de budget et de ressources hardware.
L'étiquetage des données requiert aussi l'expertise de plusieurs développeurs et ingénieurs. C'est particulièrement le cas pour des domaines hautement spécialisés comme les diagnostics médicaux.
U-NET permet de remédier à ces problèmes, puisqu'il s'avère efficace même avec un ensemble de données limité. Il offre aussi une précision supérieure aux modèles conventionnels.
Une architecture autoencoder classique réduit la taille des informations entrées, puis les couches suivantes. Le décodage commence ensuite, la représentation de caractéristiques linéaire est apprise et la traille augmente progressivement. À la fin de cette architecture, la taille de sortie est égale à la taille d'entrée.
Une telle architecture est idéale pour préserver la taille initiale. Le problème est qu'elle compresse l'input de façon linaire, ce qui empêche la transmission de la totalité des caractéristiques.
C'est là que U-NET tire son épingle du jeu grâce à son architecture en U. La déconvolution est effectuée du côté du décodeur, ce qui permet d'éviter le problème de goulot rencontré avec une architecture auto-encodeur et donc d'éviter la perte de caractéristiques.

3.4 Entrainement et Prédiction

On va entrainer le modèle en chargeant des images par lots à l'aide du script générateur de données **data_generation.py**. L'entrainement est effectué via le fichier **train_evaluation.py**.

3.4.1 Entrainement

Le modèle qui sera entrainé sera le U-NET en 3D.

3.4.1.1 Loss Fonction

La fonction de perte qui sera utilisé est **Dice Loss**. Cela est a cause d'un déséquilibre des données qui est un problème courant dans les images médicales. C'est ainsi que **Dice Loss** est utilisé afin de résoudre le problème de déséquilibre des données. Cependant, il ne traite que le problème de déséquilibre entre le premier plan et l'arrière-plan, tout en négligeant un autre déséquilibre entre les exemples faciles et difficiles qui affecte également gravement le processus de formation d'un modèle d'apprentissage. Empiriquement parlant, un exemple facile contribue généralement moins à la perte globale qu'un exemple difficile. Cependant, dans la pratique, par rapport aux exemples concrets, un grand nombre d'exemples faciles seront générés à partir d'une image médicale et domineront le modèle d'entraînement, ce qui entraînera un entraînement sous-optimal ou pire. Pour s'attaquer à ce problème, nous proposons un nouveau **Focal Dice Loss** pour atténuer le déséquilibre entre les exemples difficiles et les exemples faciles. **Focal Dice Loss** est capable de réduire la contribution d'exemples simples et de concentrer le modèle sur des exemples difficiles grâce à notre nouvelle stratégie d'échantillonnage équilibrée proposée pendant le processus de training. Au final, notre fonction de loss sera le total entre le DIce loss et Focal loss.

3.4.1.2 Metric

Le métric utilisé est **MeanIoU**. Mean Intersection-Over-Union **MeanIoU** est une métrique d'évaluation courante pour la segmentation d'images sémantiques, qui calcule d'abord l'IOU pour chaque classe sémantique, puis calcule la moyenne sur les classes. IOU est défini comme suit :
IOU = vrai_positif / (vrai_positif + faux_positif + faux_négatif)

Les prédictions sont accumulées dans une matrice de confusion, pondérées par sample_weight et la métrique est ensuite calculée à partir de celle-ci.
Si sample_weight est None, les poids par défaut sont 1. Utilisez sample_weight 0 pour masquer les valeurs.

3.4.1.3 Résultat de l'entrainement

Pour l'entrainement, le training accyracy est de **97%** et la validation accuracy de **96**. ci dessous les courbes d'évolution :

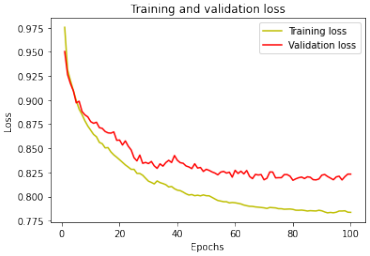

Figure 7 – Training et validation loss

3.4.2 Prediction

Pour la prédiction, sur les données de test, la précision est de **71%** avec **Mean IuO**
Ainsi, ci dessous on voit la prédiction d'une image de test :

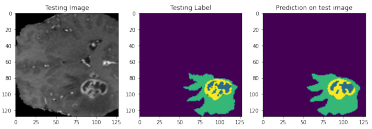

Figure 8 – Training et validation loss

4 Conclusion

La première étape sur l'étude sur les données, permet une grande compréhension du dataset avant de se lancer sur l'entrainement et la prédiction. Ainsi, avec une telle précision **71%** du modèle sur des données en 3D semble abordable mais peut toujours être amélioré soit en faisant une data augmentation ou par d'autre moyen.