Guide d'entraînement et d'exécution de U-Net (Protocole)

Préambule:

Étant donné que les serveurs grace et ada utilisent une architecture différente de celle de nodal (grace et ada fonctionnant sous arm64, tandis que nodal utilise x86), je n'ai pas pu exploiter TensorFlow avec le GPU sur arm64. Cela empêche notre nouveau supercalculateur d'atteindre son plein potentiel. Pour cette raison, j'ai scindé le code en deux parties :

- La première, U-net_Cross_validation_bce-dice.ipynb, ressemble beaucoup au code fourni par M. Abdel. Elle utilise TensorFlow et Keras, ce qui la rend plus lisible et plus facile à comprendre.
- La seconde, U-net_Cross_validation_bce-dice_pytorch.ipynb, repose sur la structure PyTorch et inclut une fonctionnalité d'augmentation des données. Ce code est plus complexe et diffère quelque peu du précédent. Cependant, bien que le traitement diffère entre PyTorch et TensorFlow, le flux d'exécution et la structure du U-net restent globalement similaires.

Pour tirer le meilleur parti de ces codes, il est conseillé de commencer par exécuter et étudier U-net_Cross_validation_bce-dice.ipynb sur le serveur nodal, avant de passer à U-net_Cross_validation_bce-dice_pytorch.ipynb sur les serveurs grace ou ada. Cette approche permet de mieux comprendre leurs points communs et leurs différences.

1. Connexion au serveur

1.1 Connexion au serveur SSH via VS Code

Veuillez vous référer au « **Guide de connexion SSH au serveur Pilcan** » et suivre les instructions pour établir une connexion SSH via VS Code.

---

2. Importation du code et des données

2.1 Fichiers de code

Les fichiers de code utilisés sont :

- U-net_Cross_validation_bce-dice_pytorch.ipynb
- U-net_Cross_validation_bce-dice.ipynb
- tab_seg3D_320_native_recad-dyn
- tab_img3D_320_native_recad-dyn

2.2 Clonage du code

Copiez tous les fichiers dans le dossier que vous avez créé sur le serveur (rappelez-vous que l'espace personnel sur le serveur est commun, c'est-à-dire que vous pouvez accéder sur le serveur nodal aux fichiers que vous avez créés sur le serveur grace).

---

3. Création de l'environnement Python

3.1 Installation et exécution de l'environnement Conda sur le serveur Linux

Sur le serveur **Nodal**, Conda est déjà installé, vous pouvez donc **passer cette étape**.

Sur les serveurs **Grace** et **Ada**, vous devez installer Conda dans votre espace personnel avec la commande suivante : Si Conda n'est pas encore installé, utilisez la commande suivante pour l'installer :

conda install anaconda::conda

Après l'installation, créez et activez l'environnement avec :

conda create -n unet_env python=3.12 -y

conda activate unet_env

3.2 Installation des dépendances

Tensorflow version :2.18.0

torch version: 2.6.0+cu126

torchvision version: 0.21.0

scikit-learn version: 1.6.1

OpenCV version: 4.11.0

NumPy version: 2.0.2

pydicom version: 3.0.1

SimpleITK version: 2.4.1

Matplotlib version: 3.10.0

Albumentations version: 2.0.4

Utilisez pip install pour télécharger toutes les bibliothèques nécessaires. Notez que, comme les serveurs grace et ada utilisent une architecture ARM, il est préférable d'utiliser la commande suivante pour télécharger les bibliothèques liées à torch :

pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu126

Cela permet de s'assurer que les versions trouvées sont compatibles avec l'accès GPU, car les versions automatiques du système ne garantissent pas toujours ce support.

---

4. Modification des chemins du code

4.1 Modification des chemins des données

Étant donné que le nom et le chemin du dossier créé diffèrent d'une personne à l'autre, vérifiez soigneusement **chaque chemin** dans le code avant de le lancer. Assurez-vous que chaque chemin est correct et que tous les fichiers que vous sauvegardez ou créez se trouvent à l'endroit approprié.

---

5. Augmentation des données

5.1 Introduction

La fonctionnalité d'augmentation des données représente une amélioration importante apportée au code U-net original dans le cadre de ce stage. Le code initial ne comportait pas cette fonctionnalité. Voici ma démarche :

À l'origine, les images IRM comprenaient 40 groupes, chacun contenant un nombre variable de coupes, pour un total de 3488 coupes. J'ai copié ces images d'origine dans une nouvelle variable, puis appliqué une transformation (par exemple, une rotation). Ensuite, j'ai recopié les images transformées dans une nouvelle variable. De cette façon, j'ai obtenu 40 groupes d'images originales et 40 groupes transformés, soit 80 groupes au total. Ces 80 groupes d'images constituent les données augmentées.

Pour la validation croisée (Kfold), les données ont été réparties en ensembles d'entraînement et de test, divisés en cinq folds. Chaque fold comprend un ensemble d'entraînement et un ensemble de test, avec une distribution aléatoire des données. L'ensemble d'entraînement représente 4/5 des données, et l'ensemble de test le cinquième restant.

Lors de l'implémentation de l'augmentation des données, j'ai utilisé les 80 groupes augmentés comme ensemble d'entraînement, tandis que les 40 groupes d'origine (non augmentés) ont servi d'ensemble de test. Cette approche a montré de bons résultats.

5.2 Sélection de la méthode d'augmentation

Dans la section du code concernant l'augmentation des données, après selected_aug =, choisissez n'importe quel mot-clé d'augmentation. Tous les mots-clés sont définis dans la variable augmentations du code, par exemple :

- rotate pour effectuer une rotation (par défaut 15 degrés)
- flip pour effectuer un retournement (flip).

Veuillez noter que, faute de temps, la fonction d'augmentation des données a uniquement été implémentée dans le fichier U-net_Cross_validation_bce-dice_pytorch.ipynb. Cela signifie qu'elle n'est pour l'instant disponible que dans la version PyTorch.

---

6. Exécution de l'entraînement

Dans le code U-net sous VSCode, cliquez sur "Run All" pour exécuter ligne par ligne.

Bonne chance !

---

7. Suivi et recommandations après la passation

- **Utilisation des outils de surveillance GPU pendant l'entraînement** :
    - Utiliser *nvidia-smi* pour surveiller les ressources GPU :
    - Utiliser *watch -c nvidia-smi* pour un suivi en temps réel de l'utilisation GPU