Rapport de Travaux Pratiques 4

Vision Avancée, Segmentation, et Données 3D

Joseph BOUIYODA
Département Génie Informatique, ENSPY
bouiyodajoseph@gmail.com
Lien vers le dépôt GitHub (Branche TP4)

22 janvier 2026

Table des matières

1 Introduction

Ce quatrième rapport de Travaux Pratiques aborde des tâches de vision par ordinateur avancées, en se concentrant sur la **segmentation sémantique** d'images médicales à l'aide de l'architecture **U-Net**. Ce TP introduit également des métriques de performance spécifiques à la segmentation, telles que le Coefficient de Dice et l'IoU. Enfin, il explore les concepts théoriques et les défis pratiques liés à la manipulation de données volumétriques 3D, tout en renforçant les bonnes pratiques d'ingénierie MLOps pour le suivi d'expériences.

Première partie
Segmentation et Bonnes Pratiques MLOps

2 Concepts Théoriques

2.1 Segmentation Sémantique et Architecture U-Net

Type de Sortie : Contrairement à la classification, qui produit un vecteur de probabilités pour une image entière, la sortie d'un modèle de segmentation sémantique est une **carte de segmentation** (ou masque). Ce tenseur a la même hauteur et largeur que l'image d'entrée (ex : `128x128`). Pour une segmentation binaire, sa profondeur est de 1, où chaque pixel contient une probabilité (entre 0 et 1) d'appartenir à la classe d'intérêt.

Structure U-Net : L'architecture U-Net se compose d'un chemin encodeur (qui contracte l'image pour extraire des caractéristiques sémantiques) et d'un chemin **décodeur**. Le rôle de ce dernier est de sur-échantillonner progressivement la représentation des caractéristiques pour reconstruire une carte de segmentation à la pleine résolution de l'image. Il permet de passer de l'information "quoi" à l'information "où". Les **skip connections** du U-Net, qui effectuent une **concaténation** (et non une addition comme dans ResNet), sont cruciales : elles ramènent les informations spatiales fines des premières couches de l'encodeur vers le décodeur, permettant une localisation et une reconstruction très précises des contours.

Fonctions de Perte : La *categorical cross-entropy* standard est souvent inadaptée pour la segmentation médicale en raison du **déséquilibre de classes extrême** (ex : une petite tumeur dans une grande image de tissu sain). Un modèle pourrait atteindre une haute précision en prédisant uniquement l'arrière-plan. Une alternative robuste est la **perte de Dice (Dice Loss)**, qui est dérivée du Coefficient de Dice et mesure directement le chevauchement entre la prédiction et la vérité terrain, la rendant moins sensible à ce déséquilibre.

Deuxième partie
Implémentations Pratiques

3 Exercice 1 & 2 : Implémentation de l'U-Net et des Métriques

3.1 Objectif

Implémenter de A à Z une architecture U-Net simplifiée, ainsi que les métriques de segmentation personnalisées (Dice, IoU) et une fonction de perte adaptée. Pour rendre l'exercice réalisable, un jeu de données a été simulé, consistant à segmenter des cercles sur un fond bruité.

3.2 Implémentation

Le script `unet_segmentation.py` a été développé. Il inclut la génération des données, la définition des métriques et de la perte, et la construction du modèle U-Net.

3.2.1 Définition des Métriques Personnalisées

Les fonctions pour le Coefficient de Dice et l'IoU ont été implémentées en utilisant le backend de Keras pour être utilisables durant l'entraînement. La perte de Dice a été définie comme `1 - dice_coeff`.

```
def dice_coeff(y_true, y_pred, smooth=1e-6):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    dice = (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) +
    smooth)
    return dice
```

Listing 1 – Implémentation du Coefficient de Dice

3.2.2 Architecture U-Net

L'architecture a été construite en respectant la structure encodeur-décodeur avec des skip connections via des couches de concaténation, comme montré dans l'extrait ci-dessous pour la première étape du décodeur.

```
#  tape  1 : Sur-  chantillonnage  + Skip Connection avec c3
u1 = keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(b)
u1 = keras.layers.Concatenate()([u1, c3]) # Concat nation
d1 = conv_block(u1, 64)
```

Listing 2 – Exemple de skip connection dans le décodeur

3.3 Résultats et Analyse

Le modèle (version allégée pour éviter la surchauffe matérielle) a été entraîné sur 15 époques. Les résultats obtenus sont excellents, démontrant la puissance de l'architecture U-Net même avec moins de paramètres. À la dernière époque, les métriques sur l'ensemble de validation étaient :

— **Coefficient de Dice (`val_dice_coeff`) :** 0.9937 (soit 99.37%)

— **IoU (`val_iou_metric`) :** 0.9875 (soit 98.75%)

Ces scores, très proches de 1.0, indiquent une superposition quasi-parfaite entre les masques prédits et les masques réels. Le modèle a parfaitement appris à segmenter les cercles. De plus, la performance en validation est supérieure à celle en entraînement, ce qui indique une excellente généralisation et une absence totale de surapprentissage, probablement due à l'effet régularisateur de la Batch Normalization et du Dropout. La visualisation des résultats (voir Figure **??**) confirme cette performance.

3.4 Analyse des Métriques IoU vs. Dice

Question : Comparez la sensibilité de la métrique IoU par rapport au Coefficient de Dice.

Réponse : Le Coefficient de Dice et l'IoU (ou Jaccard) sont deux métriques très similaires qui mesurent le chevauchement. Cependant, leur formulation mathématique leur donne des sensibilités différentes.

— Formule IoU : $\text{IoU} = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP+FP+FN}$

— Formule Dice : $\text{Dice} = \frac{2 \cdot |A \cap B|}{|A| + |B|} = \frac{2 \cdot TP}{2 \cdot TP+FP+FN}$

Le **Coefficient de Dice** est généralement considéré comme **moins sensible aux erreurs sur les petits objets**. En donnant un double poids aux Vrais Positifs (TP), il pénalise moins sévèrement les Faux Positifs (FP) et Faux Négatifs (FN) que l'IoU. Pour une petite tumeur par exemple, manquer quelques pixels (FN) ou en prédire quelques-uns en trop (FP) aura un impact plus faible sur le score de Dice que sur le score de l'IoU. C'est pourquoi le Dice (et la perte de Dice) est très populaire en imagerie médicale.

Troisième partie
Introduction aux Convolutions 3D

4 Concepts Théoriques

4.1 Opération Conv3D

— **Différence de dimension :** Un noyau Conv2D est une matrice 2D (ex : 3x3), tandis qu'un noyau **Conv3D** est un cube 3D (ex : 3x3x3). Un Conv2D se déplace sur les axes (hauteur, largeur), alors qu'un Conv3D se déplace sur les axes (profondeur, hauteur, largeur).

— **Nécessité :** Pour les données volumétriques (une pile de coupes 2D comme une IRM), un noyau Conv3D est nécessaire pour analyser un voisinage 3D de voxels et ainsi apprendre des caractéristiques contextuelles entre les coupes, ce qu'un Conv2D ne peut pas faire car il traite chaque coupe indépendamment.

4.2 Défi de la Mémoire

Les couches Conv3D sont extrêmement gourmandes en calculs et en mémoire. Les compromis d'ingénierie nécessaires incluent l'utilisation de petits noyaux (3x3x3), la limitation du nombre de filtres, le traitement de sous-volumes ("patches") plutôt que du volume complet, et un sous-échantillonnage agressif pour réduire rapidement les dimensions.

5 Exercice 3 : Bloc Conv3D et Suivi MLOps

5.1 Objectif

Implémenter un bloc Conv3D simple pour en comprendre la structure et appliquer des bonnes pratiques de suivi d'expérience avec MLflow, même sans entraînement réel.

5.2 Implémentation et Pratiques MLOps

Un script `conv3d_analysis.py` a été créé pour construire un petit modèle 3D et enregistrer ses propriétés dans une expérience MLflow nommée `"3D_Volumetric_Analysis"`. Les pratiques suivantes ont été appliquées :

1. **Enregistrement de l'architecture :** La structure complète du modèle a été sauvegardée en format JSON dans les artefacts de l'exécution MLflow, garantissant une reproductibilité totale.
2. **Enregistrement des hyperparamètres :** Les paramètres clés (optimiseur, nombre de filtres, forme d'entrée) ont été enregistrés avec `mlflow.log_param`.
3. **Simulation et enregistrement de métriques :** Un entraînement a été simulé en enregistrant des valeurs de perte et de métrique factices avec `mlflow.log_metric`.

Cette démarche démontre une discipline d'ingénierie rigoureuse, essentielle pour gérer des expériences complexes. Les résultats ont été vérifiés avec succès dans l'interface de MLflow.

6 Conclusion Générale

Ce TP a permis d'aborder avec succès la tâche de segmentation sémantique via l'implémentation complète d'une architecture U-Net. L'utilisation de données simulées a permis de valider son fonctionnement et d'obtenir d'excellentes performances, tout en mettant en pratique des métriques et fonctions de perte spécifiques à la segmentation. La partie conceptuelle et l'exercice sur les convolutions 3D ont fourni une introduction solide aux défis des données volumétriques et à l'importance des bonnes pratiques MLOps pour le suivi d'expériences complexes. Ce TP constitue une étape clé vers la maîtrise des applications avancées de la vision par ordinateur.