Université de Sherbrooke

IFT780 - Réseaux neuronaux

TP4

Réalisé par
Charles Boilard – boic0601
Hamza Boussairi – bouh2028
Tom Sartori – sart0701
Yanis Perrin – pery2002

15 avril 2024

Sommaire

1. Architectures

1.1. Liste des fichiers modifiés

Fichiers modifiés

- src/models/yourUNet.py : ajout du modèle YourUNet
- src/models/yourSegNet.py : ajout du modèle YourSegNet
- src/models/CNNBlocks.py : ajout de la dilatation dans DenseBlock
- src/manage/CNNTrainTestManager.py : ajout du checkpoint
- src/train.py : ajout des arguments et de l'augmentation de donnée
- src/train.ipynb : ajout d'arguments (save_checkpoint et load_checkpoint)

Fichiers ajoutés

- src/yourunet.ipynb
- src/yoursegnet.ipynb
- src/loss.ipynb
- src/loss/AsymLoss.py
- src/loss/DiceCELoss.py
- src/loss/DiceLoss.py

1.2. Illustration des architectures

Notre architecture YourUNet présente trois modifications majeures par rapport à l'architecture initiale :

- Ajout de couches denses
- Ajout de couches résiduelles
- Ajout de couches bottleneck

Elle garde cependant les caractéristiques d'une architecture symétrique avec des maxpooling, upsampling et skip-connection propres à l'architecture du UNet de base.

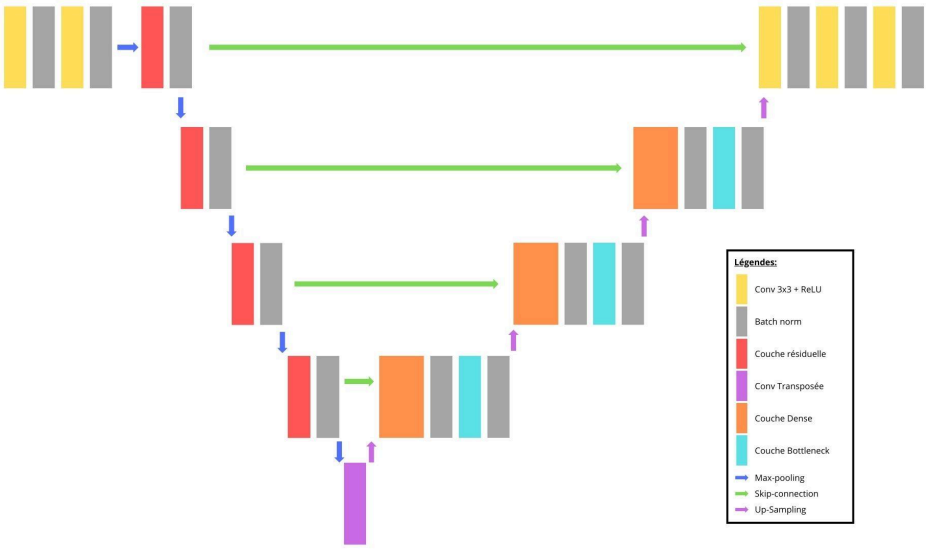


Figure 1 : Illustration de l'architecture de YourUNet

YourSegNet est inspiré du FullNet, à l'inverse de UNet on vient ici utiliser une architecture non symétrique composée de couches dilatées.

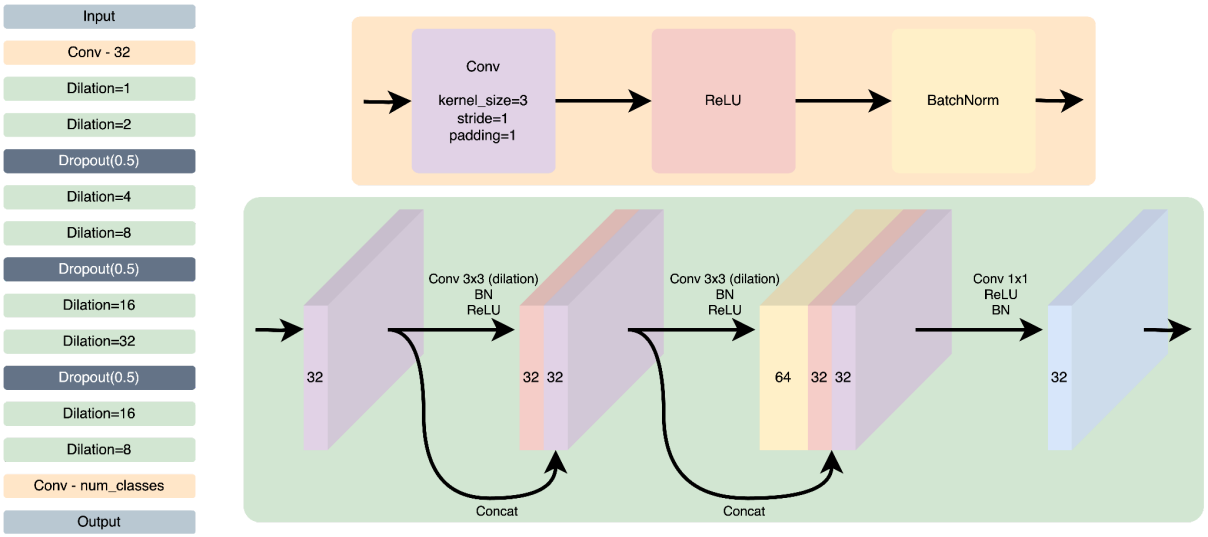


Figure 2 : Illustration de l'architecture de YourSegNet

1.3. Description de la nouvelle loss

Il est possible d'utiliser une des nouvelles loss à l'aide du paramètre '-loss', avec une des valeurs suivantes : 'CE', 'Dice', DiceCE' ou 'Asym'.

1.3.1 Dice Loss

$$DiceLoss = 1 - \frac{2TP}{2TP+FN+FP} = 1 - \frac{2|X \cap Y|}{|X| + |Y|} = 1 - DiceScore$$

Cette loss est souvent utilisée pour les problèmes de segmentation d'images, c'est pourquoi nous l'avons utilisée. Elle améliore grandement la justesse de nos résultats, dès les premières époques.

L'idée vient de ce post : Loss Function Library - Keras & PyTorch.

1.3.2 Dice + Cross Entropy Loss

$$Loss = Cross\ Entropy\ Loss + Dice\ Loss$$

La combinaison de ces deux losses permet de combiner les avantages de chacune. On obtient des résultats meilleurs que la Cross Entropy ou Dice Loss seules.

L'idée vient de ce post : Loss Function Library - Keras & PyTorch.

Asym Loss

$$AsymLoss = \frac{(|X \cap Y|)}{(|X \cap Y|) + \frac{(\beta^2)}{(1+\beta^2)}(Y \backslash X) + \frac{1}{(1+\beta^2)}(X \backslash Y)}$$

Cette loss est une version modifiée de la Dice Loss. L'article indique qu'elle est adaptée pour la segmentation d'images médicales et aussi, elle a souvent la meilleure justesse parmi les autres losses classées dans le GitHub SegLossOdyssey. Cependant, nous obtenons de très mauvais résultats.

L'idée et le code proviennent de ce GitHub : SegLossOdyssey.

1.4. Courbes d'entraînement avec une courte description

Les résultats du YourUNet sont présentés à la figure 3. La justesse de notre modèle atteint 76% sur 35 epochs et atteint 82,6% pour 40 epochs (poids de notre meilleurs modèles joint à ce rapport) sur l'ensemble de validation avec la loss Dice + CE. On remarque plusieurs pics étranges qui apparaissent aux itérations 14, 15 et 22, ces pics sont apparus au moment d'inactivité (ou pc mis en veille), ce qui pourrait expliquer leur existence. Une autre explication résiderait dans le fait que les probabilités des bonnes classes en sortie du modèle devraient être très petites, ce qui résulterait en une si "grande" loss. Cependant, nos courbes sont dans l'ensemble représentatifs d'un bon modèle dans le sens où les loss d'apprentissage et de validation décroissent au fil de l'entraînement alors que la justesse d'entraînement et de validation augmente au fil de l'entraînement.

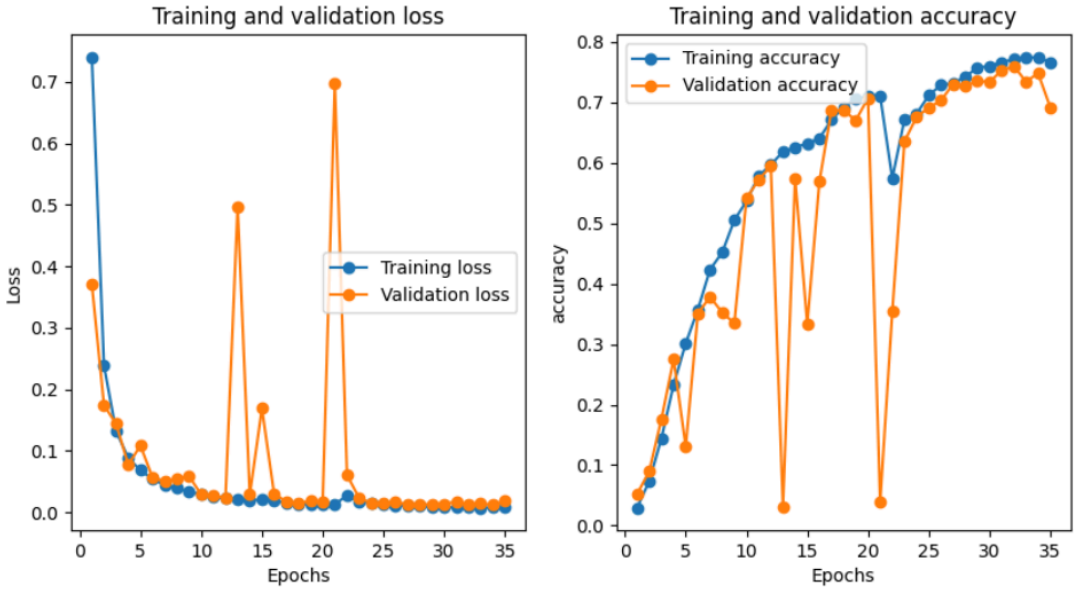


Figure 3 : Résultats de loss et justesse de YourUNet

Les résultats du YourSegNet sont présentés à la figure 4. La justesse atteint 86 % sur l'ensemble de validation et 82 % sur l'ensemble de test avec la loss Dice + CE. Un pic étrange apparaît aussi à l'itération 29, qui n'apparaît pas dans tous nos autres tests. Nous soupçonnons un problème de Colab, lors d'un captcha d'inactivité. Les mêmes observations que pour le modèle YourUNet sont effectuées dans le sens où le modèle présente de bonnes courbes d'évolution de loss et de justesse au fil de l'entraînement.

Figure 4 : Résultats de loss et justesse de YourSegNet

1.5. Lignes de commande pour exécuter votre code

YourUnet

Le notebook src/yourunet.ipynb permet de tester ce réseau. Sinon, il est possible d'exécuter les commandes suivantes :

Pour le lancer à partir de l'epoch 0:

```
python3 train.py --model=yourUNet --num-epoch=40 --batch_size=4 --lr=0.01 --loss="DiceCE" --save_checkpoint
```

Pour le lancer à partir de notre meilleur checkpoint:

```
python3 train.py --model=yourUNet --num-epoch=40 --batch_size=4 --lr=0.01 --loss="DiceCE" --save_checkpoint --load_checkpoint=checkpoints/YourUNet_DiceCE_82.pt
```

YourSegNet

Le notebook src/yoursegnet.ipynb permet de tester ce réseau. Sinon, il est possible d'exécuter les commandes suivantes :

Pour le lancer à partir de l'epoch 0:

```
python3 train.py --model=yourSegNet --num-epoch=40 --batch_size=4 --lr=0.01 --loss=DiceCE --save_checkpoint
```

Pour le lancer à partir de notre meilleur checkpoint:

```
python3 train.py --model=yourSegNet --num-epoch=40 --batch_size=4 --lr=0.01 --loss=DiceCE --save_checkpoint --load_checkpoint=checkpoints/YourSegNet_82.pt
```

2. Checkpointing

2.1. Liste des fichiers modifiés

Fichiers modifiés

- src/manage/CNNTrainTestManager.py : ajout de _save_checkpoint() et load_checkpoint()
- src/models/CNNBaseModel.py : mise à jour de save() et load_weights() pour récupérer les données des checkpoints
- src/train.py : ajout d'argument (save_checkpoint et load_checkpoint)

Fichiers ajoutés

- src/checkpoint.ipynb
- src/checkpoints/.keep

2.2. Lignes de commande pour exécuter votre code

Les checkpoints des modèles sont enregistrés dans le répertoire src/checkpoints/. Un notebook est également disponible dans src/checkpoint.ipynb.

Si vous souhaitez activer l'enregistrement du checkpoint, vous pouvez ajouter l'argument --save_checkpoint. Par défaut, il est désactivé.

```
python train.py --model=UNet --num-epochs=2 --save_checkpoint
```

Si vous souhaitez charger un checkpoint, vous pouvez ajouter l'argument --load_checkpoint avec le chemin du fichier de checkpoint (ex: --load_checkpoint=checkpoints/UNet.pt).

```
!python3 train.py --model=UNet --num-epochs=4 --save_checkpoint
--load_checkpoint=checkpoints/UNet.pt
```

3. Augmentation des données

3.1. Liste des fichiers modifiés

Fichiers modifiés

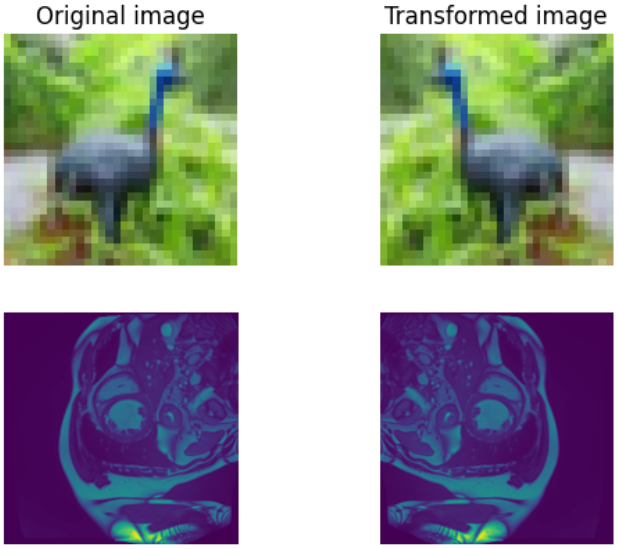
- src/train.py

Fichiers ajoutés

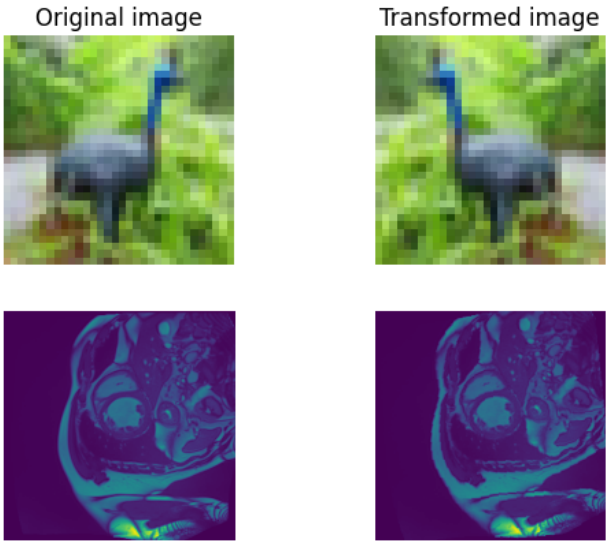
- src/data_augmentation.ipynb

3.2. Illustration de l'effet de l'augmentation de données sur quelques données

Première transformation avec Random Horizontal Flip (1ère image) et Random Perspective (2nd image) :



Deuxième transformation avec Random Horizontal Flip et Random Rotation (angle de 15 degrés) :



3.3. Description du type d’augmentation de données utilisé

Dans le tp nous avons privilégié les transformations géométriques aux transformations photométriques. Voici les trois types d’augmentation de données utilisées :

- Random Horizontal Flip
- Random Rotation
- Random perspective

Random Horizontal Flip est une transformation utilisée pour **retourner horizontalement** l'image donnée selon un angle aléatoire avec une probabilité donnée.
Random Rotation fait **pivoter** l'image selon l'angle spécifié.
Random perspective effectue une **transformation de perspective** aléatoire de l'image donnée avec une probabilité donnée.

3.4. Lignes de commande pour exécuter votre code.

Un notebook est disponible dans src/data_augmentation.ipynb.

Il est possible d’activer l’augmentation des données avec --data_aug et en utilisant les valeurs 0, 1 ou 2 pour choisir la transformation.

```
!python3 train.py --model=UNet --num-epoch=2 --data_aug=0
```