Semantic Segmentation with Deep Learning

Saâd Aziz Alaoui, Yassine Jamoud, Samy Haffoudhi

5 mars 2022

Introduction

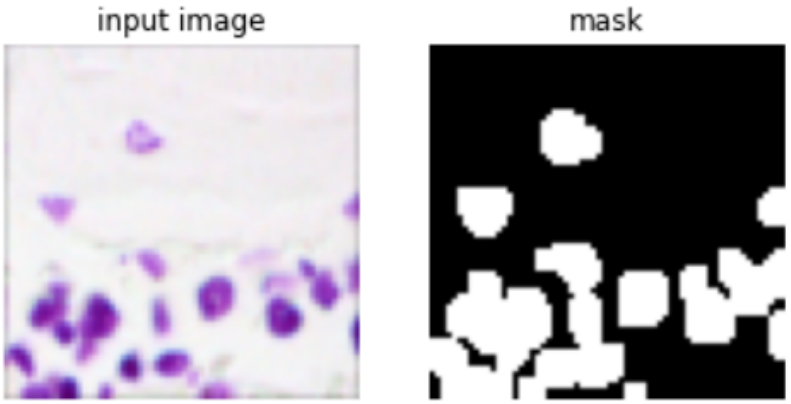
Le TP suivant porte sur de l'utilisation de techniques de Deep Learning pour la segmentation sémantique. Cette dernière consiste à étiqueter chaque pixel d'une image avec une classe correspondante à ce qui est représenté. L'idée derrière l'utilisation de Deep Learning est que les outils automatiques permettent de gagner énormément de temps et d'argent pour les diagnostics biomédicaux, ces outils deviennent de plus en plus cruciaux car les machines peuvent épauler les analyses effectuées par les radiologues, afin de réduire le temps nécessaire pour établir des diagnostics.

Nous nous intéressons ici à l'architecture **U-Net** qui permet justement d'effectuer des taches de segmentation sémantique. Le U-net est un réseau de neurones à convolution entièrement convolutionnel. L'idée principale derrière cette architecture est de remplacer les opérations de pooling par des opérateurs de suréchantillonnage ce qui implique l'augmentation de de la résolution de la sortie. Nous verrons lors de ce TP les différents atouts et défauts que comporte cette architecture en l'appliquant à des images biomédicales.

1 Les données

Nous disposons pour ce TP de différentes images biomédicales et plus précisément, des images de cellules. Ces données sont regroupées entre un dossier de test et un dossier d'entrainement. On dispose, pour les images d'entrainement, de plusieurs images qui, un fois combinées, correspondent au masque.

FIGURE 1 – Un exemple d'image et du masque associé



Nous disposons de 50 images pour l'entrainement. On les sous-échantillonne toutes au mêmes dimensions et on conserve uniquement les 3 premiers canaux. De même pour les labels mais on dispose que d'un unique canal.

2 Le modèle

2.1 L'architecture

Le modèle u-net est composé d'une couche d'entrée, un encodeur et un décodeur. L'encodeur et le décodeur ont une structure en blocs similaires mais des dimensions différentes. Chaque bloc de l'encodeur est composé de deux couches de convolution de mêmes dimensions, d'une couche de pooling et d'une fonction d'activation. Pour compenser la baisse de la dimension de l'image,

le nombre de filtres augmente. Enfin, le modèle dispose également d'une liste de connexions entre l'encodeur et le décodeur.

Une visualisation de l'architecture est disponible en annexe A. On observe bien la forme en U.

2.2 Les fonctions de coût

Le coefficient de Dice vaut $s = \frac{2|X \cap Y|}{|X|+|Y|}$. Cet indice permet de mesurer la similarité entre deux ensembles X et Y. IL varie de 0 quand X et Y sont disjoints à 1 quand X et Y sont égaux.

La fonction de coût Dice est alors définie par $L(y_{pred}, y_{true}) = 1 - s(y_{pred}, y_{true})$.

3 Entrainement et test

3.1 Entrainement

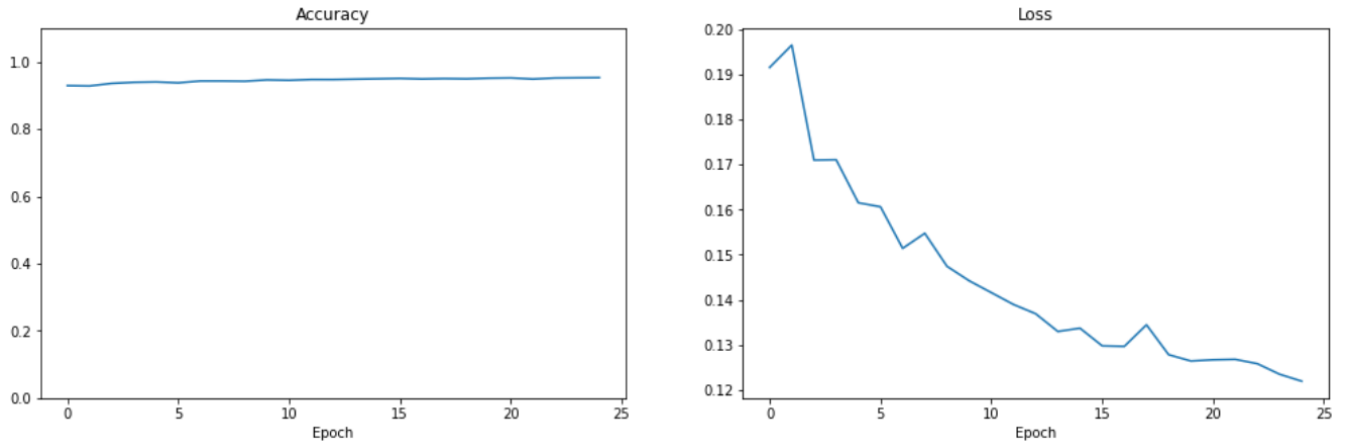
Pour entrainer le modèle on utilise 25 epochs et un batch size de 25.

On remarque naturellement l'impact des dimensions des images. Plus ces dernières sont grandes, plus l'entrainement est long. Par exemple, pour notre machine on obtient comme temps d'éxécution :

— 0.6s par step pour des images de taille (64, 64)
— 8s par step pour des images de taille (256, 256)

On obtient les tracés suivants :

FIGURE 2 – Accuracy & Loss



On observe que la précision converge vers 1 et que la fonction de coût décroit convenablement. On peut faire varier les paramètres `epochs` et `batch_size`. Augmenter `epochs`, nous permet d'obtenir de meilleurs résultats jusqu'à un certain seuil d'overfitting qui baissera l'efficacité du modèle. En ce qui concerne `batch_size`, il s'agit du paramètre définissant le nombre de samples propagés dans le réseau, on retrouve les défauts et avantages principaux de choisir `batch_size` inférieur au nombre total de samples :

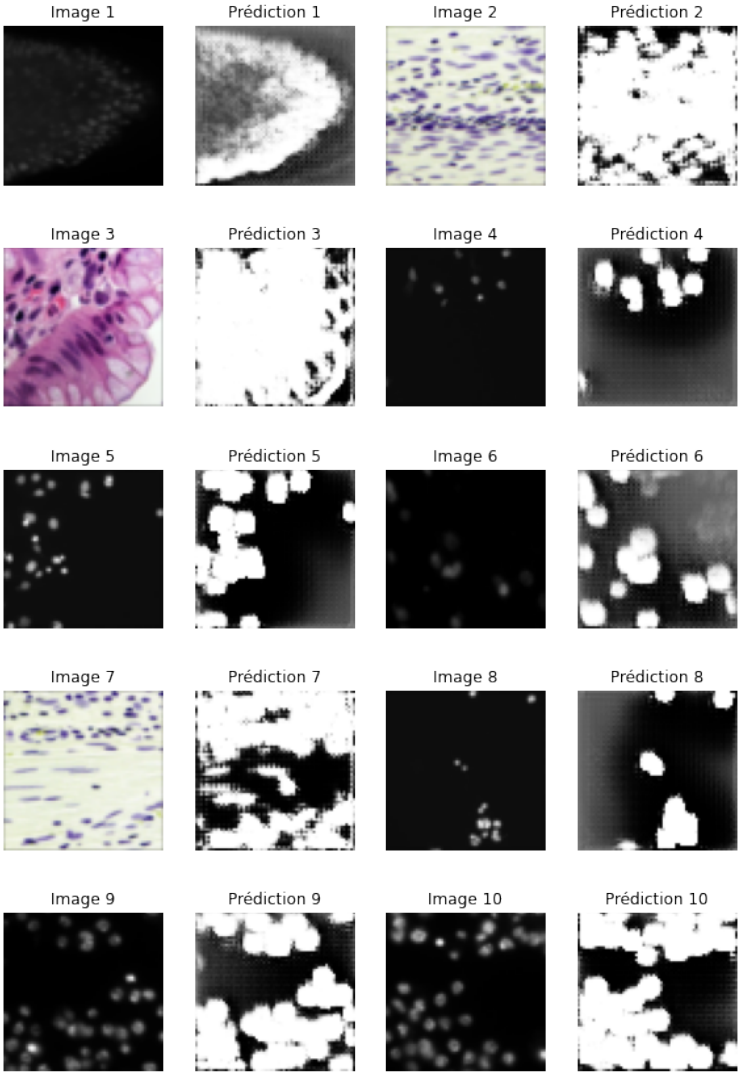
— Avantage : Ça demande moins de mémoire et l'entrainement est plus rapide.
— Inconvénient : Plus la valeur de `batch_size` est faible, plus l'estimation du gradient est mauvaise.

3.2 Test

On dispose de 10 images pour tester notre modèle.

On obtient les résultats suivants :

FIGURE 3 – Prédictions du modèle



On observe que ces résultats sont assez convaincants. En effet, comme précisé plus haut, nous disposons pas de masques pour l'ensemble de test donc, on ne peut faire que des comparaisons visuelles. Ici, les masques sont souvent corrects en comparaison avec les images. On aurait pu afficher les images en grayscale pour mieux comparer visuellement mais ça n'aurait pas eu un grand effet. Par ailleurs la précision tendant vers 1 pourrait indiquer que notre modèle est en overfitting.

4 Comparaison et améliorations

Lors de cette dernière partie on va essayer d'améliorer les performances du modèle précèdent à travers différentes méthodes.

4.1 Data Augmentation

Cette première piste consiste, comme son nom l'indique, en une augmentation de la quantité de données. Cette augmentation, artificielle, consiste en la modification des images à travers différentes transformations telles que des rotations ou des translations. Cette méthode permet de réduire l'overfitting puisqu'on fournit alors plus de données à notre modèle.

On choisit de réaliser cette augmentation en ligne. C'est-à-dire qu'au lieu de générer plus de données dès le chargement des données on le fait lors de l'entrainement du modèle. Cette approche présente deux avantages :

— On gagne en vitesse grâce à l'accélération GPU
— On conserve cette étape même après exportation du modèle

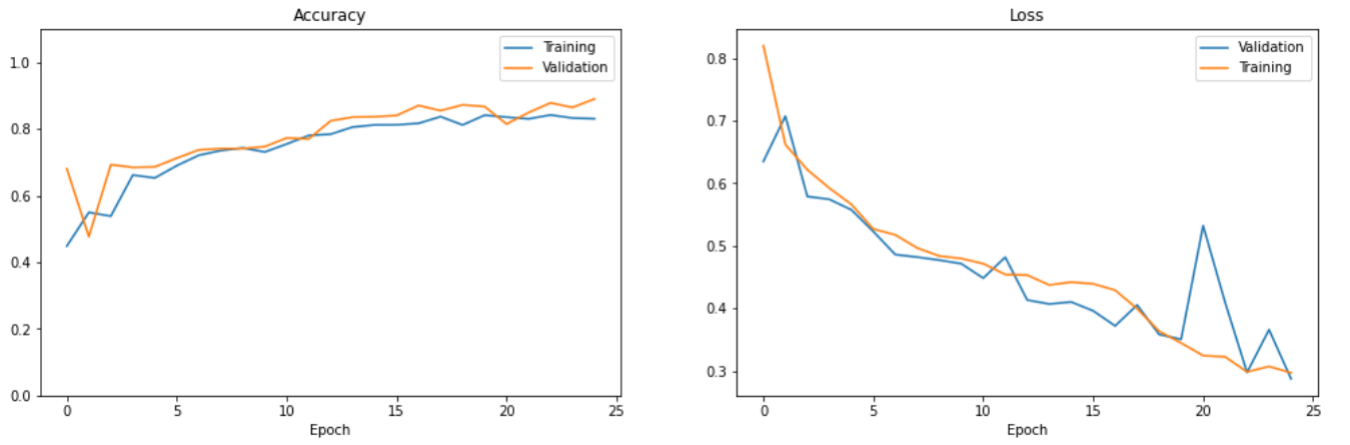
4.2 Validation Set

On opte également pour l'utilisation d'un ensemble de validation. Le modèle mettra à part des données et les utilisera à la fin de chaque epochs pour évaluer la fonction de coût et la précision. Cette étape nous permettra alors d'identifier des cas d'overfitting ou d'underfitting après analyse des courbes obtenues.

4.3 Résultats obtenus

Après application des deux techniques ci-dessus on obtient les tracés suivants :

FIGURE 4 – Accuracy & Loss



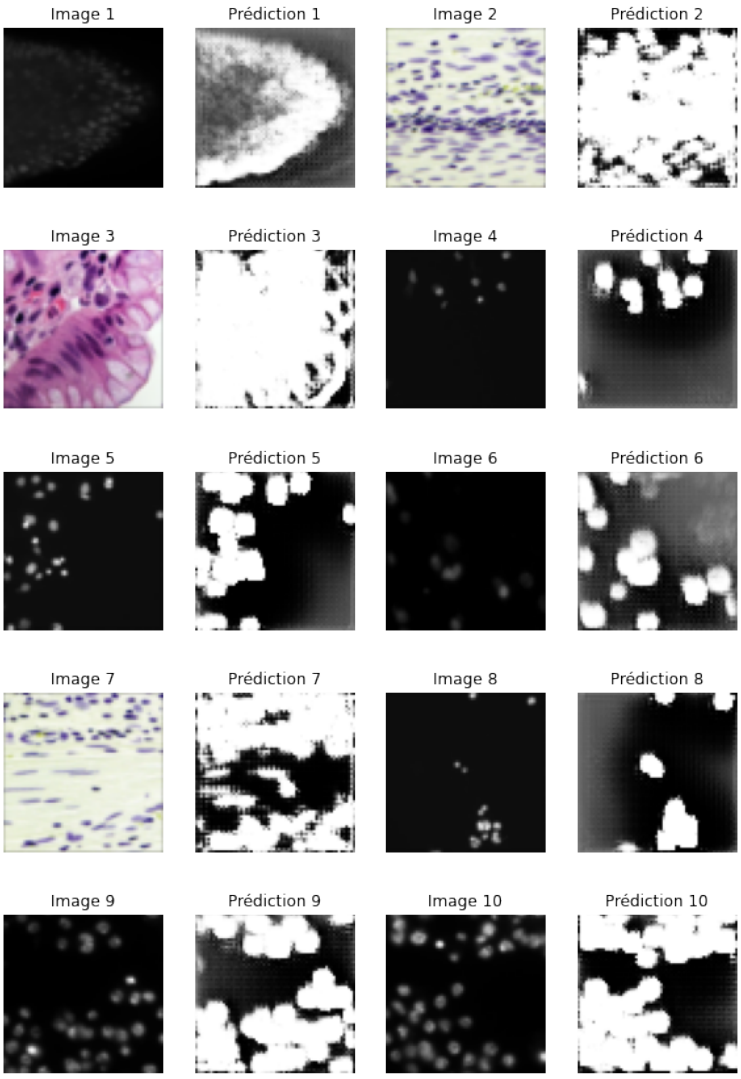
On observe que :

— Les courbes de précisions croissent et tendent aux alentours de 0.8
— Les courbes de coût décroissent

— Les performances entre l'ensemble de validation et d'entrainement sont très similaires, le modèle généralise bien

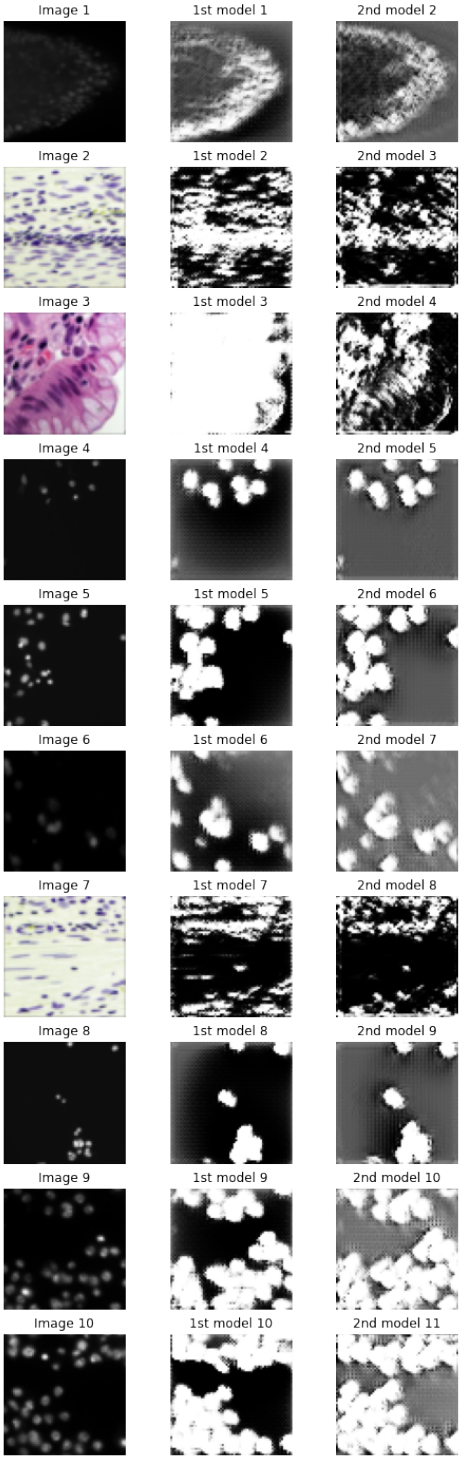
On obtient les prédictions suivantes :

FIGURE 5 – Prédictions du modèle



Affichons les avec les prédictions fournies par le premier modèle :

FIGURE 6 – Comparaison des prédictions



On observe alors que le deuxième modèle est légèrement plus performant que le premier. Pour l'image 3 par exemple, qui est assez complexe, on voit que le deuxième modèle fournit un bien meilleur masque. Cependant, on voit sur l'exemple de l'image 10 par exemple que c'est le modèle 2 qui grossit légèrement les traits. Disons que de manière globale, le modèle 2 est plus performant même si les fonds fournis par le premier sont généralement plus noirs. Notre amélioration de modèle est bien effective mais peut encore être améliorée.

Conclusion

En conclusion, nous avons pu nous entraîner sur l'architecture U-net de deep learning Nous avons pu voir que l'encodeur réduit les dimensions spatiales dans chaque couche et augmente les canaux. D'autre part, le décodeur augmente les dimensions spatiales tout en réduisant les canaux. Ces types de modèles sont extrêmement utilisés dans les applications du monde réel comme on a pu le voir. Notre amélioration du modèle nous prouve qu'il y'a encore beaucoup de travail à faire sur l'optimisation des paramètres de ces modèles et que leur capacité d'expansion semble un peu infinie.

A Architecture du modèle

FIGURE 7 – Architecture du modèle