Implémentation et Analyse d'une Architecture U-Net

Yann Trottier
École Polytechnique de Montréal
yann-2.trottier@polymtl.ca

Abstract

Dans ce projet, nous reconstruisons l'architecture très populaire *U-Net*, qui est utilisée pour faire de la segmentation d'image, en nous référant à l'article original [Ronneberger, 2015]. Nous expliquons ce en quoi consiste cette architecture ainsi que les travaux antérieurs et postérieurs. Nous expliquons ensuite notre implémentation et illustrons les résultats de nos expériences avec la base de données Pascal VOC 2012.

1 Introduction

L'une des tâches les plus importantes en vision robotique est de faire de la segmentation d'image, c'est-à-dire de pouvoir détecter la forme des objets même dans des images. Cela implique de pouvoir faire une classification pixel par pixel de l'image afin de reproduire un masque des formes détectées. Plusieurs architectures permettent de faire une telle classification, incluant le U-Net bien sûr, mais aussi les *Fully Convolutional Network* (FCN), les *Feature Pyramid Network* (FPN) et bien d'autres.

L'architecture U-Net a été décrite pour la première fois dans l'article *U-Net: Convolutional Networks for Biomedical Image Segmentation*, publié en 2015. Cette architecture se base sur le *Fully Convolutional Network* proposé par Long, Shelhamer, and Darrell [Long, 2014], et a été utilisée pour la première fois dans le domaine biomédical sur des images de cellules. En effet, il a fallu concevoir une nouvelle architecture parce que la quantité d'images de cellules était extrêmement petite comparée aux datasets de l'époque, tel que *ImageNet* comportant plus de 1 million d'images d'entraînement. De plus, on propose aussi de prendre pleinement avantage d'un très petit dataset en effectuant des augmentations, c'est-à-dire des déformations et des étirements, pour mieux entraîner le réseau.

1.1 Travaux antérieurs

Antérieurement au U-Net, l'état de l'art était un réseau à fenêtre glissante, proposé par Hence, Ciresan et al [Ciresan, 2012], ayant gagné la compétition du *EM segmentation challenge at ISBI* en 2012. Ce réseau permettait de classifier un pixel en provisionnant une petite région autour de ce pixel, et d'assigner une classe au pixel selon cette région. Cependant, ce type de réseau avait deux pénalités: il est difficile de trouver un compromis entre se fier au contexte local du pixel ou à la localisation, ce qui limitait la performance du réseau.

1.2 Travaux modernes

Bien entendu, U-Net n'est plus l'état de l'art, malgré qu'au moment de sa publication, l'architecture a grandement impactée le domaine de la vision robotique. Bien que l'architecture a été conçue pour des images de cellules dans le domaine biomédical, elle a été réutilisée dans tous les domaines vu sa grande efficacité. De nos jours, les méthodes modernes comportent par exemple les réseaux *Mask R-CNN* et *DeepLab*.

2 Rappel théorique du U-Net

Architecture du U-Net

L'architecture du U-Net décrite dans l'article original est illustrée à la figure suivante:

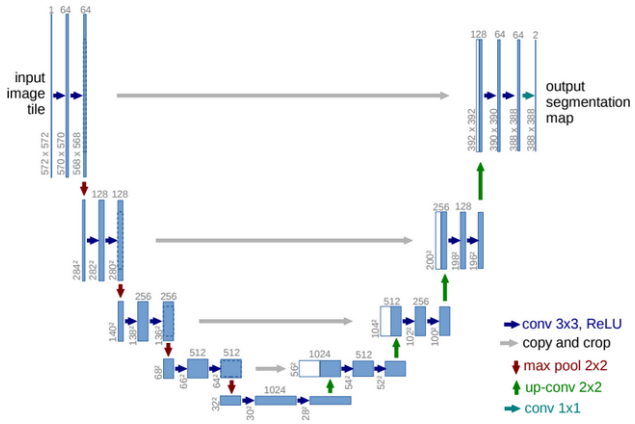

Figure 1: Architecture du U-Net

Tel qu'observé à la figure 1, l'architecture a une forme de "U", dont la provenance du nom. Le U-Net, tout comme le *Fully Convolutional Network*, est muni d'un encodeur et d'un décodeur. La phase d'encodage (ou "contractante") sert à extraire les *features* de l'image et de placer cette représentation dans l'espace latent: cela permet d'obtenir le contexte des pixels. Ensuite, dans la phase de décodage (ou "expansive"),

le réseau reconstruit l'image. Cependant, la particularité du U-Net est la présence de *skip connections*, ou de "sauts de connections" : la phase de décodage reprend l'information spatiale provenant de la phase d'encodage afin de reconstruire une image avec des contraintes spatiales beaucoup mieux définies.

Concrètement, l'architecture peut être décrite avec les mécanismes suivants.

Dans la phase d'encodage, on prend une image en entrée et on applique successivement les opérations suivantes: une série de deux convolutions 3x3, chacune suivie d'une unité ReLU, finalement suivie d'un *max pooling* 2x2. Cette séquence est répétée quatre fois dans la phase d'encodage. À chaque fois que cette séquence est appliquée, le nombre de filtres est multiplié par deux.

Dans la phase du goulot, on applique simplement une série de deux convolutions 3x3. Cette partie lie la partie encodeur et la partie décodeur.

Dans la phase de décodage, on applique successivement les opérations suivantes: un *up-convolution* 2x2, suivi d'une concaténation avec la sortie de la phase d'encodage du même niveau, suivi d'une série de deux convolutions 3x3, chacune suivie d'une unité ReLU. On répète cette séquence quatre fois. La *up-convolution* 2x2 permet de redimensionner l'image en l'agrandissant et peut se faire avec une convolution transposée.

Finalement, on applique une convolution 1x1 pour obtenir le nombre de classes désirées.

Entraînement

Dans l'article original, on propose d'utiliser une descente de gradient stochastique (SGD), d'utiliser une taille de *batch* de 1, et d'utiliser une fonction de perte d'entropie croisée.

Augmentation des données

Dans l'article original, on propose de ne faire que des déformations élastiques afin de laisser intacte les valeurs de décalage et de rotation.

3 Approche théorétique

Dans notre approche théorétique, l'idée originale du projet était d'implémenter l'architecture de base du U-Net avec Pytorch et Pytorch Lightning, puis de tenter d'améliorer la performance avec différentes techniques modernes. Par exemple, la technique de *batch normalization* n'est apparue que quelques mois avant la publication de l'article de U-Net et n'a donc pas pu être implémentée dans l'architecture originale. De plus, l'entropie croisée n'est pas la meilleure fonction de perte disponible: il existe la fonction de perte *dice loss* et aussi *hard pixel mining* qui peuvent être beaucoup plus performantes.

En termes de métriques, pour mesurer la performance du réseau, nous voulions implémenter la *mean IoU (Intersection over Union)* ainsi que le *Dice score*. La métrique IoU est aussi appelé coefficient de Jaccard : nous utilisons la librairie sklearn pour l'implémenter.

Cependant, nous avons manqué de temps, et seulement la *batch normalization* a pu être testé autre que la performance de base. Seule la métrique *mean IoU* a pu être implémentée. Les augmentations d'image non plus n'ont pas pu être testées. En effet, entraîner l'architecture prend plusieurs heures (plus de 4 heures) et la collecte de données se fait trop lentement. Aussi, nous avons utilisé Adam au lieu de SGD.

Pour le *dataset* utilisé, nous employons le fameux dataset Pascal VOC 2012. Chaque image dans ce dataset est de taille différente. Cependant, pour l'entraînement, il est nécessaire que chaque image soit de même taille: celles-ci, ainsi que leurs masques, sont redimensionnées sous taille 256x256. De plus, l'architecture originale s'attend à prendre en entrée des images en tons de gris, alors que les images Pascal VOC sont en tons RGB. Finalement, le dataset ne comporte pas deux classes (binaires), mais bien 21 classes (incluant la classe du fond).

Information	Valeur
Nombre d'échantillons de training	1464
Nombre d'échantillons de validation	1449
Nombre de classes	21
Type d'image	RGB

Table 1: Informations sur le dataset Pascal VOC 2012

L'architecture a conséquemment dû être modifiée pour prendre en compte ces changements. Concrètement, le nombre de canaux en entrée et en sortie à chaque couche du U-Net est décrit à la table 2 :

No. de couche	Canaux d'entrée	Canaux de sortie
0 (Entrée)	3	64
1 (Contractante)	64	128
2 (Contractante)	128	256
3 (Contractante)	256	512
4 (Contractante)	512	1024
5 (Goulot)	1024	1024
6 (Expansive)	1024	512
7 (Expansive)	512	256
8 (Expansive)	256	128
9 (Expansive)	128	64
10 (Sortie)	64	21

Table 2: Nombre de canaux en entrée et sortie par couche

4 Expériences et discussion

Lien vers le Github: https://github.com/ytro/unet-project

4.1 Visualisation du dataset

Chaque image du dataset est fourni avec son masque de segmentation. Dans l'image suivante, nous mettons en évidence deux images au hasard ainsi que leur masque associé (la visualisation ne se fait pas en RGB malheureusement):

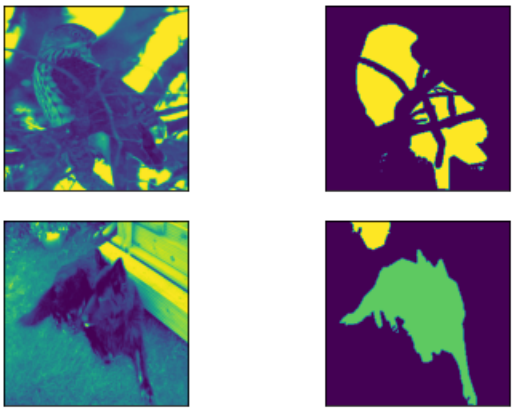

Figure 2: Deux images et leur masque fourni

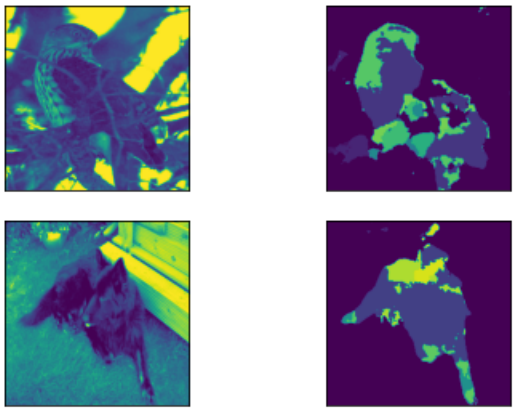

Figure 3: Deux images et leur masque prédit

4.2 Résultats

Par manque de temps, nous n'avons que pu rouler deux expériences (le temps d'entraînement étant très long).

Pour chaque expérience, nous avons utilisé les paramètres suivants:

- learning rate = 0.0005
- batch size = 4
- max epochs = 100

Dans le tableau suivant, nous mettons en évidence le résultat de nos deux expériences avec leur métrique *mean IoU*, mais aussi les scores publiques de d'autres types de réseaux plus performants sur le même dataset (Pascal VOC 2012) et sur le même ensemble de validation.

Type de réseau	Score (mIoU)
U-Net (base)	0.5848 (val set)
U-Net (+ Batch norm)	0.5968 (val set)
ResNet101-DeepLabv3	0.7721 (val set)
ResNet101-DeepLabv3+	0.7885 (val set)

Table 3: Type de réseau et le score mIoU associé

Exemples de prédictions

En réutilisant le modèle U-Net + BatchNorm entraîné avec les images de la section 4.1, nous obtenons les prédictions suivantes:

Courbes d'apprentissage, de perte et IoU

Finalement, pour l'expérience U-Net + BatchNorm, voici quelques figures sur les courbes d'apprentissage et de perte (train_acc, train_loss, val_acc, val_loss) ainsi que la courbe de la métrique IoU.

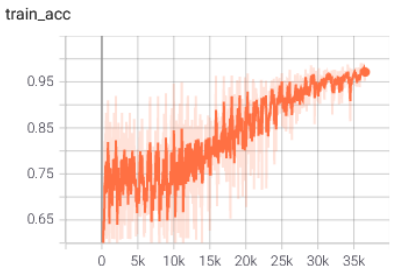

Figure 4: Courbe d'apprentissage (accuracy)

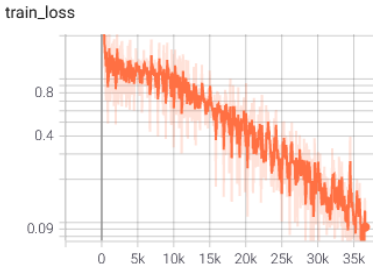

Figure 5: Courbe d'apprentissage (perte)

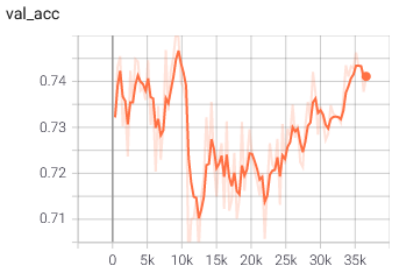

Figure 6: Courbe de validation (accuracy)

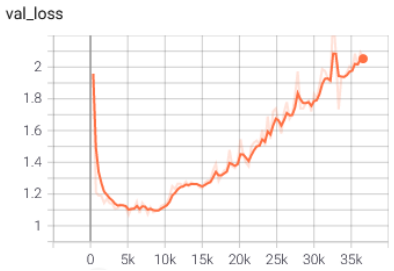

Figure 7: Courbe de validation (perte)

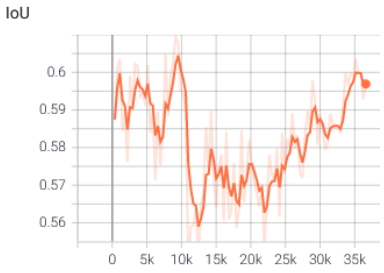

Figure 8: Courbe de métrique IoU

4.3 Discussion

Bien que nous n'avons fait que deux expériences, il est tout de même possible d'affirmer la performance du U-Net. Le résulat du *mean IoU* est tel qu'attendu.

Le U-Net, dans les deux expériences, apprend correctement. En effet, on peut observer à la figure 4 que l'accuracy augmente, et à la figure 5 que la perte décroît. Il est aussi possible d'affirmer que le modèle commence à *overfitter* à partir de 10k itérations, car la courbe de validation dans la figure 6 décroît et celle de 8 aussi, alors qu'elle augmente dans 4.

Il est certain que si nous avions eu le temps d'augmenter les images et de faire les autres améliorations sur le U-Net, nous aurions encore de meilleurs résultats, mais ceux-ci sont déjà satisfaisants.

5 Analyse critique de l'approche

Pour apprendre mon sujet, j'ai dû me renseigner beaucoup sur des sites web et sur YouTube pour bien comprendre le fonctionnement du U-Net. Mais aussi, il fallait comprendre où se situe le U-Net en lien avec les méthodes plus modernes (telles que DeepLab), et donc il y avait beaucoup de recherche à faire. Je crois que ce qui est important, la prochaine fois, serait de cibler les sources importantes d'informations.

References

[Ciresan, 2012] Gambardella L.M. Giusti A. Schmidhuber J. Ciresan, D.C. Deep neural networks segment neuronal membranes in electron microscopy images. *NIPS*, 2012.

[Long, 2014] Shelhamer E. Darrell T. Long, J. Fully convolutional networks for semantic segmentation. *arXiv:1411.4038*, 2014.

[Ronneberger, 2015] Fischer P. Brox T Ronneberger, O. U-net: Convolutional networks for biomedical image segmentation. *arXiv:1505.04597v1*, 2015.