Université LAVAL

Apprentissage et reconnaissance GIF-4101/GIF-7005

Projet : Détection automatique de prolongements neuronaux

Auteurs :
Baptiste Amato
Alexandre Chavenon
Arnoud Vanhuele

1 Introduction

Présentation du projet

Le projet est proposé par le centre de recherche CERVO. Il consiste à reconnaître des axones et des dendrites sur des images d'une protéine (actine), en étiquetant ces images n'ayant pas de marqueurs axonaux et dendritiques. Nous disposons d'une banque de données d'images déjà marquées : il s'agit donc un problème d'apprentissage supervisé.

Jeu de données

Le jeu de données initial comprend 1024 images au format *.tiff*, ayant chacune 3 canaux : un pour l'actine (la protéine d'intérêt), un pour les axones, et un pour les dendrites.
Ce jeu de données étant relativement petit pour un apprentissage par réseau neuronal, nous allons utiliser des méthodes d'augmentation comme les symétries, rotations, ou encore découpes de sous-parties des images.

Etat de l'art

Il s'agit ici de détecter différents objets dans une image (axones et dendrites à partir d'une image globale d'actine) : c'est un problème de détection particulier, car il n'est pas possible d'encadrer les objets par des "bounding boxes", utilisées par exemple pour la détection de visage, de personnes ou de voitures ; on cherche alors à détecter le contour des objets. Un article de recherche assez récent a démontré une capacité de détection de contour impressionnante : *Object Contour Detection with a Fully Convolutional Encoder-Decoder Network*, par **Yang *et al.***. Nous pensons donc nous orienter vers un réseau de neurones profond avec une architecture *Encoder-Decoder* ; cette architecture est aussi utilisé dans les traductions de textes (séquences en entrée et sortie). Des résultats probants concernant de la segmentation d'images sont présentés dans l'article *Iterative Deep Convolutional Encoder-Decoder Network for Medical Image Segmentation*, par **Jung Uk Kim, Hak Gu Kim, et Yong Man Ro**, suivant une architecture similaire (*Encoder-Decoder*).
Les principes de segmentation d'image sont clairement expliqués dans l'article **Fully Convolutional Networks for Semantic Segmentation**, par **Shelhamer *et al.***.
Concernant l'augmentation de notre jeu de données, nous aurons une approche classique par traitement d'image traditionnel et dans un second temps, nous explorerons la possibilité d'utiliser un Generative Adversarial Network pour l'augmentation des données, comme précisé dans l'article*Biomedical Data Augmentation Using Generative Adversarial Neural Networks. Artificial Neural Networks and Machine Learning* par **Calimeri, F. *et al.***.

2 Pré-traitements

Masques

Une image du jeu de données fournit contient trois canaux, pour l'actine, les axones et les dendrites. Nous séparons donc le *.tiff* afin d'obtenir trois images, avec l'actine en vert, les axones en rougeet les dendrites en bleu. Après traitements, on a des images comme suit :

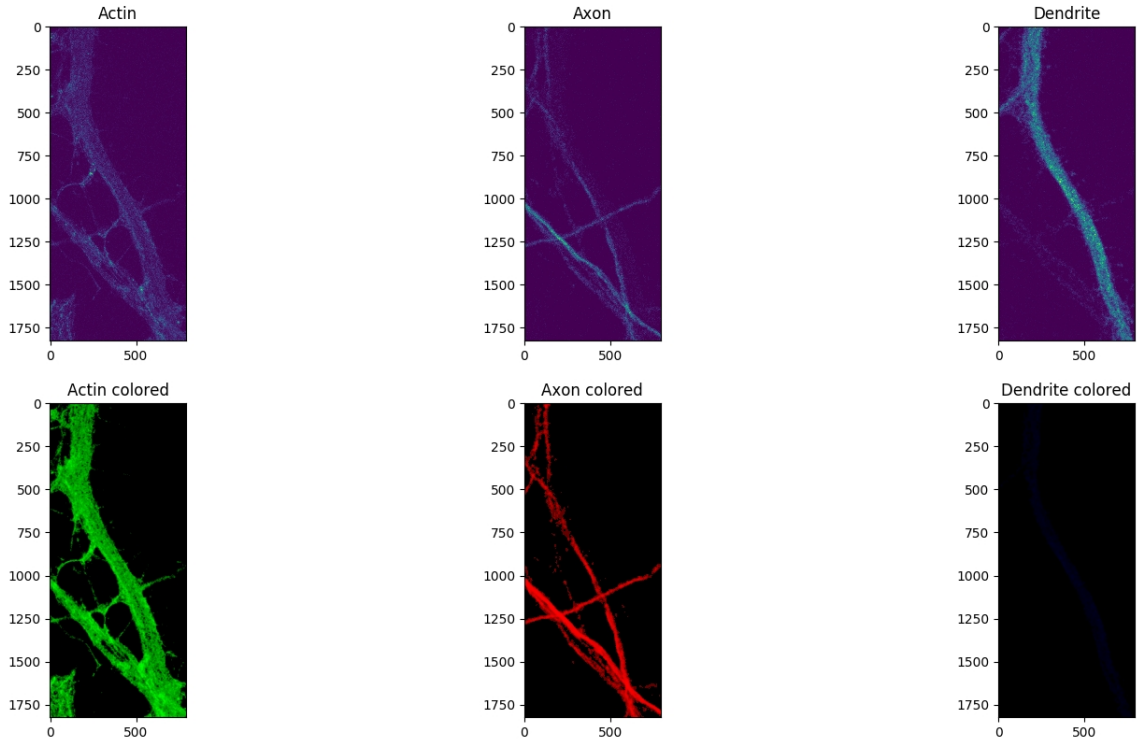

Figure 2.1: En haut, un triplet actine-axone-dendrite, et en bas les mêmes images une fois le masque appliqué

Le pré-traitement commence par une égalisation de l'histogramme des intensités des images en niveaux de gris. On utilise ensuite deux opérations, une d'érosion afin de ne garder que du contenu autour des contours, et une de flou gaussien, afin d'obtenir des contours légèrement plus denses.
On applique ensuite un *thresholding* sur chaque image, permettant de supprimer du bruit, soit les pixels de très faible intensité : on définit un seuil entre 0 et 255 (dans notre cas, 10), et tous les pixels ayant une valeur inférieure à 10 sont ramenés à 0. On convertit ensuite les images en RGB, et on ne conserve des valeurs non nulles que pour un canal par image (vert pour l'actine, rouge pour les axones et bleu pour les dendrites).

Images d'entrée

Taille

Les images en entrée sont toutes de dimensions différentes, mais de même résolution. Il est donc nécessaire de conserver ces résolutions (ayant un sens physique) en n'effectuant aucun redimmensionnement. Le problème est que le réseau de neurones attend des images de même taille en entrée : nous avons donc découpé chacune des images en *crops* (petits carrés) de taille 224x224, correspondant à la taille des images en entrée du réseau **VGG16**, mentionné plus bas.

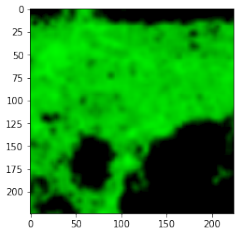

Figure 2.2: Exemple de *crop* d'une image d'entrée (actine seule)

Contenu

Notre problème est une détection de contour, mais nous devons inclure la notion d'**intensité** dans nos matrices d'entrée (valeur allant de 0 à 255). Les images sont normalisées entre 0 et 1 ; une matrice d'entraînement sera de taille 224x224x1 (intensités de l'actine), et le label associé sera une matrice de taille 224x224x1, selon si l'on souhaite entraîner le réseau sur les axones ou les dendrites. Nous avions commencé en traitant simultanément les axones et dendrites (donc en ayant des labels de taille 224x224x2), mais il semblait que l'un des channels est souvent sur-appris alors que l'autre était délaissé. La fonction de perte devait retourner une valeur unique pour la matrice, donc le réseau n'avait aucun moyen de savoir quel channel devait être corrigé.

Prédiction d'une nouvelle image

Là encore, il est nécessaire de procéder par *crops* pour traiter une nouvelle image. Ainsi, chaque *crop* sera associé à une image de prédiction, et l'image résultante sera reconstituée à partir de ces *crops*. Le réseau de neurones n'a donc affaires qu'à des images de taille 224x2242z.

Augmentation des données

Nous avons, pour chaque *crop*, ajouté des *flips* (opérations "miroir") permettant d'avoir en sortie 4 nouvelles données (l'originale, la symmétrie horizontale, verticale et l'enchaînement des deux). Ces opérations ne sont effectuées que lorsque l'image contient de l'information intéressante, c'est-à-dire si elle n'est pas trop noire ni trop colorée (en effet, dans ces cas, les opérations de symmétrie n'auraient que peu d'intérêt).

3 Réseau de neurones

Nous utilisons la librairie *Keras* avec un back-end en *TensorFlow*. L'architecture sera celle présentée dans l'article de recherche *Object Contour Detection with a Fully Convolutional Encoder-Decoder Network*, par **Yang *et al.*** : la première parte du réseau est l'encodeur, basé sur l'architecture du réseau **VGG16**, en s'arrêtant juste avant le *Fully Connected layer*. Ensuite, le décodeur est celui décrit par l'article, permettant de reconstituer une image de la taille d'origine avec des opérations de *déconvolution*, qui en *Keras* se font grâce à l'opération *UpSampling* suivie d'une *same convolution* (convolution ne modifiant pas la taille de l'image d'entrée grâce à l'ajout de *padding*).
L'optimisation est faite avec la méthode **Adam** utilisant les avantages des algorithmes **AdaGrad** et **RMSProp** ; celle-ci a fait ses preuves dans le monde du Deep Learning. + TODO: loss function

4 Tests

Le jeu de données, après transformations, est très lourd (plusieurs dizaines de Giga Octets). De plus, chaque image n'est pas fournie telle quelle au réseau de neurones, mais est découpée en *crops* qui, selon la taille de l'image, peuvent être au nombre de 30 voire 40 pour une unique image. Ainsi, nos ordinateurs ne pouvaient pas créer des tableaux *numpy* contenant toutes les données. De plus, lors des entraînements du réseau, nous avons été confrontés à des *ResourceExhaustedError*, donc nous devions utiliser des tailles de *mini-batches* très petites (4 au plus), ce qui ne donne pas de bons résultats.
Nous avons alors obtenu l'accès à une machine des laboratoires de l'université ayant un GPU puissant afin d'effectuer des tests sur un grand nombre de données et des tailles de *mini-batches* raisonnables (32). Il a été nécessaire de créer une instance *Docker* pour pouvoir empaqueter notre code dans la machine ; une fois cela fait nous avions accès à distance à notre code et pouvions donc lancer plusieurs tests sur l'ensemble des données.
Chaque exécution de modèle génère deux fichiers, tous deux accompagnés du nom du modèle en question : l'un conserve la structure du modèle en *.json* et l'autre sauvegarde les poids appris en *.hdf5*. On peut ainsi "re-créer" en local le modèle déjà appris et voir ce qu'il donne sur différentes images.

5 Résultats

Premiers résultats

Nos premiers résultats ont montré que notre réseau avait du mal à distinguer les axones et dendrites de l'actine. En effet, on voit ci-dessous que les deux labels retracent en grande partie l'image de l'actine, bien que les dendrites soient assez bien détourées. Afin de supprimer les faibles intensités, correspondant aux valeurs "peu sûres", nous appliquons un *threshold* en post-traitement.

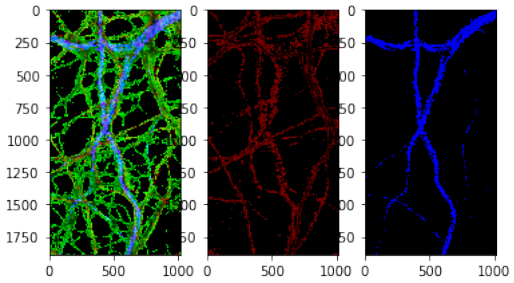

Figure 5.1: Images originales, avec l'image fusionnée, les axones et les dendrites

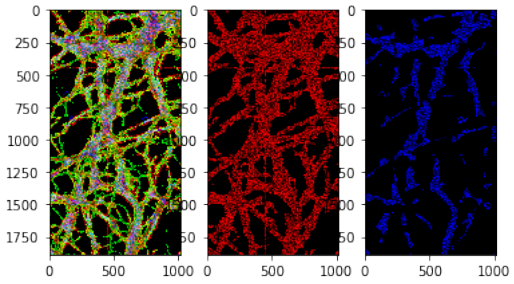

Figure 5.2: Images prédites, avec l'image fusionnée, les axones et les dendrites

Les résultats ci-dessus ont été obtenus avec le modèle issu de l'article de *Yang et al.*, entraînés sur 500 images d'origine. On obtient un erreur très élevée d'environ 45%.

6 Analyse

7 Contraintes

Les images fournies étaient en moyenne de haute résolution. Ainsi, les données avant augmentation pesaient déjà assez lourd, mais après augmentation nous avions un jeu de données d'environ 55 gigas. De plus, nous devions utiliser un ordinateur à distance afin d'avoir un GPU puissant, ce qui rendait les transferts de données compliqués (sans parler du fait que l'espace mémoire de l'ordinateur à distance était en grande partie utilisé). Cela a donc grandement ralenti le processus, étant donné que la sauvegarde du jeu de données entier après augmentation prenait un jour entier. Nous n'avons donc pas pu tester différentes augmentations sur l'ensemble des images d'origine.
L'entraînement était aussi particulièrement long ; nous disposions de plus de 110 000 matrices à traiter (après augmentation). Un entraînement sur 70% des données (en enlevant une partie pour les tests), pendant 30 epochs prenait une dizaine d'heures.

8 Pistes d'amélioration