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.***.

Tâches

La première chose à faire est d'affiner notre algorithme de masquage des images données par CERVO. En effet, nous avons déjà des résultats nous permettant d'avancer en parallèle sur l'architecture du réseau de neurones, mais nous souhaitons parfaire ces premiers résultats afin d'avoir des images en entrées avec le moins de bruit possible.

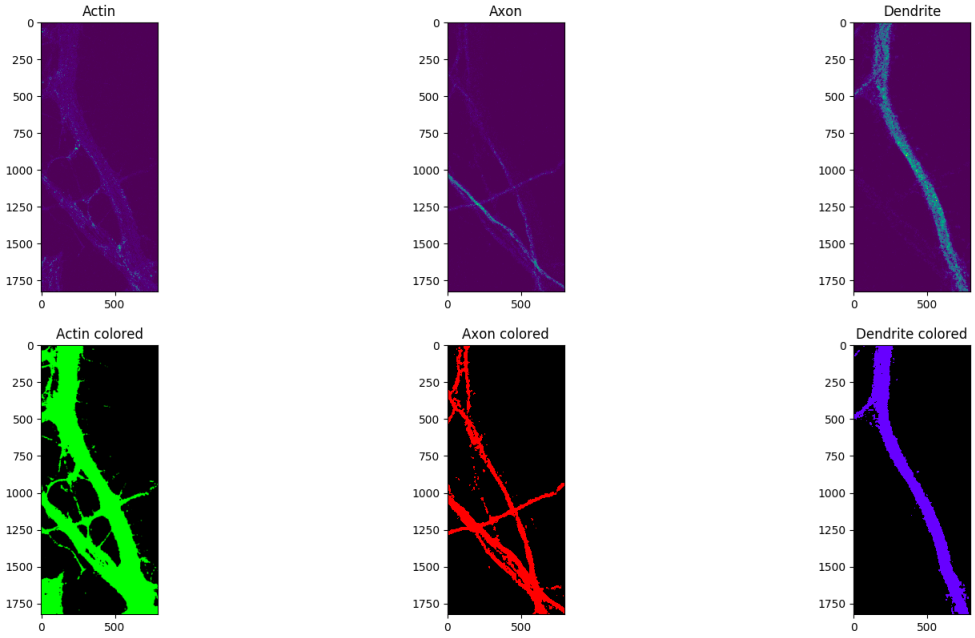

Figure 1.1: En haut, un triplet actine-axone-dendrite, et en bas les mêmes images une fois le masque appliqué

L'étape suivante est de créer une *pipeline* afin d'amener les images d'origine vers le réseau de neurones, en ayant appliquant les transformations et augmentations nécessaires. L'étape de transformation (en particulier le redimmensionnement) a déjà été effectué ; nous travaillons sur l'augmentation de données mentionnée plus haut.
Concernant le réseau en lui-même, nous avons déjà une architecture sur laquelle travailler. Nous utiliserons la librairie *Keras* avec un back-end en *TensorFlow*. L'architecture sera celle présentée plus haut, soit un **Encoder-Decoder**. L'encodeur amènera à de très petites images constituée d'un grand nombre de canaux, puis le décodeur ramènera à la taille d'origine, avec trois canaux (Rouger Vert Bleu). Nous avons commencé des tests avec un encodeur inspiré du réseau *AlexNet* ; le décodeur faisant seulement les opérations inverses (les *MaxPooling* étant substitués par des *UpSampling*). Selon nos recherches, la fonction de perte la plus pertinente à utiliser est *sigmoid_cross_entropy_with_logits* (https://www.tensorflow.org/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits).