Vision par ordinateur

Rapport de BE

Semantic Segmentation

MOD 3.2

Étudiants :
Morchoisne Dewone
Burie Thomas

Enseignant :
CHEN Liming

25 Janvier 2022

Sommaire

1 Introduction

La segmentation sémantique est un algorithme de Deep Learning qui associe une étiquette ou une catégorie à chaque pixel d'une image. Elle permet de reconnaître un ensemble de pixels qui forment des catégories distinctes. Par exemple, un véhicule autonome doit identifier des véhicules, des piétons, des panneaux de signalisation, des trottoirs et autres éléments de l'environnement routier.

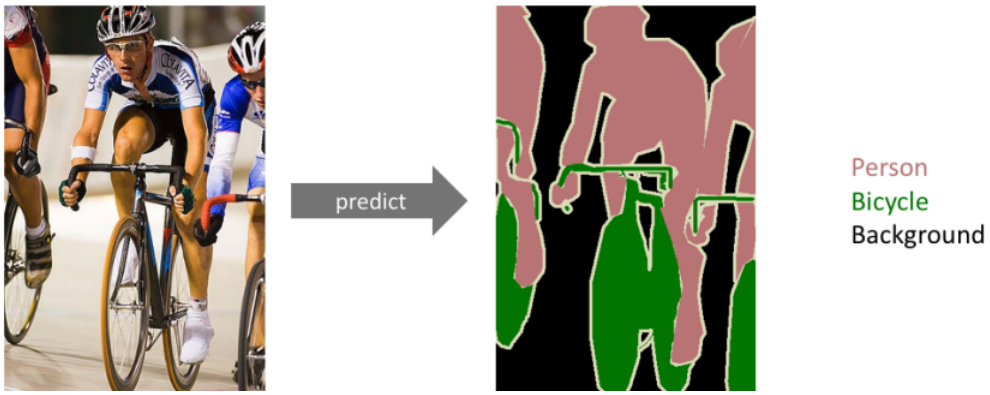

FIGURE 1 – Exemple de segmentation sémantique

La segmentation sémantique intervient dans de nombreuses applications telles que la conduite autonome, l'imagerie médicale et les contrôles industriels. Mais le problème que nous allons traiter est celui de la détection de sel dans sur des images sismiques.

2 Compréhension des données

Les données d'entraînement comportent 4000 images sismiques et 4000 masques noir et blanc qui correspondent à la vérité terrain lié à une image sismique sur la présence de sel ou non. Les zones présentant du sel sont représentées en blanc et les zones n'en présentant pas en noir. On a donc 2 classes de sortie Présence de sel ou Absence de sel pour chaque pixel. Les images ayant une taille de 101x101, un crop ne gardant que la partie en haut à gauche de l'image a été réalise pour obtenir des images de taille 64x64 ce qui sera utile étant donné l'utilisation d'un réseau Unet pour réaliser cette segmentation sémantique.

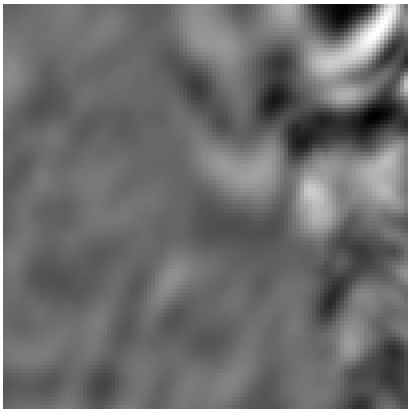

FIGURE 2 – Image Sismique

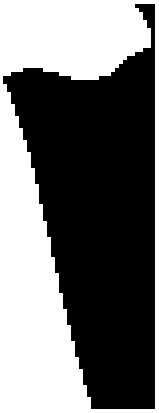

FIGURE 3 – Masque de l'image

3 Architecture Unet

L'architecture originale du réseau Unet crée par Olaf Ronneberger et utilisée pour la segmentation sémantique d'images médicale est presenté figure 4.

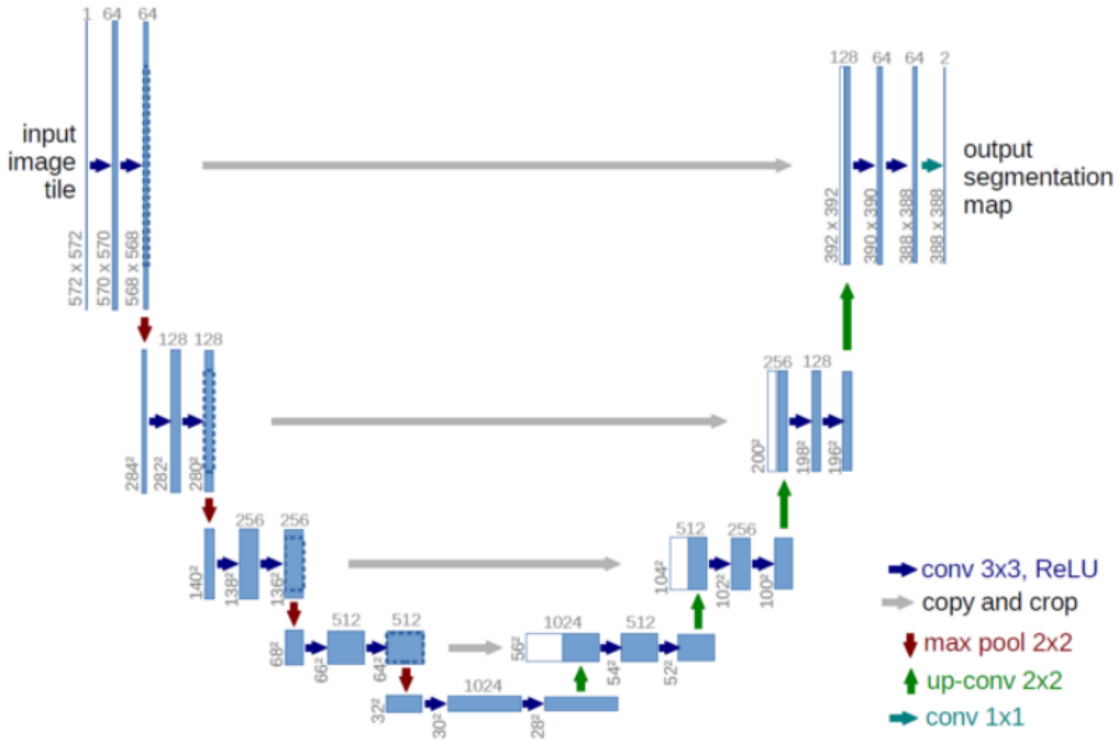

FIGURE 4 – Exemple de segmentation sémantique

Nous allons utiliser des images de taille 64x64, donc les tailles vont varier un peu mais l'architecture et les composants restent identiques. A noter que l'unsampling se fait en utilisant une convolution transposée.

L'architecture utilisée est la suivante :

— 1. Layer Doubleconv(3,16)
— 2. Layer Down(16,32)
— 3. Layer Down(32,64)
— 4. Layer Down(64,128)
— 5. layer Down(128,256)
— 6. Layer Up(256,128)
— 5. Layer Up(128,64)
— 6. Layer Up(64,32)
— 7. Layer Up(32,16)

Détaillons un peu les layers utilisés.

— Doubleconv(in-channel,out-channel) : 2 convolution successives ayant un nombre out-channel de filtres de taille 3x3 et un padding de 1 pour conserver les proportions de l'image. L'activation de ces convolutions se fait avec la fonction Relu et un layer de batchnorm est utilisé après chaque convolution.
— Down(in-channel,out-channel) : Un layer maxpool avec un filtre de taille 2 suivit d'un layer double conv(in-channel,out-channel).
— Up(in-channel,out-channel) : Un layer ConvTranspose2d(in-channel,in-channel/2, kernel-size = 2, stride = 2) pour faire de l'unsampling suivit d'un layer doubleconv(in-channel,out-channel)

4 Entraînement

La fonction de perte utilisée est la fonction **Binary Cross Entropy** étant donné qu'il n'y a que 2 classes à prédire (Présence ou Absence de sel) pour chaque pixel. Un mécanisme **d'Early-Stopping** a été mis en place, l'entraînement s'arrête si lors de 10 epochs consécutifs, la perte moyenne sur les données de validation ne s'est pas.améliorée. On utilise également le **Scheduler ReduceLROnPlateau** pour ajuster le taux d'apprentissage utilisé par **l'optimizeur Adam**. Si lors de 5 epochs d'affiliés, la perte moyenne sur les données de validation ne s'est pas améliorée, alors on multiplie par 0.1 le taux d'apprentissage.

On définit ici l'accuracy comme la proportion de pixels bien étiquetés dans une image.
Les courbes d'apprentissages sont les suivantes.

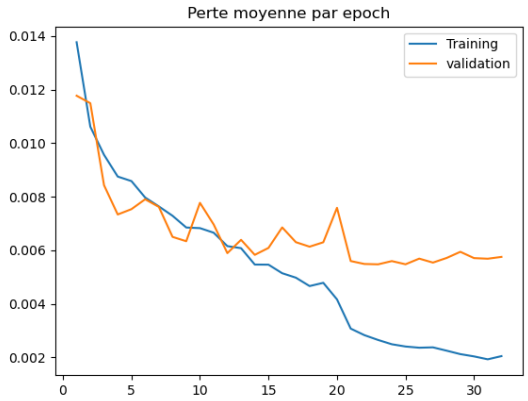

FIGURE 5 – Courbe de perte

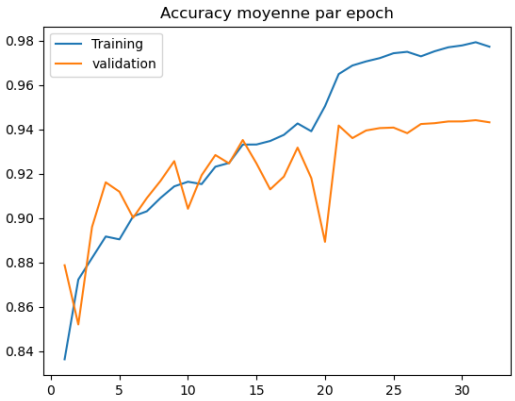

FIGURE 6 – Courbe d'accuracy

5 Résultats

5.1 Images d'entraînement

Les résultats sur les données d'entraînement sont assez bonnes avec une accuracy de 97.2%.

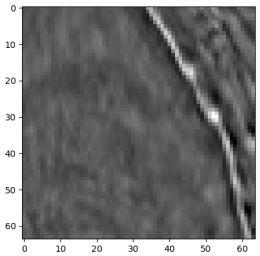

FIGURE 7 – Image Sismique

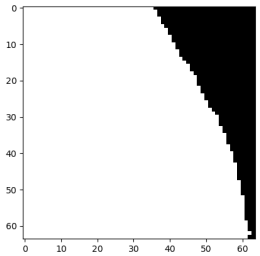

FIGURE 8 – Prédiction du masque

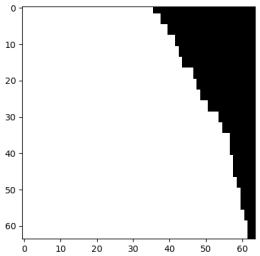

FIGURE 9 – Masque réel

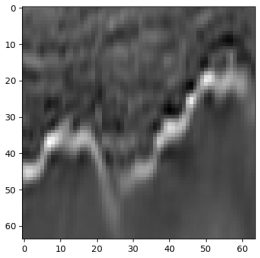

FIGURE 10 – Image Sismique

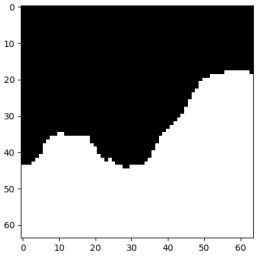

FIGURE 11 – Prédiction du masque

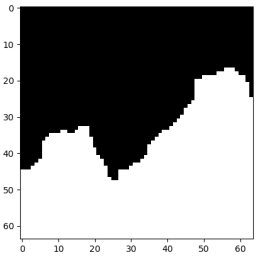

FIGURE 12 – Masque réel

5.2 Images de validation

Les résultats sur les données de validation sont également bonnes avec une accuracy de 93.7% ce qui signifie qu'il y a un léger overfitting. On observe également qu'il existe des cas où la prédiction du réseau est très éloigné de la vérité terrain comme dans l'exemple 1.

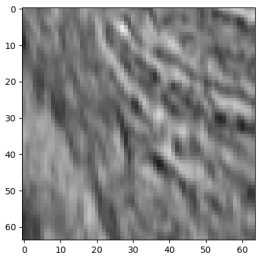

FIGURE 13 – Image Sismique

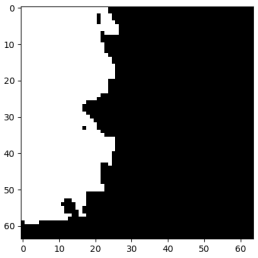

FIGURE 14 – Prédiction du masque

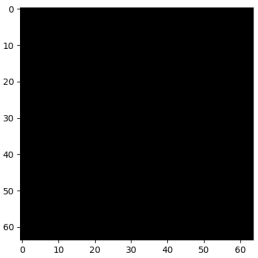

FIGURE 15 – Masque réel

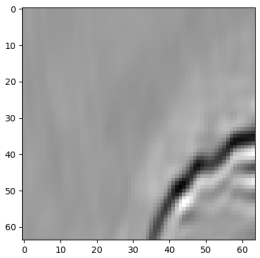

FIGURE 16 – Image Sismique

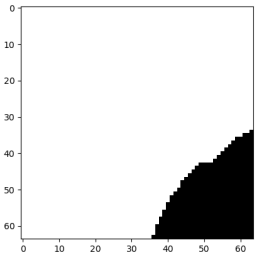

FIGURE 17 – Prédiction du masque

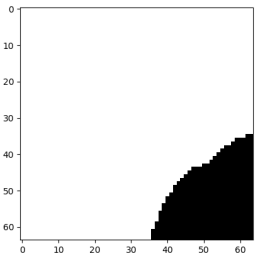

FIGURE 18 – Masque réel

6 Questions

— **5.1 Quelle est la différence entre l'utilisation d'une opération de maxpooling avec un filtre de taille 2 et l'utilisation d'un stride de 2 lors de la convolution ?**

En supprimant l'opération de maxpooling et en utilisant un stride de 2 pour la couche de convolution précédente, on réduit de manière significative le chevauchement opéré par une couche de convolution analogue de stride 1. Cela revient à réaliser l'opération de maxpooling mais à ne garder que la réponse en haut à gauche. Cela va avoir pour effet de réduire le nombre d'opérations mais la performance sera impactée négativement. Ce qu'il serait possible de faire pour améliorer les performances c'est de remplacer l'opération de maxpooling en ajoutant une couche de convolution utilisant des filtres de taille 2x2 et un stride de 2. Cela correspondrait à faire apprendre à notre réseaux l'opération de pooling qui ne serait pas forcément un max. Les résultats serait alors meilleurs mais le nombre de paramètres augmente si l'on ajoute cette couche.

— **5.2 En quoi les sauts de connections utilisés sont importants ? Supposons que l'on supprime ces connexions sautées de l'architecture proposée, quel impact cela a-t-il sur le résultat ? De plus, les sauts de connexion actuels consistent à concaténer les cartes de caractéristiques de l'encodeur avec celles du décodeur au même niveau, est-il possible d'utiliser d'autres alternatives, par exemple l'addition, l'opération Max ou Min ? Influent-ils sur les résultats ?**

L'architecture auto-encodeur sans sauts de connections va permettre de déterminer des caractéristiques qui vont permettre de déterminer les classes, mais les informations relatives à la localisation ainsi que les petits détails vont avoir tendance à être oubliés en passant dans l'encodeur. Pour conserver un souvenir de ces informations, on utilise des sauts de connections concaténant des features maps de même dimensions qui serviront d'entrées au décodeur. Les sauts de connections peuvent également de manière générale faciliter le flux du gradient, éviter les problèmes de gradient évanescent et rendre plus lisse les surfaces de loss et ainsi faciliter l'apprentissage.
Il serait possible de réaliser une sommation des features maps de manière un peu similaire à ce que l'on peut retrouver dans un réseau resnet. La concaténation semble néanmoins plus naturelle dans ce cas étant la nature de l'architecture. La concaténation des features maps se fait en entrée d'une couche du décodeur et permettent de reconstruire les images de dimension supérieur. Selon les créateurs des Dense Net, la sommation devient un frein au flux de gradient en polluant les features maps obtenues après sommation. Les 2 méthodes pourraient cependant à priori fonctionner dans ce cas. Pour ce qui est des opérations min ou max, elles ne me semblent pas vraiment pertinentes en terme de réduction de l'information dans le cas présent mais elles pourraient peut être quand même fonctionner.

— **5.3 L'architecture proposée est une architecture de type auto-encodeur, c'est-à-dire composée d'un encodeur et d'un décodeur. Est-ce nécessaire ? Est-il possible que nous utilisions un réseau de neurones entièrement convolutif (FCN) constitué de séries de convolutions de l'entrée à la sortie tout en gardant la taille de l'image, c'est-à-dire la largeur et la hauteur, pendant toutes les convolutions ?**

L'architecture de type auto-encodeur permet de déterminer les caractéristiques de l'image qui influence le choix de sa classe. Cette architecture est performante mais pas indispensable, il serait en effet possible d'utiliser une architecture entièrement constituée de couches de convolutions dont les entrées et sorties seraient de même taille. Cependant, l'architecture de type auto-encodeur possède un nombre de paramètres d'apprentissage bien plus faible et le temps de calcul pour effectuer une prédiction est également bien plus faible. Cela permet donc de réduire drastiquement le temps d'apprentissage et de réaliser des inférences en temps réel. Il serait probablement possible d'obtenir des résultats similaires en utilisant un FCN mais le temps d'apprentissage serait beaucoup plus élevé.

— **5.4 Nous avons pris 0,5 comme seuil pour décider de classer un pixel comme sel ou non. Cependant, ce choix s'avère ne pas être optimal en termes de précision et de rappel pour les pixels étant classés sel. Expliquez comment nous devrions définir la précision et le taux de rappel dans notre analyse. Proposez une stratégie pour améliorer à la fois la précision et le rappel ?**

De manière général, la précision et le rappel sont définis de la manière suivante.

$$\textbf{précision} : \frac{Vrai\ positifs}{Vrai\ Positifs + Faux\ Positifs}$$
$$\textbf{rappel} : \frac{Vrai\ positifs}{Vrai\ Positifs + Faux\ négatifs}$$

Pour améliorer la précision et le rappel, on pourrait faire en sorte que le seuil ne soit pas arbitrairement fixé à 0.5 mais soit également un paramètre appris par le réseau.