POLYTECH
NICE SOPHIA

2021-2022

Détection de tumeurs avec un réseau neurone en imagerie médicale

Projet de fin de semestre MAM4

Yassine Bounou
Nada Benchekroun

- Encadrant de projet : Monsieur Lionnel Filatre
- Responsable pédagogique : Madame Christine Malot

Table des matières

Introduction

La radiologie médicale est devenue une méthode largement utilisée d'imagerie médicale de haute qualité. Cependant, les techniques de détection des tumeurs du foie doivent être fiables et robustes pour que les diagnostics soient corrects. À cette fin, l'objectif du système proposé est de minimiser les fausses alarmes, d'accroître les performances de la détection précoce et de contribuer à réduire le taux de mortalité par les tumeurs malignes. Le système dépend des systèmes d'imagerie, de la segmentation, de l'extraction des caractéristiques, des systèmes de classification utilisant un réseau neuronal. Ce dernier va nous permettre de localiser ces tumeurs et d'estimer leurs tailles. De cette manière, les médecins peuvent exploiter ces informations pour diagnostiquer la maladie mais aussi suivre l'évolution du traitement.

1. Concepts

1.1 Le Deep learning

Le Deep Learning est une forme spécifique de Machine Learning imitant le fonctionnement du cerveau humain pour classifier les données qui lui sont fournies, un réseau de neurones doit être entraîné au préalable.

Durant les dernières années, l'apprentissage profond a fait l'objet de nombreuses études et a obtenu des résultats remarquables dans de nombreux domaines de la vision par ordinateur, notamment la classification des images, la reconnaissance vocale, la détection d'objets, la segmentation sémantique, le traitement du langage naturel, la capture des images, l'estimation de la pose, la détection de la saillance, la détection des contours, etc. Il a montré ses performances face aux différents problèmes de l'intelligence artificielle dépassant ainsi les algorithmes classiques de Machine Learning. Dans une comparaison entre les méthodes traditionnelles de l'apprentissage automatique, le Deep Learning a été mis en place.

La prédiction liée au Deep learning peut être correcte ou incorrecte, et c'est au programmeur de le corriger en cas d'erreur. Au fil des essais, le système apprend de ses erreurs et son taux de réussite s'améliore jusqu'à ce que ses prédictions deviennent justes.

1.2 Contextualisation du sujet :

Dans le domaine de l'imagerie médicale, le prétraitement des images radiographiques est essentiel afin de détecter efficacement des tumeurs cancéreuses. Il s'agit également de localiser ces tumeurs et d'estimer leurs tailles. De cette manière, les médecins peuvent exploiter ces informations pour diagnostiquer la maladie mais aussi suivre l'évolution du traitement. Dans ce projet, nous chercherons à détecter des tumeurs cancéreuses dans le foie en utilisant un réseau de neurones. Nous utiliserons des données en accès libre qui ont été annotées par des experts (radiologistes) sur Kaggle, les tumeurs sont donc connues et localisées. La figure ci-après donne une idée des images gui seront utilisées :

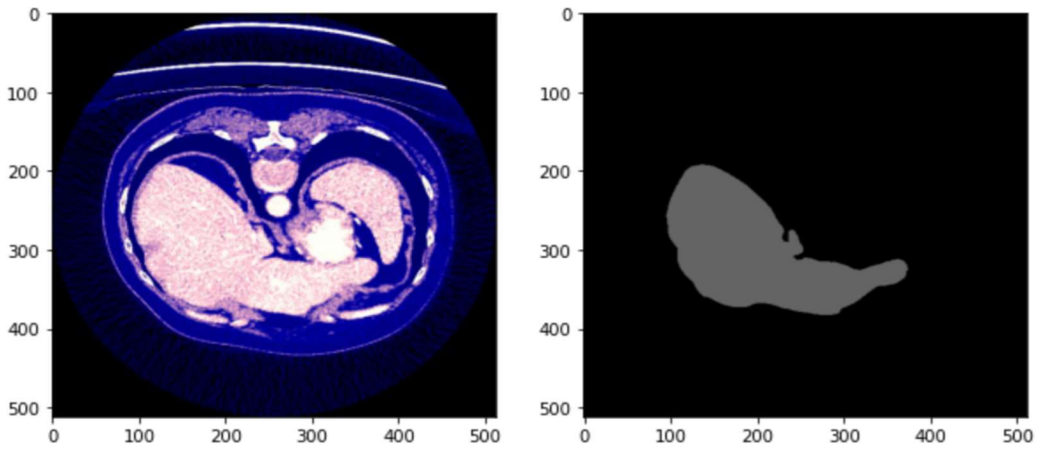


Fig 1 : Exemple de figure à traiter : image de radio + image mask

Nous pouvons voir l'image originale et l'image "mask" qui représente les pixels associés au foie.
Cette image "mask" représente l'annotation fournie par les spécialistes. Tout au long de ce projet, on est convenu alors à construire un algorithme qui permet de classifier cette image (fond noir, foie, tumeur) de manière automatique dans l'image originale.
Dans le domaine du traitement d'images, les méthodes les plus efficaces actuellement s'appuient sur des réseaux de neurones profonds. Le but de ce projet est donc de développer un réseau de neurones profond pour détecter les tumeurs dans les images du foie.
L'objectif principal de notre projet consiste principalement à réaliser deux tâches principales :
La première tâche consiste à concevoir et programmer un réseau de neurones profond pour détecter et localiser les tumeurs cancéreuses. Il s'agit notamment d'identifier quels sont les réseaux de neurones actuellement disponibles qui pour traiter efficacement résoudre le problème de détection abordé dans le projet.
La deuxième tâche consiste à évaluer les performances de notre algorithme développé sous python, afin de conclure sur sa pertinence.
Lors de ce projet, il était nécessaire de s'autoformer en Pytorch qui est un package de Python très utile en Deep Learning pour l'utiliser dans l'intégralité du projet.

1.3 Application des réseaux neurones en classification :

La classification est l'une des tâches de prise de décision les plus fréquentes de l'activité humaine. Un problème de classification se produit lorsqu'un objet doit être affecté à un groupe ou une classe prédéfinie en fonction d'un nombre d'attributs observés liés à cet objet. Des chercheurs dans différents domaines ont eu recours aux réseaux de neurones.

Un neurone artificiel n'est rien d'autre qu'une opération mathématique relativement simple. La complexité repose avant tout dans l'interconnexion de plusieurs neurones

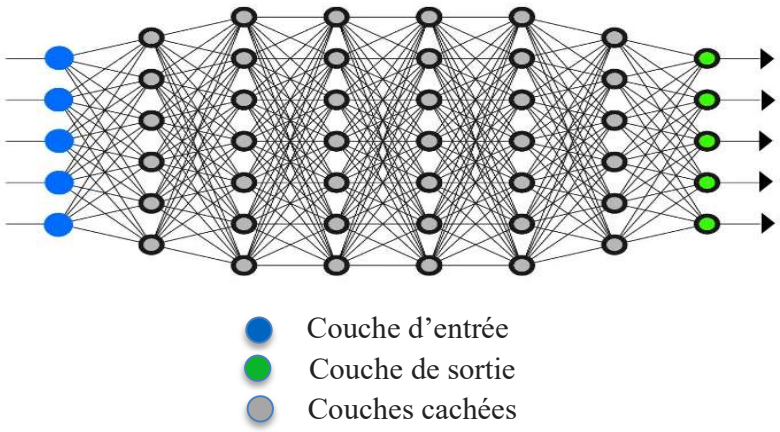


Fig 2 : Structure d'un réseau neuronne

L'architecture d'un réseau neuronne disposant de nombreuses couches cachées (c'est à dire de nombreuses couches de neurones situées entre les couches d'entrées, acceptant des données à traiter, et les couches de sortie, destinées à délivrer le résultat du calcul). L'ajout de ces couches de neurones a eu un impact extrêmement bénéfique sur la qualité des résultats obtenus.

2 Mise en œuvre :

2.1 Mise en place de l'environnement :

Google Colaboratory

L'ensemble de notre projet a été développé sur Google Colab qui est un environnement de notebook Jupyter gratuit qui s'exécute entièrement dans le cloud où il n'y a pas de processus de configuration. Google Colab prend en charge de nombreuses bibliothèques populaires en Deep Learning qui peuvent être facilement chargées (torch – keras…).

Google Colaboratory a une fonctionnalité qui le distingue est l'accès gratuit à un processeur graphique GPU (Graphics Processing Unit). Pour cause, les modèles de Deep Learning requièrent beaucoup de temps pour être entraînés. Les GPU surpassent tous les processeurs graphiques grâce à son parallélisme. Les réseaux de neurones, eux-mêmes, reposent sur le calcul parallèle. Les opérations telles que le calcul de poids et les fonctions d'activation de chaque couche du réseau ou la rétropropagation peuvent être effectués en parallèle.

En effet, il existe différents frameworks pour la programmation GPU. L'un des plus populaires est **CUDA** : Compute Unified Device Architecture. Initialement lancé en 2007, ce framework permet d'exploiter pleinement la puissance du GPU pour le calcul parallèle de manière optimisée pour de meilleures performances.

2-2 Définition du modèle

2-2-1 Le réseau neurone U-net :

UNet, qui a évolué à partir du réseau neuronal convolutif traditionnel, a été conçu et appliqué pour la première fois en 2015 par Oral Ronneberger pour la segmentation d'images médicales. En général, le réseau de neurones convolutifs se concentre sur la classification d'images, où l'entrée est une image et la sortie un label, mais dans les cas biomédicaux, il faut non seulement distinguer s'il y a une maladie, mais aussi localiser la zone d'anomalie. Le modèle UNET est donc le modèle qui permet de localiser et classifier en faisant une classification de chaque pixel, donc l'input et l'output doivent avoir la même taille.

L'architecture du modèle contient deux chemins. Le premier chemin est le chemin de contraction (également appelé **l'encodeur**) qui est utilisé pour capturer le contexte dans l'image. L'encodeur n'est qu'une pile traditionnelle de couches convolutionnelles et de mise en commun maximale (max pooling). Le deuxième chemin est le chemin d'expansion symétrique (également appelé **décodeur**) qui est utilisé pour permettre une localisation précise en utilisant des convolutions transposées. Il s'agit donc d'un réseau entièrement convolutif (FCN) de bout en bout, c'est-à-dire qu'il ne contient que des couches convolutives et aucune couche dense, ce qui lui permet d'accepter des images de toute taille.

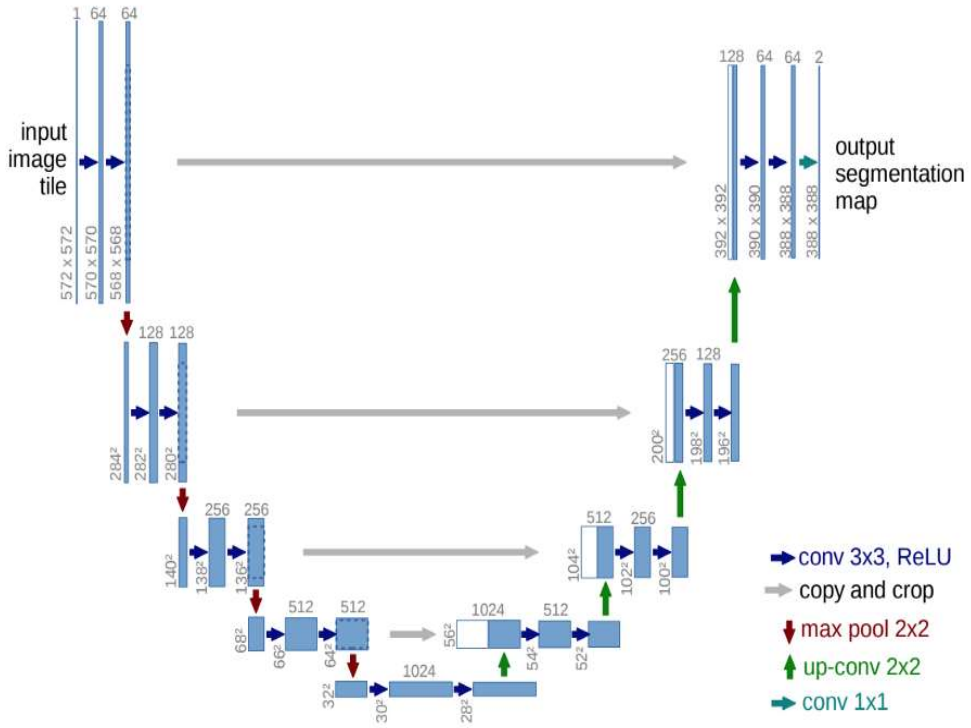


Fig 3 : Architecture du réseau U-Net

2.2.2 U-Net utilisé dans le cadre de notre projet :

Dans le cas de notre projet, nous avons implémenter un modèle UNET simplifié qui consiste à faire une convolution puis un max pooling à la place de plusieurs convolutions puis un max pooling, afin d'optimiser le temps d'exécution du code même si on perd un peu de précision car les performances de nos outils utilisés ne nous permettent pas d'avoir une bonne exécution du modèle.
L'image d'entrée est une image de taille 512 x 512 et de 3 canaux. L'image de sortie du modèle est de taille 512 x 512 x 3, car le nombre de classes à classifier est de 3 (fond de l'image, fois, tumeurs).

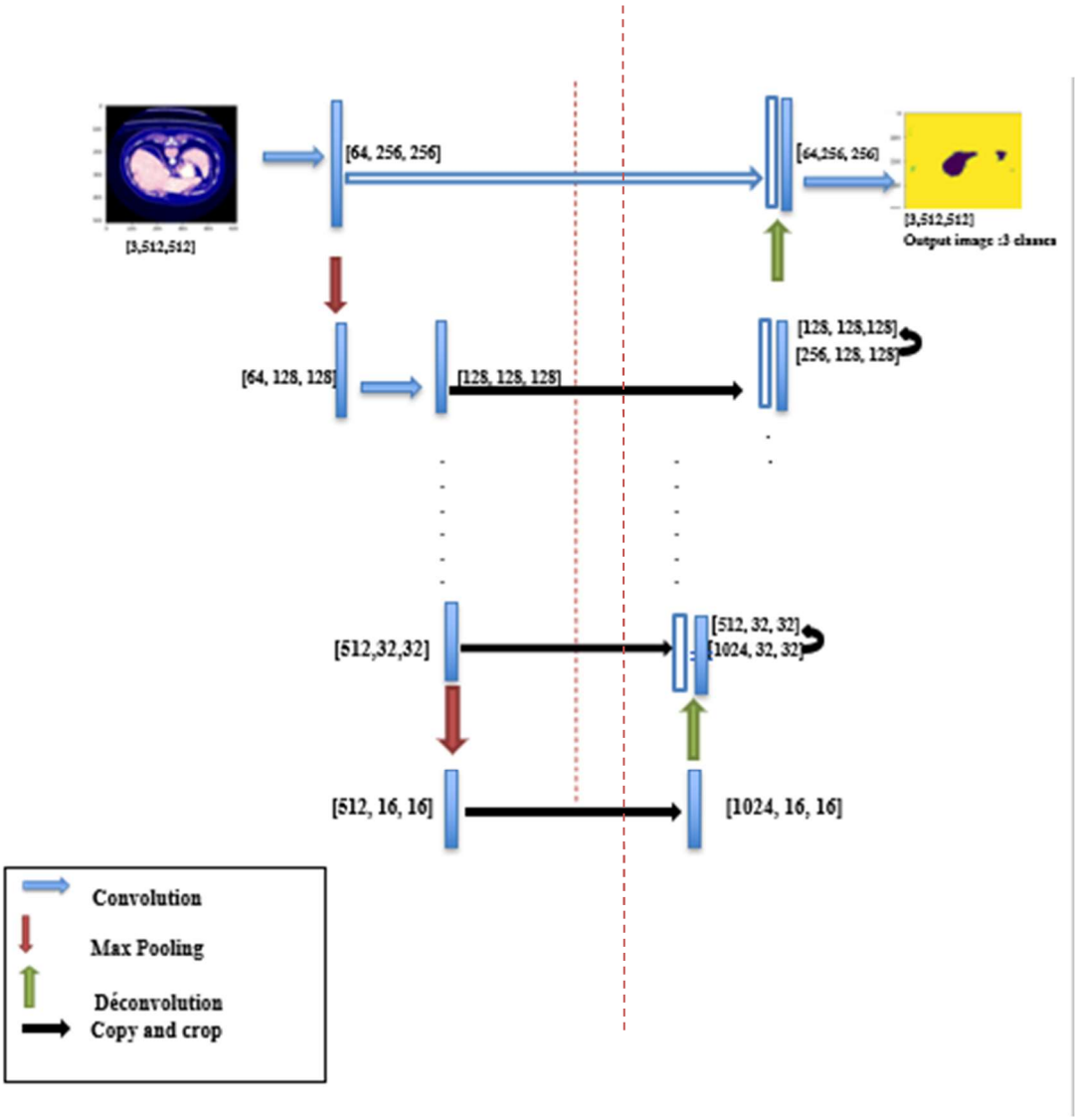


Fig 4 : notre propre U-Net utilisé dans le cas de notre projet

2.2.3 les couches du réseau neurone convolutif U-Net

- **La Convolution :**

La convolution met en jeu deux matrices : la matrice image I, (par exemple 512 x 512= 262144 pixels) et une matrice K nommé le noyau. Cette dernière va donc agir sur chacun des pixels, c'est dire sur chacun des éléments de la matrice "image" en input. L'image en input est représentée par la matrice I de taille Nin x Min. Le noyau est quant à lui composé de la matrice carré K de taille 3x3. Appliquer un filtre de convolution consiste à multiplier chacun des pixels de la matrice I par le noyau K.

Le processus de convolution se passe de la façon suivante :

- La matrice filtre K est superposée sur la matrice image I.
- Chaque coefficient du filtre est multiplié avec le pixel qu'il recouvre, puis on additionne toutes les valeurs pour obtenir la valeur de sortie du pixel courant ;
- Cette opération est répétée pour tous les pixels de l'image.
- À chaque portion d'image rencontrée, un calcul de convolution s'effectue permettant d'obtenir en sortie une feature map.

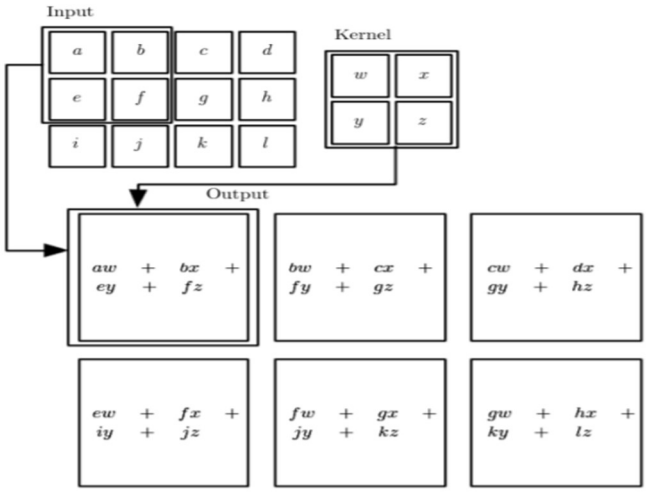


Fig 4 : Schématisation mathématique de la convolution

Le produit de convolution est donné par :

$$S(i,j) = (I * K)(i,j) = \sum_{i=0}^{n} \sum_{i=0}^{m} I(i-m, i-n)K(m,n)$$

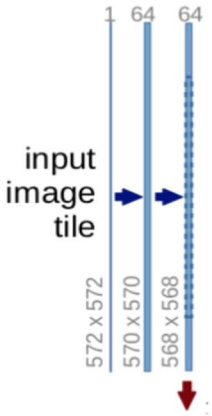
L’opération de convolution en U-Net est une opération qui consiste à prendre en paramètre l’image input de taille (Nin x Nin) et un filtre pour sortir une nouvelle image output de taille (Nout x Mout) avec un nombre de canaux plus grand. Trois hyperparamètres permettent de dimensionner le volume de la couche de convolution (aussi appelé volume de sortie) : la profondeur, le pas et la marge (padding, stride…).

La formule mathématique qui met en relation la taille d’entrée et la taille de sortie est la suivante :

$$\mathbf{Mout} = \left[\frac{\boldsymbol{Nin} + \boldsymbol{2p} - \boldsymbol{k}}{\boldsymbol{s}}\right] + \mathbf{1}$$

- Nin : number of input features
- Mout : number of output features
- K : convolution kernel size
- p : convolution padding size
- s : convolution stride size

Dans le modèle U-Net décrit précédemment, on fait plusieurs étapes de convolution consécutives avant un max pooling :



Chaque processus constitue deux couches de convolution, et le nombre de canaux change de 1 → 64, car le processus de convolution augmentera la profondeur de l'image. La flèche rouge pointant vers le bas est le processus de max pooling qui réduit de moitié la taille de l'image (la taille réduite de 572x572 → 568x568 est due à des problèmes de padding, mais l'implémentation ici utilise padding= "same").

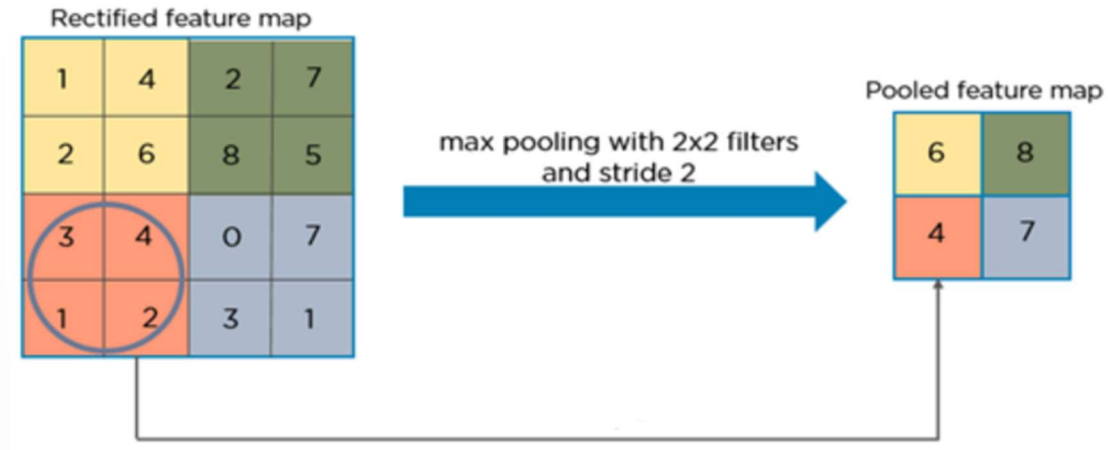
Dans notre cas étudié, dans chaque couche, il n’y aura qu’une seule convolution afin d’optimiser le temps d'exécuter de l’algorithme en passant d’abord de 3 canaux initiaux à 64 canaux, puis 128, 256, 512 et 1024.

- **Max Pooling :**

La fonction du Max Pooling est de réduire la taille de la première couche de convolution d'entrée pour avoir peu de paramètres dans le réseau : Dans chaque bloc 2x2 de de l'input de la feature map, on sélectionne la valeur du pixel max pour obtenir une couche dite "Pooled feature map". L'idée générale est de garder juste les valeurs maximums du pixel et de jeter que les informations importantes qui décrivent au mieux le contexte de l'image.

Exemple :

Supposons qu'on a une matrice 4x4 représentant notre entrée initiale et qui représente la première couche de convolution. Après l'application du Max Pooling, on obtient en sortie une carte caractéristique pooled dite " Pooled feature map" de taille 2x2.



Max(3,4,1,2)= 4

Fig 5: processus du max pooling

La sortie du tenseur a une taille 2x2 dont les éléments (i,j) représentent le maximum de chaque fenêtre de pooling :

max(1,4,2,6)= 6
max(2,7,8,5)=8
max(3,4,1,2)=4
max(0,7,3,1)=7

Dans l'architecture U-Net, le Max pooling est représenté avec les flèches descendantes rouges. Il réduit à moitié la taille de l'image en entrée. La taille est réduite de **280x280 —>140x140**

- **Déconvolution (Convolution Transposée) :**

Le principe de la déconvolution suit le processus opposé de la convolution normale.

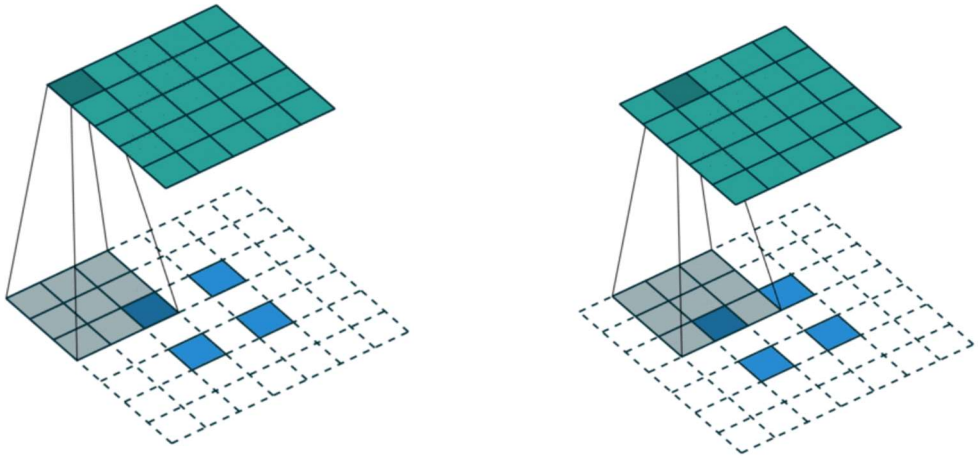


Fig 6 : Schématisation de la déconvolution dans le cas général

Dans le décodeur, l'image est agrandie à sa taille original :

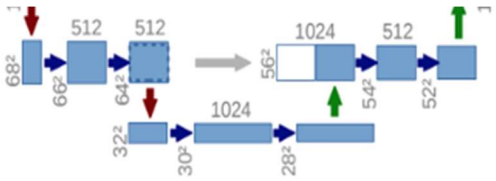


Fig 7 : Déconvolution suivi d'un copy and crop selon U-Net

Après la convolution transposée, l'image est agrandie de 28x28x1024 → 56x56x512, puis elle est concaténée grâce au copy and crop (flèches horizontales grises) avec l'image provenant de l'encodeur (image après application de la convolution), et les deux donnent une image de taille 56x56x1024. Le but est de combiner l'information des couches précédentes pour avoir plus de prédiction.
Ce processus est répété plus que 3 fois dans la partie décodeur selon le processus suivant : **ConvTranspose2d -> Torch.cat (concatenate) -> contracting_11_out -> contracting_12_out**.

On atteint maintenant le summum de l'architecture U-Net, et la dernière étape est de réformer la taille de l'image pour avoir un output ***Y_pred*** qui satisfait notre prédiction.

3 Préparation des données

La base de données sur laquelle on va travailler est disponible en accès libre sur kaggle w.kaggle.com/andrewnvd/liver-tumor-segmentation et a été annotée par des experts (radiologistes et oncologues): les tumeurs sont donc connues et localisées. La base de donnée contient 572 images radiographiques originales et leur images "masks" correspondants, soit 1144 images au total, en format NII. Les fichiers dont l'extension est NII sont des fichiers de type "NIfTI-1 data format by Neuroimaging Informatics Technology Initiative", c'est donc un format de fichiers dédié à l'imagerie médicale. Étant très volumineux (le dataset en entier est de taille 49,9 Go), la manipulation des images radiographiques sous le format NII prend beaucoup de temps et de mémoire. Il a été pensé donc à transformer ces images en format png qui est beaucoup moins volumineux et plus facile à manipuler. On passe à la réunion de l'ensemble des images png dans un nouveau dataframe en associant à chaque image originale son image "mask".

4 Entrainement et Résultat :

Pour avoir une prédiction précise, nous avons divisé la dataset en 3 parties, une partie réservée à l'entrainement, une autre partie pour le test et une dernière partie pour la validation du modèle et donc effectuer des prédictions.
Afin d'avoir un résultat de prédiction, il est nécessaire d'entrainer le modèle plusieurs fois pour gagner en précisions (minimiser l'erreur de prédiction) en utilisant un optimizer.

4-1 Cross-Entropy Loss :

La Cross-Entropy Loss mesure la performance du modèle pour un pixel c'est-à-dire la différence entre la valeur de sortie de prédiction y_pred du pixel pour un label donné et la valeur de l' image d'entrée y.

Le but de notre modèle est de réduire au maximum cette différence dite « le criterion » et avoir une fonction de perte la plus minime possible. Il faut que la sortie de notre réseau U-net soit le plus proche possible à la sortie désirée (valeurs réelles).

Cross-entropy Loss mesure la performance de notre modèle de classification dont la valeur de sortie a une probabilité **p** entre **0** et **1** ***(p ∈ [0,1])***

Explication probabiliste

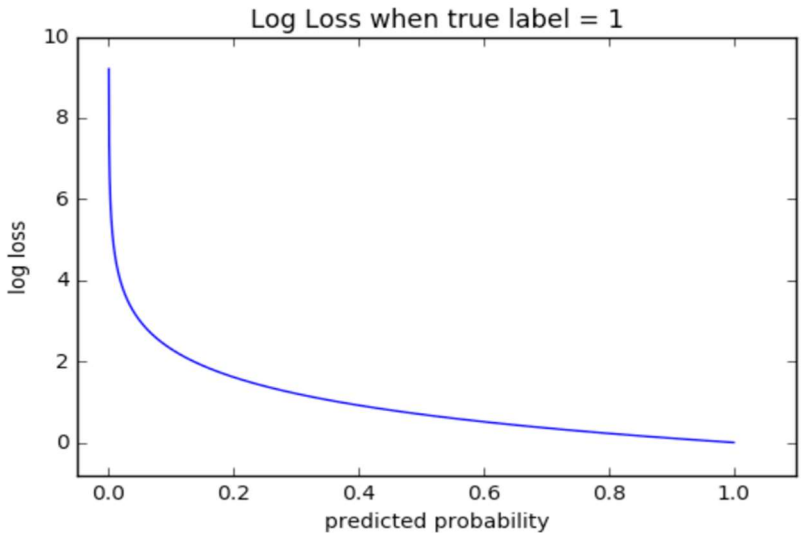


Fig 8 : Graphe d'évolution de la perte par rapport aux probabilités prédites

Le graphique ci-dessus montre la plage de valeurs de perte possibles compte tenu d'une observation vraie. À mesure que la probabilité prédite **p** se rapproche de 1, la perte logarithmique diminue lentement. Cependant, à mesure que la probabilité prédite **p** diminue, la perte de log augmente rapidement.
Cross entropy Loss dans notre cas d'étude prend les deux paramètres suivants :

- Predicted output (y_pred) : sortie de réseau neurone y_pred pour un pixel
- Target value : valeur de l'image réelle y

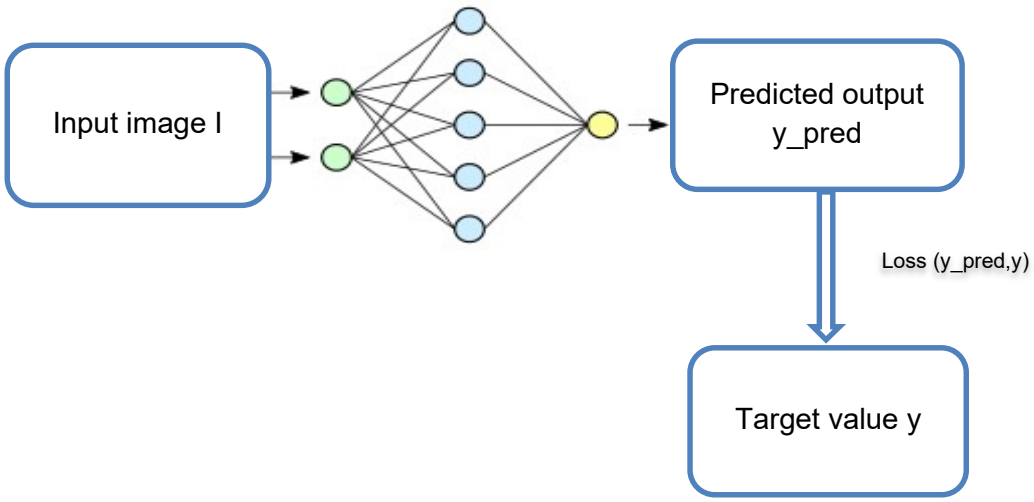


Fig 9 : Illustration de la fonction de perte Cross-Entropy Loss d'un réseau neurone

Cette fonction va déterminer les performances de notre modèle en comparant le " predicted output" à l'expected output. Si la déviation entre **y_pred** et y est large, la perte dite "loss" va être très grande.

La fonction Cross Entropy loss en Pytorch est exprimé de la sorte :

$$loss(y\_pred, y) = log(\frac{exp(ypred[y]}{\sum_j \ exp(y\_pred[j]}) = -\ y\_pred[y] + \ log(\sum_j exp(y\_pred[j\ ])$$

4.2 Optimizer Adam (Adaptative Momentum estimation)

Un optimizer a un rôle principal pour régler les paramètres du réseau neurone afin de minimer la loss function cité ci-dessus. L'optimizer avec lequel on se servir pendant l'entrainement de notre modèle est l'optimizer Adam. Il s'agit de l'optimizer le plus utilisé **(**surtout pour les réseaux de neurones) en raison de son *efficacité* et de sa stabilité. C'est une méthode de descente de gradient stochastique dont l'algorithme d'optimisation suit le gradient négatif d'une fonction objectif afin de localiser le minimum de la fonction Loss (y_pred, y).

Pour le présenter, nous devons d'abord saisir la notion de momentum. Par exemple, jetez un rocher dans la pente d'une montagne : à mesure qu'il va rouler dans la même direction il va gagner en vitesse, mais s'il tourne, sa vitesse sera réinitialisée. Pour **Adam**, le principe est identique : tant que le gradient est dans la même direction que ceux précédents, on va accélérer la vitesse de l'apprentissage.

4.3 Résultat :

En pratique, le choix des données d'entrainement, de test et de validation a été établis d'une façon aléatoire pour pouvoir exploiter plusieurs images différentes (images noirs, images sans tumeurs…). Nous avons essayé de prendre un nombre de données pour l'entrainement assez faible dû aux performances de nos outils. Dans le cas du choix de plusieurs données pour l'entrainement, une erreur d'allocation de mémoire est affichée chez tous les membre du groupe de projet, ainsi nous avons décidé d'entrainer le modèle sur une nombre petit de données par rapport à la base de données initiales pour pouvoir avoir des résultats.

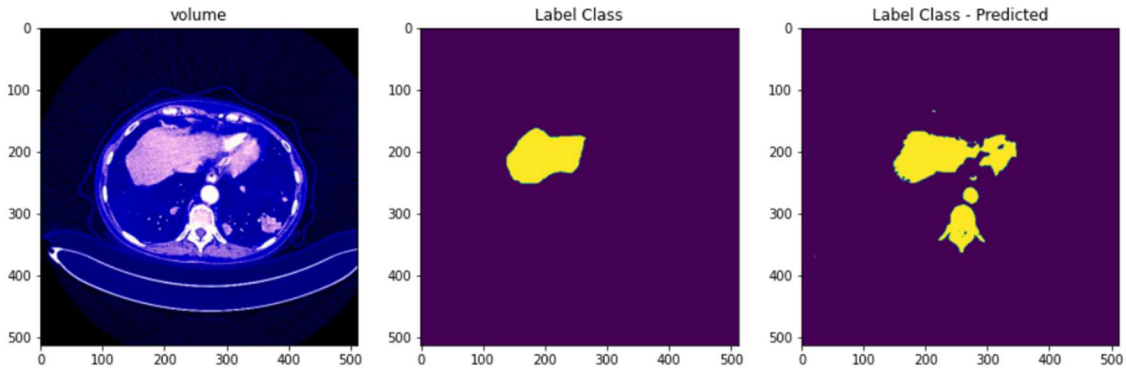


Fig 10 : une image de radio avec son image mask prédite

Dans cette image, à gauche nous retrouvons une image aléatoire de radio accompagné de son image mask correspondante, et à droite l'image mask prédite par le model UNET implémenté. On remarque alors que notre prédiction arrive à détecter la présence d'un foie dans cette image et ne détecte pas de tumeurs, qui n'est pas présente initialement dans l'image.

Pour mieux évaluer notre prédiction, on a comparé l'erreur de prédiction (cross entropy) de l'apprentissage et celle de la phase de test.

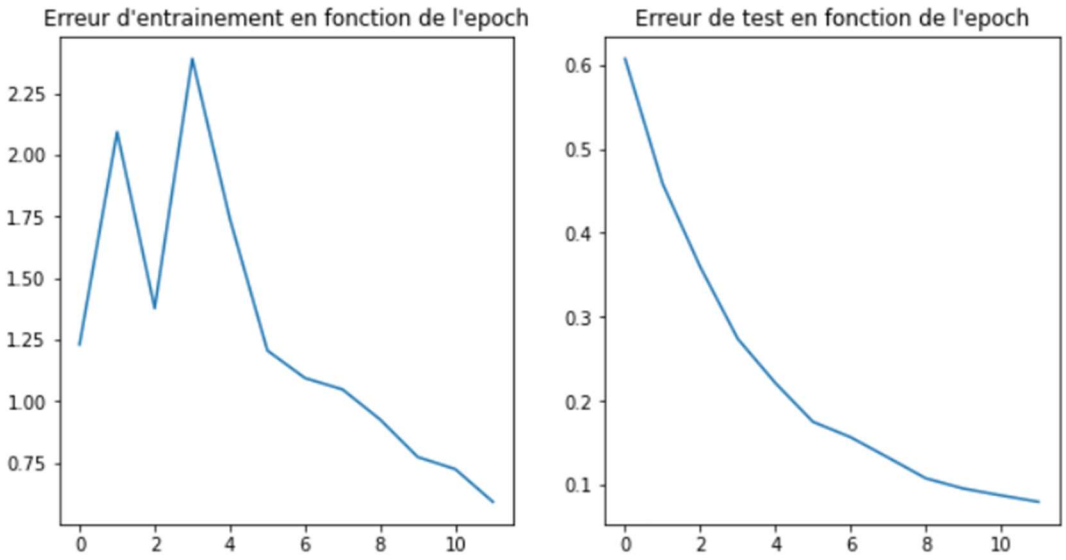


Fig 11 : Erreur d'entrainement et erreur de test en fonction de l'epoch

On remarque que l'erreur de test est plus lisse que celle de l'apprentissage, on peut aussi remarquer que l'erreur d'apprentissage est supérieure à celle de l'erreur test. Ce résultat semble logique car dans notre cas, on a entrainé en utilisant un nombre petit de données, ce qui ne permet pas un bon apprentissage du modèle pour avoir des résultats très correct (la prédiction effectuée sur l'image n'était pas précise aussi).

Pour améliorer le modèle, il faut l'entrainer en utilisant plusieurs images (un grand nombre d'images) tout en évitant l'overfitting.

Conclusion :

Notre conclusion sera une synthèse de plusieurs facteurs interdisciplinaires dans une nouvelle et inattendue découverte où on a pu, sur une période très courte, avoir les yeux ouverts sur le mode du Deep Learning. De l'avis général, nous avons pu consolider nos connaissances acquises pendant le semestre, de plus ce projet nous a fait découvrir des nouveaux concepts intéressants, liant programmation en python, apprentissage de Pytorch et réflexion mathématique et extension de nos savoirs en Deep Learning. C'était aussi l'occasion de connaître les différentes étapes d'une prédiction par un data scientist, commençant par l'extraction des données et son nettoyage, l'analyse des données puis l'entrainement et prédiction, et enfin la discussion du résultat.

Nous sommes globalement satisfaits de ce que nous avons réalisé dans ce projet car nos principaux objectifs ont été atteints dans les délais grâce à l'esprit d'équipe de chaque collaborateur avec son caractère distinct, à notre efficacité organisationnelle et notre communication qui fut fluide et opérationnelle avec le groupe Image. On remercie également Monsieur Filatre pour son implication et son encadrement, grâce aux réunions hebdomadaires organisées en sa compagnie, on a pu avancer d'une façon incrémentale et bien s'approprier le sujet.

Enfin, on ne pourrait qu'être reconnaissants pour ce choix de sujet qui a été pour nous fructif et riche en connaissance. Ce fut pour nous une agréable expérience !

Bibliographie :

https://pytorch.org/tutorials/

https://www.kaggle.com/gokulkarthik/image-segmentation-with-unet-pytorch?fbclid=IwAR0CBti6APwtQEfUHHoYHL544mR8nqK9cgPAWRV5casAQ9I_XK6wNnwB28Y#5.-Define-Model

https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5

https://towardsdatascience.com/u-net-for-semantic-segmentation-on-unbalanced-aerial-imagery-3474fa1d3e56

https://datascientest.com/convolutional-neural-network

https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e