Segmentation automatique du système respiratoire à partir des méthodes d'apprentissage profond

Yanisse FERHAOUI – Abdennour SLIMANI – Hafid OUCOUC

Encadrant : Hamid LADJAL

Résumé :

Ce projet de recherche s'inscrit dans le contexte de la lutte contre le cancer des poumons, avec pour objectif ultime la segmentation automatique de l'appareil respiratoire humain. Cette segmentation permettra de prédire les mouvements respiratoires, et par conséquent le déplacement d'une tumeur dans les poumons. Pour ce faire, nous avons utilisé des outils tels que ITKSnap pour la segmentation manuelle de radiographies thoraciques, ainsi que Keras et Tensorflow pour construire un modèle de réseaux de neurones convolutifs. Ce modèle a été entraîné sur les radiographies segmentées afin de réaliser la prédiction des mouvements respiratoires.

Mots-clés : CNN – Intelligence Artificielle – Computer Vision – Segmentation – Tensorflow

Table des matières

1. Introduction

La radiothérapie externe est une méthode de traitement du cancer qui vise à délivrer une dose létale de rayonnement dans la tumeur tout en minimisant les dommages aux tissus sains environnants. Cependant, les mouvements internes, en particulier ceux liés à la respiration, peuvent altérer la forme, la position et la densité des organes, introduisant ainsi des erreurs et des incertitudes dans le positionnement de la dose de rayonnement. Cette problématique est particulièrement critique dans le traitement des tumeurs pulmonaires en raison du caractère chaotique et non reproductible des mouvements respiratoires.

La segmentation automatique consiste à identifier et délimiter précisément les structures anatomiques d'intérêt, en l'occurrence le système respiratoire, sur des images scanner des poumons. Cela signifie que nous cherchons à différencier les poumons, ainsi que d'autres composants du système respiratoire du reste des tissus environnants. Cette opération est cruciale pour la planification précise des traitements en radiothérapie, car elle permet de visualiser et de cibler spécifiquement les tissus cancéreux tout en épargnant les tissus sains.

L'apprentissage profond est une technique de l'intelligence artificielle, qui consiste à entraîner un modèle de réseau de neurones informatiques dans le but de répondre à une problématique. Le réseau de neurones que nous avons besoin sera dit "convolutif", cet adjectif signifie qu'il permet de d'analyser les détails d'une image afin de récupérer les informations souhaitées.

Ce projet a pour but de segmenter automatiquement l'appareil respiratoire afin de répondre aux exigences de la radiothérapie externe, et de pouvoir rendre plus rapide et précise son exécution, cependant une partie du système respiratoire nous apporte certaines difficultés dans la réalisation de ce projet : le diaphragme, organe permettant la respiration. Nous allons donc voir toutes les étapes qui nous ont permis de mener à bien ce projet, mais également celles que nous pourrons envisager de faire dans le futur.

1. Segmentation manuelle du diaphragme

La première étape de ce projet consistait à obtenir une base de données d'images qui nous serviront à entraîner notre modèle de réseau de neurones. Pour ce faire nous avons eu besoin de radiographies de poumons, qui nous ont été fournies par notre encadrant sous forme de fichiers de format .raw et .mhd. Au total, nous avons eu en notre possession des radiographies de poumons de 3 patients, et pour chaque patient au temps T00 et au temps T50, qui correspondent à une certaine période de la respiration. Et pour segmenter le diaphragme, nous avons utilisé le logiciel ITK-SNAP.

ITK-SNAP est un logiciel interactif qui permet aux utilisateurs de naviguer dans des images médicales tridimensionnelles, de délimiter manuellement les régions anatomiques d'intérêt et d'effectuer une segmentation automatique des images. Très simple d'utilisation, elle nous a permis de récupérer le masque de segmentation du diaphragme de chaque radiographie 3D selon la profondeur de la vue.

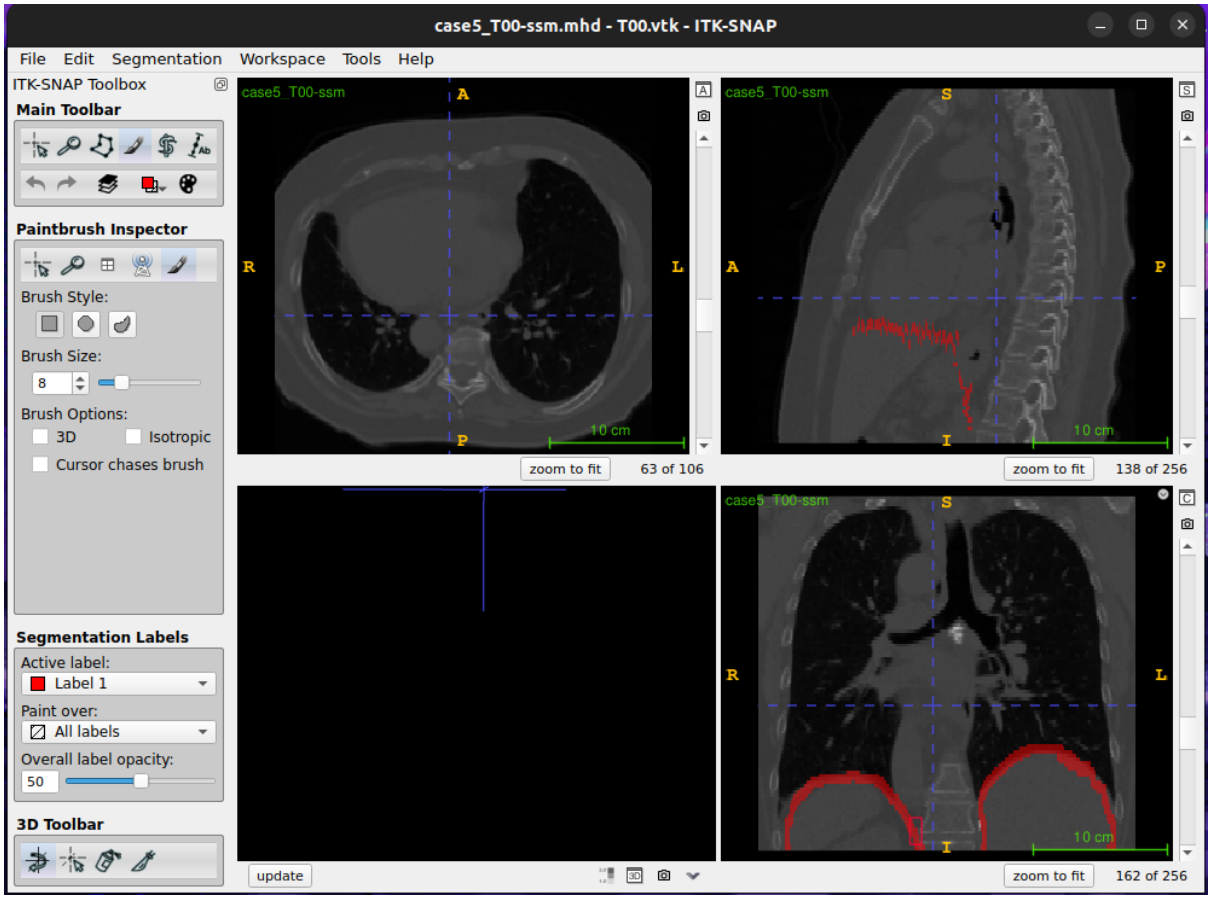

Figure 1 : Image du logiciel ITK-SNAP permettant la segmentation manuelle du diaphragme

Nous avons sélectionné le menu permettant de réaliser une segmentation, puis nous avons tout simplement tracé le diaphragme en rouge pour chacune des coupes de radiographie de poumons, nous avons ensuite répété cela pour chaque patient. Les masques de segmentation de chacune des radiographies sont ensuite conservés sous format .vtk.

2. Récupération des images et des masques

Après avoir segmenté manuellement toutes nos radiographies pulmonaires, il a donc fallu les récupérer pour en faire un dataset prêt pour l'entraînement. Pour cela nous avons créé un script python, qui, grâce aux librairies **Numpy**, **SimpleITK** et **vtk**, nous a permis d'obtenir dans un premier temps les radiographies et les masques sous forme d'un tableau 3D, puis nous les avons convertis en tableau 2D pour enfin les enregistrer en tant qu'image. Ces images sont stockées dans deux dossiers différents, **data_image** et **data_mask**, et chaque paire image/masque porte le même nom de fichier, cela permettra au programme créant notre dataset de bien repérer quel masque correspond à quelle image.

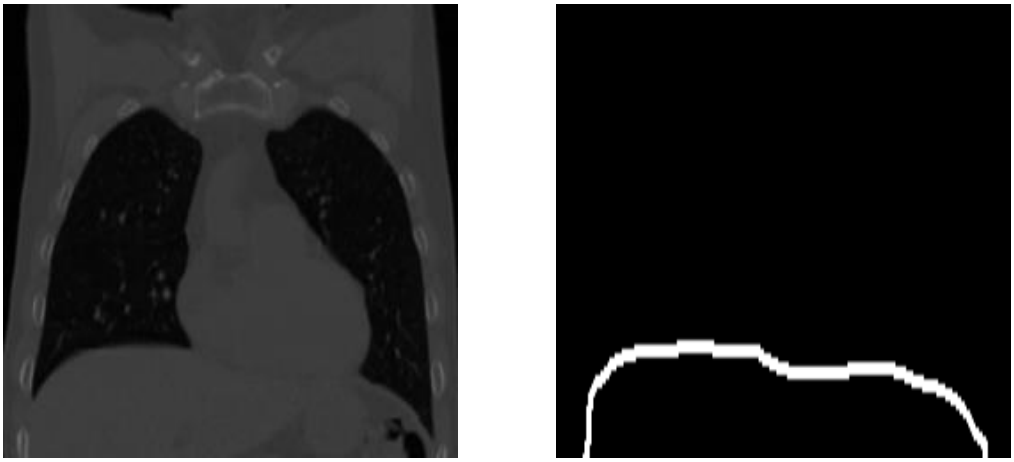

Figure 2 : Exemple d'image de poumons et de masque récupérées grâce à notre script python

Nous avons obtenu au total environ 1200 images de radiographies et de masques de segmentation associés à chaque image. Cependant cela n'était quand même pas assez pour réaliser un entraînement correct, nous avons donc fait un autre script permettant d'**augmenter** notre base d'images.

L'augmentation de données consiste à réaliser une légère modification sur une image, afin de la démultiplier sans modifier ses informations principales, nous pouvons par exemple ajouter de l'étirement, de la rotation, du zoom. Cette technique est efficace et permet d'augmenter la taille d'une base de données d'environ 3 fois.

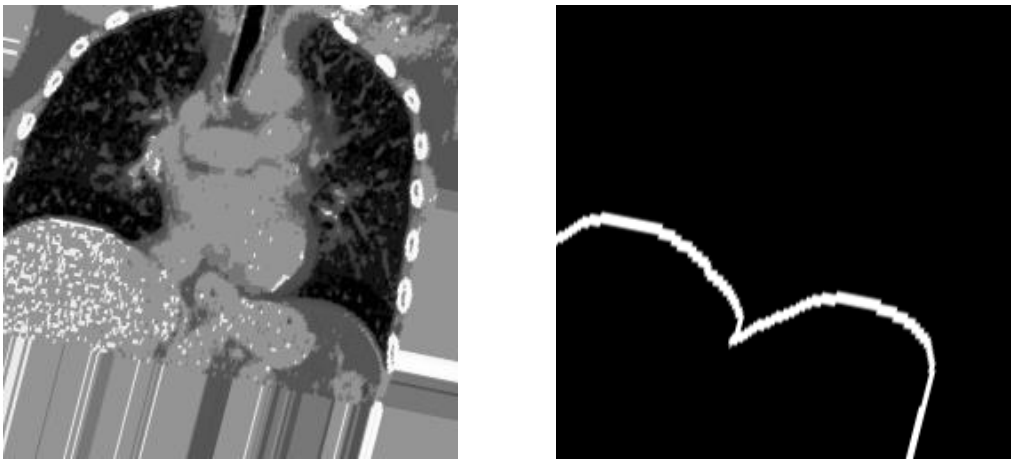

Figure 3 : Exemple d'image de poumons et de masque après la réalisation de la data augmentation

Grâce à la data augmentation, nous avons pu obtenir 4800 images supplémentaires, leur masque associé a subi a même modification que sa radiographie de poumons respective, l'information n'a donc pas changé.

3. Architecture de réseau de neurones convolutifs

Une fois la récupération et l'augmentation des images faite, la prochaine étape consiste à créer une architecture de réseau de neurones permettant de répondre à notre problématique.

Pour la réalisation de notre architecture de réseau de neurones, nous nous sommes mis d'accord dès le début qu'il nécessitera de faire du **transfer learning** pour obtenir des résultats concrets. Le transfer learning consiste à récupérer un modèle de réseau de neurones déjà entraîné, de retirer la dernière couche (car c'est celle qui correspond à la problématique de celui qui a créé ce modèle), et enfin d'ajouter nos propres couches, pour que le modèle puisse cette fois-ci répondre à notre problématique.

Quant à l'architecture pré-entraînée, nous avons utilisé **VGG16** en premier temps, puis **VGG19**, ces 2 algorithmes étant similaire, excepté leur nombre de couches de convolution, et ils ont été entraînés par des milliards d'images, cela nous donne la quasi-garantie que notre futur modèle sera plus efficace. Nous avons ensuite étendu l'algorithme afin d'obtenir une architecture similaire à celle de **U-Net**, reconnaissable grâce à sa forme en "U" comme nous pouvons le voir dans la figure ci-dessous.

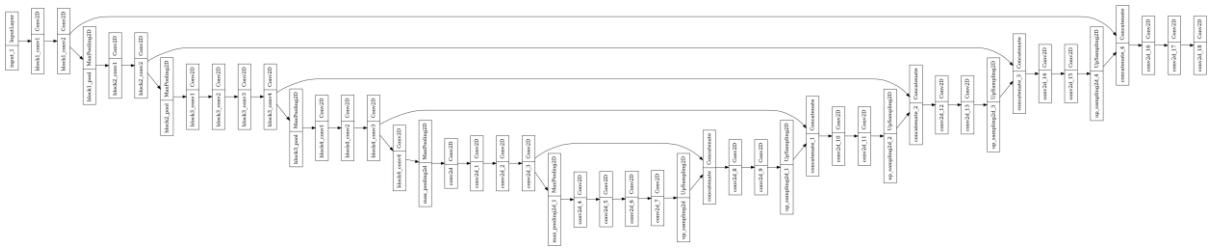

Figure 4 : Image (peu visible) de l'architecture complète utilisée pour ce projet

Cet algorithme contient 46 couches, et plus de 12 millions de paramètres, cependant, avant d'obtenir cette architecture finale, il a fallu réaliser un grand nombre de modifications, que ce soit au niveau de l'algorithme, mais également au niveau du pré-traitement des images de poumons.

4. Entraînements, tests, validations

A. Préparation des images

La phase d'entraînement s'est faite en plusieurs étapes, la première consistait à charger notre dataset d'images et de masques de segmentation. Pour cela, nous avons utilisé des fonctions de Tensorflow telles que **from_tensor_slices**, permettant de créer un dataset en mettant en paramètre les chemins de chaque image et de chaque masque, nous avons donc notre dataset qui est pour le moment composé de chaînes de caractères. Ensuite nous avons fait une fonction **charger_et_prétraiter**, qui charge chaque image à l'aide du chemin où elle se trouve, et qui réalise des opérations de normalisation sur ces dernières, comme la division par 255,0. Cette opération très souvent réalisée en apprentissage profond permet d'échelonner les pixels de chaque image pour qu'ils aient une valeur comprise entre 0 et 1, rendant les calculs lors de l'apprentissage plus performants.

Après avoir réalisé quelques entraînement, nous avons décidé de revoir notre approche et de, cette fois-ci, ajouter l'égalisation des images dans notre fonction de prétraitement, cela permet de mieux gérer la répartition des pixels dans l'image, et par conséquent d'améliorer la qualité et le contraste de l'image. Grâce à cela, nous avons des images dont les parties ambiguës sont un peu mieux visibles pour l'humain, et par conséquent, le même phénomène sera observé avec l'ordinateur.

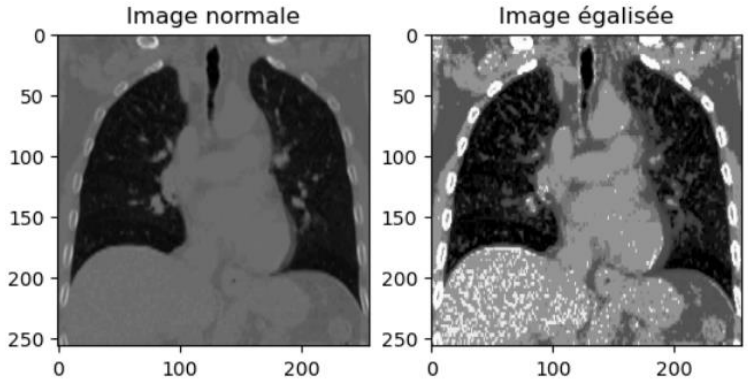

Figure 5 : Image de comparaison avant/après son égalisation

Les images sont toutes de tailles 256x256, les radiographies sont converties sous format de colorimétrie RGB pour répondre aux contraintes du modèle pré-entraîné, tandis que les masques de segmentation sont restés en nuances de gris.

B. Environnement d'entraînement

Nous avons vu dans les sections ci-dessus, que nos images sont de taille 256x256, et que notre architecture de réseau de neurones comporte un nombre conséquent de couches et de neurones. L'objectif de cette étape était donc de trouver un environnement d'entraînement performant pour que notre modèle puisse s'entraîner, car sur nos propres pcs, le terminal affiche "processus arrêté" dès que nous lançons l'entraînement, et ce même si nous réduisons la taille des images à 64x64, et lorsque cela fonctionnait, le temps d'entraînement était très élevé.

Une solution a été trouvée, nous avons utilisé les serveurs de calculs de Kaggle, qui offre un accès à ses GPU pouvant augmenter jusqu'à 12,5 fois la vitesse d'un entraînement. Bien-sûr cela a porté ses fruits, car nous avons réussi à entraîner notre modèle en environ 45 minutes, tandis que sur un pc classique, l'entraînement aurait duré plus d'une nuit.

L'entraînement était initialisé sur 50 epochs, avec un batch_size de 8 en prenant la métrique d'accuracy et une fonction de perte d'entropie croisée binaire *(binary crossentropy).* Les epochs correspondent au nombre de fois où l'entièreté de notre lot d'images pour l'entraînement sont entré dans le réseau de neurones, le batch_size correspond au nombre d'images parcourant le modèle avant que celui-ci ne se met à jour, la métrique calcule la précision en comparant le résultat obtenu pour une image, et celui qu'on est censé avoir, tandis que la fonction de perte observe les pixels dans le résultat qui sont égales à 0 ou 1 et les compares avec le masque fourni.

Nous avons également ajouté un earlysopper qui permet d'arrêter l'entrainement lorsqu'un certain nombre d'epochs sont écoulés sans que le pourcentage de perte se soit réduit, cela permet de prévenir **l'over-fitting**, qui survient lorsqu'un modèle est entraîné par le même type d'images, et qui n'arrive donc plus à généraliser ses prédictions.

L'entraînement s'est donc déroulé en 14 epochs, avec une précision avoisinant les 97%, ainsi qu'une perte d'environ 3%, ces résultats sont satisfaisants, cependant il faut voir ce que donne la courbe d'apprentissage pour pouvoir analyser de plus près l'efficacité de l'entraînement.

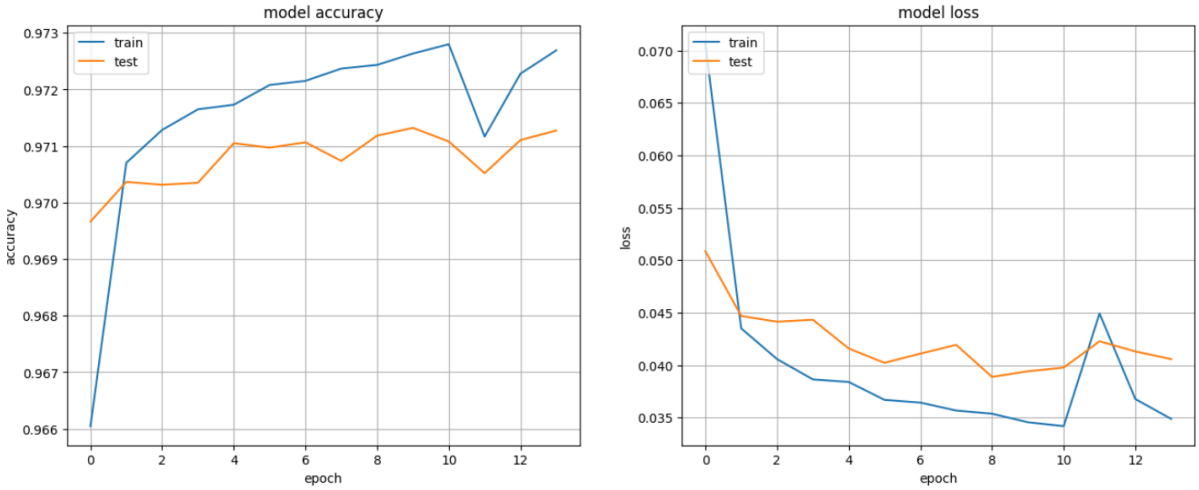

Figure 6 : Courbes d'apprentissage du modèle utilisé

Ces deux graphiques correspondent à l'évolution de la précision et de la perte au fil des epochs, la courbe bleue correspond au lot d'entraînement tandis que la courbe orange correspond au lot de validation. Pour faire simple nous pouvons d'ores et déjà dire que notre entraînement s'est bien passé car pour chacun des graphiques, les courbes bleues et orange restent assez proches, et cela est un signe indiquant que l'entraînement a été validé.

C. Evaluation

Nous avons également évalué le modèle après son entraînement, en calculant sa matrice de confusion grâce à la méthode **confusion_matrix** de scikit-learn et en réalisant un rapport de classification grâce à la méthode **classification_report**. Il s'agit de calculs entre le les masques dessinés manuellement, et ceux qui ont été prédits par le modèle. Les 2 valeurs à évaluer dans notre cas sont si le pixel prédit est un 0 (noir) ou un 1 (blanc), les pixels du tracé du diaphragme sont égaux à 1. Une matrice de confusion est une matrice 2x2 avec en haut à gauche les vrais négatifs (valeurs correctement prédites), en bas à gauche les faux positifs (valeurs normalement égales à 0 mais prédites comme un 1), en haut à droite les faux positifs (valeurs égales à 1 mais prédites comme un 0) et en bas à droite les vrais positifs (valeurs à 1 correctement prédites). Le modèle a été évalué avec 280 images.

```
Matrice de confusion:
[[17663760   140162]
 [  164259   381899]]
```

Figure 7 : Matrice de confusion de l'évaluation du modèle

Nous pouvons donc remarquer que 17663760 pixels noirs et 381899 pixels blancs ont été correctement prédits, 164259 pixels blancs alors qu'ils sont noirs de base, et 140162 pixels noirs qui devaient normalement être blancs. Regardons ensuite le rapport de classification.

Rapport de classification:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	17803922
1	0.73	0.70	0.72	546158
accuracy			0.98	18350080
macro avg	0.86	0.85	0.85	18350080
weighted avg	0.98	0.98	0.98	18350080

Figure 8 : Rapport de classification de l'évaluation du modèle

Dans ce tableau, seuls les 2 premières lignes et les 3 premières colonnes nous intéressent. Les lignes correspondent aux pixels noirs et blancs, pour la précision, sur tous pixels dont le modèle prévoyait qu'ils allaient être blancs, **73%** l'ont effectivement été. Pour le rappel (*recall*), sur tous les pixels blancs des masques segmentés manuellement, le modèle a prédit ce résultat pour **70%** d'entre eux. Quant au score-F1, cette valeur est calculée comme ceci : Score F1 = 2* (Précision * Rappel) / (Précision + Rappel) <=> 2 * (0.73 * 0.70) / (0.73 + 0.70), ce qui donne 0.72 de note-F1. Plus le score-F1 est proche de 1, plus le modèle est efficace car c'est ce qui indique que le modèle prédit correctement ce qu'on lui demande. Dans notre cas, 72% de score-F1 est vraiment loin d'être mauvais, car le tracé du diaphragme reste quand même visible s'il manque quelques pixels blancs, les zones "d'hésitation" sont visibles lors des tests, mais cela n'influe pas sur le résultat final.

D. Phase de tests et validations

Après chaque entraînement de modèle, il a fallu tester les résultats sur quelques images pour être convaincu que l'entraînement a été efficace. Pour cela, nous avons chargé nos images de test, prétraité ces images en les égalisant et en les mettant à un format compatible à la couche d'entrée du modèle (taille 256x256, format RGB) et puis nous avons utilisé la méthode **predict** du modèle pour obtenir un résultat. Ce résultat a donc été affiché pour valider l'entraînement.

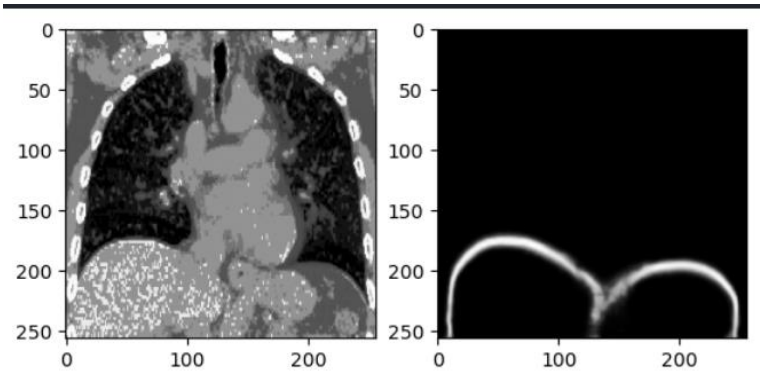

Figure 9 : Image de radiographie de poumons (à gauche) suivi du masque de segmentation tracé automatiquement (à droite)

Au total, nous avons entraîné près d'une dizaine de modèles pour que nous puissions arriver à ce résultat, le modèle reste toutefois perfectible, mais il nous permet de segmenter les zones ambiguës

du diaphragme, que même l'homme peut difficilement distinguer. L'ajout de l'égalisation des images dans le pré-traitement a non seulement permis de rendre plus efficace notre modèle, mais il nous a également permis de constater qu'il nous manque encore beaucoup d'images d'entraînement pour que nous nous permettons de faire varier les contrastes, bien que nous ayons près de 4000 images.

5. Conclusion

Pour conclure, ce projet nous a permis de découvrir le monde de la recherche en nous posant une problématique concrète et ambitieuse, et le fait de pouvoir contribuer à la science a été une nouvelle source de motivation. De plus, les résultats obtenus nous ont rendus optimiste sur la potentielle suite que peut prendre ce projet, le diaphragme pouvant être correctement segmenté, il ne nous restera plus qu'à étendre notre algorithme pour qu'il puisse segmenter automatiquement l'ensemble de l'appareil respiratoire. Et si ce projet arrive à son terme, le travail des professionnels luttant contre le cancer des poumons deviendra plus efficace, car les mouvements respiratoires deviendront prévisibles, et il en sera par conséquent de même pour les tumeurs se déplaçant dans les poumons. Cela nous montre que l'informatique peut contribuer à l'amélioration des autres domaines, notamment grâce à l'intelligence artificielle, qui depuis ces dernières années connaît une fulgurante expansion.

6. Références

CHEN YILONG, WANG KAI, LIAO XIANGYUN, QIAN YINLING, WANG QIONG, YUAN ZHIYONG, HENG PHENG-ANN (2019) - *Channel-Unet: A Spatial Channel-Wise Convolutional Neural Network for Liver and Tumors Segmentation*
https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2019.01110

Tensorflow Documentation en Python – Bibliothèque d'apprentissage profond
https://www.tensorflow.org/api_docs/python/tf/all_symbols

ITK-SNAP – Logiciel de traitement d'images médicales
http://www.itksnap.org/pmwiki/pmwiki.php