Projet M1 2021 : Apprentissage profond pour la caractérisation de l'hétérogénéité du mélanome

Lisa TOUZARI
Master ISI
Student number 28617271

Delphine DOUTSAS
Master ISI
Student number 3671238

***Abstract*—En France, les mélanomes constituent entre 2% et 3% des cancers détectés chaque année. Depuis les années 1960, son taux d'incidence augmente de 10% tous les ans. Détecté tôt, ce cancer a un très bon taux de remission (plus de 90% [1] [2]) c'est pourquoi un diagnostic précoce est important. Son diagnostic repose sur l'observation annuelle d'un dermatologue et, si la lésion apparaît suspecte, l'analyse microscopique d'un prélèvement. C'est en vertu de cela qu'a été initié un projet de recherche visant à segmenter les différents types cellulaires présents sur l'image numérisée des lames grâce à l'intelligence artificielle. En particulier, ce travail de recherche s'appuie sur les réseaux de neurones ayant montré leur efficacité dans les problèmes de segmentation. Notre projet, qui est l'héritage du premier élève ayant travaillé dessus, vise à améliorer les défauts du premier réseau *U-Net* en testant l'architecture *MultiResUNet*. Notamment, nous nous concentrons sur les problèmes de classification dus aux éventuels plis présents sur les lames.**

À l'attention du correcteur, l'annexe ne fait pas partie du rapport. Elle est destinée à M. Racoceanu et son équipe dans le dessein de faciliter la prise en main du serveur pour la prochaine personne en charge de ce projet.

INTRODUCTION

En France, les mélanomes constituent entre 2% et 3% des cancers détectés chaque année [1]. Ce type de cancer fait partie de la 9ème cause de cancer chez les femmes et 11ème cause chez les hommes [1] [2]. Bien que n'étant pas dans le top 5 des cancers les plus fréquents, son incidence a augmenté de 10% par an depuis les années 1960 [1]. Détecté tôt, ce cancer a un très bon pronostic [1] [2]. La détection s'effectue par l'observation d'un dermatologue et peut-être complétée par l'analyse d'un prélèvement si la lésion pigmentée paraît suspecte. Notre projet consiste à utiliser la puissance des réseaux de neurones de l'intelligence artificielle afin d'analyser ces lames de prélèvement et d'y détecter les élèments qui y sont présents : cellules tumorales, adipocytes, vaisseaux sanguins, etc. Ce projet a été commencé par un étudiant de l'ENS Lyon que nous devons reprendre et prolonger. Nous héritons donc d'un programme ayant implémenté l'architecture U-Net afin de classifier les types cellulaires présents sur les lames [3]. Cependant, la matière à étudier et insérer dans les lames n'est pas toujours correctement applatie ce qui peut favoriser la formation de plis. La classification en est donc rendue plus difficile puisque les cellules graisseuses sont confondues avec les éventuels plis présents sur la lame. **L'objectif de notre projet sera donc de réduire ce problème et de mesurer l'efficacité de la solution apportée en comparaison de celle de l'élève précédent.** Pour cela, nous nous sommes tournées vers une amélioration de l'architecture U-Net : MultiRes-Net.

Notre rapport s'articulera en plusieurs parties. Dans un premier temps, nous présenterons l'algorithme d'origine dont la classification est effectuée par le réseau U-Net. Dans une seconde partie, nous présenterons les résultats et les analyserons. Enfin dans les deux dernières parties, nous présenterons notre nouvelle architecture MultiRes-Net et nous en analyserons les résultats. Nous conclurons sur son efficacité.

I. PRÉSENTATION DE L'ALGORITHME ORIGINEL ET BREF HISTORIQUE DU PROJET

Initialement, le projet consistait à réaliser la classification des différents types cellulaires grâce au réseau de convolution *U-Net* [3]. Le premier élève sur ce projet l'a donc implémenté et a également réalisé un pré-traitement sur les images avant d'entraîner le modèle *U-Net* afin d'augmenter les performances du classificateur [3]. Nous avons dû dans un premier temps, avant toutes considérations de résolution de la problématique, comprendre le but et la démarche de l'élève avant nous et nous approprier son code.
Dans cette partie, nous présentons le programme ainsi que le fonctionnement spécifique d'un réseau de neurones dont l'architecture est *U-Net*.

A. Les fichiers et fonctionnement global du code [3]

Le programme est assez conséquent donc les différentes tâches de l'algorithme (apprentissage, classification, etc) ont été séparées en plusieurs fichiers. L'élève précédent a décidé d'utiliser la programmation orientée objet. Chaque classe a une fonction précise.
Le fonctionnement des classes et de certains algorithmes étant expliqué dans le dossier `Details_algo` fait par l'ancien élève et fourni avec le code, nous n'allons pas nous étendre dans un souci de concision. Voici un bref résumé de l'organisation des fichiers présents dans le dossier uniquement :

- *learning.py* : `learning` entraîne le modèle. Il charge les données de manière appropriée pour le modèle, initialise le modèle et lance l'apprentissage.
- *main.py* : Le `main` prend en entrée une image de la base de données de test et donne en sortie des masques qui une fois assemblés reforment l'image où tous les pixels sont classés. L'analyse des résultats (matrice de confusion, ...) est aussi donnée.
- *make_dataset.py* : Ce fichier effectue le chargement des images des lames afin de constituer la base de données.
- *Normalize_process.py* : `Normalize_process` est, comme son nom l'indique, le fichier normalisant les images (normalisation de Reinhard) dans le but de perfectionner la classification.

Il y a 5 classes en tout numérotées de la façon suivante : 0 - ROI (region of interest, tous les pixels que l'intelligence artificielle prendra en compte pour l'apprentissage), 1 - négatif (dont on est sûr que le pixel n'appartient à aucune classe) , 2 - pixel d'un vaisseau sanguin, 3 - pixel d'un adipocyte, 4 - pixel de l'épiderme, 5 - pixel d'une cellule tumorale.

B. Méthode de reproduction des résultats

N'ayant aucune expertise en biologie, nous avons dû ré-utiliser les lames dont les éléments d'intérêt avaient déjà été segmentés. Nous sommes de ce fait malheuresement en possession de deux lames : la lame 432 que nous avons utilisée pour la phase d'apprentissage et la lame 056 que nous utilisons dans la phase de test. La lame 056 est la seule des deux à présenter des plis d'où notre choix de la garder lors cette dernière phase.

C. Explication du modèle U-Net

U-Net est un réseau de neurones convolutionnels particulièrement adapté lorsque l'on recherche à segmenter et classer les parties d'une image [4]. Le but de cette architecture est à la fois d'augmenter les performances de classification avec une base de données de taille réduite tout en augmentant la vitesse d'exécution du programme [4].

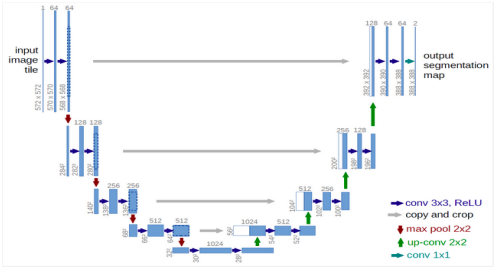

Fig. 1. Architecture du réseau U-Net [4]

Il possède une architecture en forme de U (figure 1) partageable en 4 parties : la couche d'entrée, la partie "descendante" nommée *U-Net down*, la partie "ascendante" dite *U-Net up* et la couche de sortie. Ces différentes parties ont des rôles bien particuliers qui, couplés à un "*raccourci*", chemin direct reliant de manière symétrique un bloc de la partie descendante à la partie ascendante, atteignent une plus grande efficacité de segmentation et classification des pixels que les réseaux "fully convultionnels" précédents [4].
La couche d'entrée réalise plusieurs convolutions sur le tenseur ce qui augmente le nombre de matrices - donc la première dimension du tenseur - et offre un nombre plus élevé de poids à optimiser. Dans son algorithme, l'élève nous précédant l'a programmé exactemment comme une couche descendante [3].
Les couches descendantes, *U-Net down*, opèrent sur le tenseur une convolution ainsi que l'application de trois fonctions : Drop-out, LeakyRelu et instance2Dnorm. Un passage par une couche descendante multiplie le nombre de matrice par 2 et divise la taille de chaque image par deux dans le but d'augmenter la résolution de l'information et de conserver les informations sur les caractéristiques. Cependant, à chaque diminution de taille, les informations de localisations sont perdues [4].
Les couches ascendantes sont constituées de couches de convolution qui procèdent au sur-échantillonnage de chacunes des images et applique également trois fonctions : Drop-out, Relu et Instance2Dnorm. Elle procède à l'inverse des opérations de la couche descendante donc à une multiplication de la taille des images et à une diminution du nombre de matrice. Chaque bloc du réseau *U-Net up* reçoit en entrée les résultats du bloc précédant concaténé avec la sortie du bloc *U-Net down* symétrique apporté par le *raccourci*.

Comme nous avons pu l'observer, dans son programme, l'élève nous précédant a implémenté l'architecture *U-Net* sous forme de classes. L'utilisation de la Programmation Orientée Objet (POO) nous a paru très adaptée et nous avons continuer de programmer en gardant ce paradigme.

L'élève a amélioré son algorithme au fur et à mesure [3]. Au départ, il a programmé *U-Net* et a, lors de la classification manuelle des lames de la base d'entraînement, utilisé des rectangles. Par la suite, il s'est mis à utiliser des polygones et à augmenter la base de données en ajoutant les mêmes images auxquelles sont appliquées des isométries. Enfin, il a appliqué la normalisation de Reinhard au préalable sur les images de lames et a annoté les parties dont il était sûr qu'elles étaient négatives. Cette normalisation est utilisée pour homogénéiser les contrastes et couleurs entre les images de la base de données d'apprentissage et de la base de test. Nous ne la détaillerons cependant pas plus, celle-ci étant déjà exposée de manière précise dans son rapport (voir référence [3]).
Les résultats obtenus se sont ainsi progressivement affinés [3].

II. Analyse des résultats

Nous allons maintenant présenter puis analyser les résultats obtenus par l'algorithme dont nous avons hérité afin d'ensuite les comparer à l'architecture du *MutliRes-Net* dans une prochaine partie.

Pour faire tourner ce programme de taille conséquente, nous avons eu recours au serveur de l'université de Montpellier 2 dont l'accès nous a été donné par notre maître de Projet, M. Racoceanu. Son fonctionnement est expliqué en Annexe.

Dans l'optique d'évaluer l'efficacité de l'algorithme, nous avons décidé de conserver les indicateurs de l'élève initialement utilisés par l'élève avant nous. En effet, notre projet consiste en partie à évaluer l'efficacité du réseau de neurones *MultiresUnet* par rapport à *U-Net*.
Les indicateurs sont :

- **la matrice de confusion** est définie de la manière suivante : sur les lignes se trouvent les classes réelles C_i de chacun des pixels tandis que dans les colonnes se trouvent les classes prédites C_j. La valeur dans la matrice, par exemple dans la case (i, j) représente le nombre de pixels dont la prédiction les range dans la classe C_i et dont la classe réelle est C_j La matrice est diagonale dans le cas d'une prédiction juste à 100%.

- **la matrice de précision** sert à calculer la précision pour chacune des classes. Notons C_i la classe de prédiction et C_j la classe réelle. La précision $P(i, j)$ se calcule ainsi et sera placée en (i, j) dans la matrice :

$$P(i,j) = \sum_{i=1,j=1}^{n,n} (\frac{|C_i \cap C_j|}{|C_j|})$$

  En (i, j), cette matrice nous montre le rapport entre le nombre d'élément correctement classés sur le nombre d'éléments de la classe de prédiction C_i.
- **la matrice de rappel** est calculée de manière analogue à la matrice de précision à la différence que le dénominateur devient le cardinal de l'ensemble C_i :

$$R(i,j) = \sum_{i=1,j=1}^{n,n} (\frac{|C_i \cap C_j|}{|C_i|})$$

  En (i, j) se trouve la rapport entre le nombre d'éléments correctement rangés sur le nombre d'éléments de la classe réelle d'apparatenance C_j.
- **la matrice F-β** où $\beta = 1$ est une matrice indiquant la moyenne harmonique pour chacune des classes. Ses coefficients (i,j) se calculent comme suit :

$$F_1(i,j) = \frac{2 \cdot P(i,j) \cdot R(i,j)}{P(i,j) + R(i,j)}$$

  Dans le cas d'une bonne classification, les coefficients diagonaux de la matrice F_1 doivent être proches de 1.

La génération d'une base de données exploitables par l'algorithme durant plus de 4 jours et l'apprentissage prenant sûrement autant de jours, nous n'avons pas encore pu reproduire les résultats. Les algorithmes sont actuellement en train de tourner sur le serveur et nous sommes en attente des résultats. Le serveur n'a pas assez d'espace mémoire pour appliquer la normalisation de ReinHard et stocker la base de données sur l'espace alloué à notre maître de stage. Nous nous attendons donc à des résultats de classification moins bons que ceux de l'étudiant avant nous. Cependant, c'est à partir d'eux que nous comparerons notre classificateur et qu'ainsi nous quantifierons l'amélioration. Cela ne posera donc pas de problème particulier lorsque nous procéderons à la comparaison.

III. Méthode de résolution de la problématique

Dans cette section, nous allons justifier nos différentes approches de résolution *MultiResUNet*, présenter les caractéristiques de son architecture et expliquer rapidement les choix de programmation faits.

Au début du projet, pour résoudre la problématique, nous avions essayé de créer une classe "pli". Cependant, nous nous sommes vite aperçues que cette méthode de résolution n'était pas bonne. En effet, nous ne sommes pas des expertes et notre classification manuelle de la lame à l'aide du logiciel de segmentation *Calopix* n'aurait pas été précise. Ce n'était pas une méthode de résolution rigoureuse. C'est en partant de ce constat que nous avons plutôt pris la décision de trouver une solution dont l'objectif serait d'augmenter l'efficacité de la classification globalement. Nous nous sommes donc dirigées vers l'implémentation du réseau de neurone *MultiResUnet*.
Ne pouvant pas résoudre directement le problème, nous avons cependant décider de segmenter grossièrement (avec nos compétences de profanes) à l'aide du logiciel Calopix les plis et d'observer le pourcentage de classification pour tout de même conserver un indicateur vis-à-vis de ce problème. Nous détaillerons mieux notre procédure dans la prochaine partie (Analyse des résultats).

A. Méthode d'acquisition des résultats

Nous allons ici expliquer les étapes que nous avons effectuées pour obtenir des résultats.

Nous avons en premier lieu eu besoin de faire appel à un serveur (celui de Montpellier 2) pour faire tourner notre programme et ainsi effectuer un nombre conséquent d'itérations (plus de 12 000 rien que pour générer un dataset exploitable). Nous l'avons configuré et installé tous les modules et librairies nécessaires. Dans un second temps, nous avons généré un fichier `dataset.p` exploitable par les algorithmes grâce au programme `make_dataset.py`. Dans un troisième temps, (nous y sommes toujours à l'heure où nous remettons ce rapport) nous avons lancé les procédures

d'apprentissage par le fichier `learning.py`. Enfin, dans un dernier temps, nous laisserons tourner le fichier `main.py`, appelant *MultiResUNet* et *U-Net*, sur le serveur dans l'optique d'obtenir nos résultats et de calculer nos indicateurs.

B. Amélioration globale de l'architecture : MultiResUNet [5]

En vue d'améliorer l'efficacité globale de classification, nous avons programmé l'architecture *MultiResUNet* dont nous allons présenter le fonctionnement ci-dessous.
MultiRes-Net est une architecture de réseau de neurones qui vise à surpasser *U-Net* [5]. Son but est de rendre moins chronophages les opérations de convolutions présentes dans l'architecture de *U-Net* et de rendre plus précise la classification [5].

À la manière de *U-Net*, *MultiResUNet* est composée uniquement de couches convolutionnelles et est construit de manière symétrique avec une branche descendante *"Encoder"* et une branche ascendante *"Decoder"* et un *"MultiResPath"*.

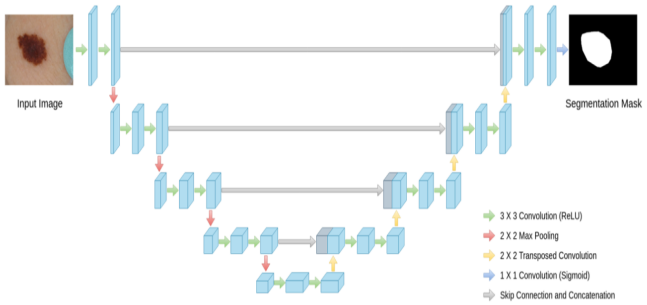

Fig. 2. Architecture du réseau MultiResUNet [5]

Pour constituer ces branches, on crée au préalable ce que nous nous avons appelé des "*blocs U-Net*" qui comprendront des convolutions 3x3. À chaque sortie d'une convolution, le résultat est conservé et est concaténé avec les autres résultats obtenus par les autres convolutions du bloc. Il est important ici de comprendre que plus l'on avance dans les convolutions 3x3 plus le nombre de filtre augmente. Une "*connection résiduelle*" (la convolution 1x1) est aussi présente pour garder une trace des matrices du tenseur avant convolution. On somme enfin les matrices concaténées avec le résulat de la *connection résiduelle*. Le schéma ci-dessous illustre le fonctionnement d'un *bloc U-Net*.
Afin de passer d'un réseau *U-Net* à un réseau *MultiResUNet*, il faut déterminer le nombre de filtre équivalent. On le calcule grâce à un paramètre α comme suit $W = \alpha \cdot U$ où W est le nombre de filtre dans un bloc (que l'on verra en dessous), U est le nombre de filtre sur la couche correspondante (au même niveau) de *U-Net* et α est un scalaire que l'on doit trouver. Dans notre cas, nous avons trouvé $\alpha = 1.67$
L'idée derrière l'architecture des *blocs U-Net* a été de réduire la quantité de mémoire utilisée tout en gardant les propriétés d'extraction des caractéristiques des images du

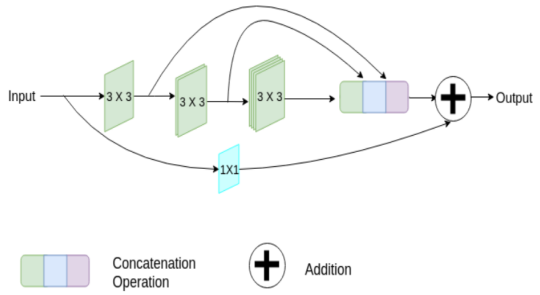

Fig. 3. Architecture d'un *bloc U-Net* [5]

tenseur [5].

Sur la branche "*Encoder*", en sortie de chacun des *blocs U-Net*, on applique des opérations de *max-pooling* (c'est notre fonction `forward` dans le code). Cela divise les deux dernières dimensions du tenseur par 2.
Sur la branche "*Decoder*", en sortie de chacun des *blocs U-Net*, on réalise une opération de convolution transposée - ce qui a pour effet de doubler les deux dernières dimensions du tenseurs - tout en concaténant la sortie du *ResPath* provenant de la couche "symétrique" avec le tenseur.
Le *ResPath* est en quelque sorte un raccourci dans le réseau de neurones. Il offre la possibilité de connecter deux couches que l'on dira symétriques, dont les tenseurs, en sortie du bloc de la branche "*Encoder*" et en entrée du bloc de celle de "*Decoder*", sont de même taille. Ce raccourci permet la fusion des caractéristiques spatiales et des caractéristiques de contexte présentes dans les images contenues dans le tenseur [5]. Le *ResPath* n'est pas qu'un simple raccourci dans l'architecture ; il comprend aussi des opérations et de ce fait constitue lui-même une séquence de couches convolutionnelles. Il applique sur le tenseur en sortie d'un bloc de la couche "*Encoder*" des convolutions 3x3 et 1x1. Leur nombre varie suivant la profondeur du réseau de neurones : plus on descend dans les blocs et plus le nombre de convolutions diminue. La contrainte étant de s'assurer que l'avant-dernière couche la plus profonde ne comporte un *ResPath* qu'avec une seule convolution. On concatène ensuite le résultat de cette convolution au tenseur en entrée du bloc correspondant sur la couche "*Decoder*".

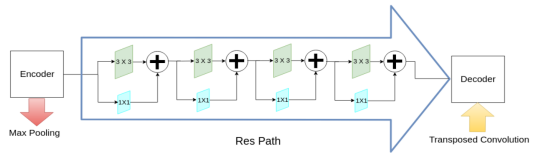

Fig. 4. Architecture du *ResPath du réseau MultiResUNet* [5]

En ce qui concerne l'implémentation du *MultiResUNet*, comme cela a été dit plus haut, nous avons décidé d'avoirs recours à la programmation orientée objet pour construire l'architecture. Cela nous assure à la fois la compatibilité entre

objets et la conservation de l'esprit du projet fait par l'élève nous précédant. Nous nous sommes inspirées de deux codes présents en ligne sur *Github* : celui de l'équipe à l'origine du *MultiResUNet* [6] et d'une autre équipe l'ayant implémenté avec l'environnement *PyTorch* [7].

IV. Analyse des résultats

Pour analyser les résultats de performance globale et obtenir une comparaison intéressante, nous avons repris exactement les mêmes indicateurs que dans la partie II du rapport en ce qui concerne l'efficacité globale du programme : c'est-à-dire la matrice de confusion, la matrice de précision, la matrice de rappel et la matrice F_β.

Nos programmes sont actuellement en train de tourner sur le serveur. Leur exécution prennant plusieurs jours, nous n'avons malheureusement pas la possibilité de présenter nos résultats ici. Cependant dès qu'ils seront finis, nous actualiseront ce rapport.
Nous nous attendons à une meilleure performance du classificateur que dans le cas de *U-Net* ; la matrice de confusion devrait être plus proche d'une matrice diagonale que dans le cas de *U-Net*, c'est-à-dire avec une Trace de la matrice de confusion, une acurracy plus élevée et la matrice F_β avec des coefficients diagonaux plus proches de 1.

Dans le but de mesurer l'erreur de classification due aux plis, nous avons eu l'idée de nouveaux indicateurs. La zone du pli ne devant pas être classée car on ne peut pas reconnaître une superposition de cellules, nous calculons le pourcentage de mauvaise assignation de classe aux pixels dans le pli.
On peut avoir deux sortes d'indicateurs :

- Le pourcentage de mauvaise classification θ_i de la classe C_i dans les plis :
  Soit P l'ensemble des pixels appartenant au(x) pli(s), soit M la matrice telle que l'on ait 0 en l'élément (i, j) si le pixel (i, j) de l'image n'appartient pas au(x) pli(s) présent(s) sur la lame et 1 si le pixel (i, j) appartient au(x) pli(s), soit l'ensemble $C = \{C_i | i \in [\![0, 5]\!]\}$ où C_i est la classe i contenant tous les pixels qui lui ont été attribués lors de la phase de classification de l'image.
  $$\theta_i = \frac{|C_i \cap P|}{\sum_{i,j=1,1}^{n,n} M_{i,j}}$$
- Le pourcentage Θ de mauvaise classification global dans les plis :
  $$\Theta = \sum_{i=0}^{n=5} \theta_i$$
  où 5 est le nombre de classes.

Cependant, ayant passé beaucoup de temps à configurer le serveur et à programmer *MultiResUNet* et nos programmes mettant plusieurs jours à s'exécuter, nous n'avons pas eu le temps d'implémenter cet indicateur ; nous avons préféré implémenter une amélioration plus globale. Sa programmation est donc laissée comme perspective d'amélioration au prochain élève en charge de ce projet.

De plus, nous sommes conscientes que la classification est très dépendante de la segmentation manuelle des lames. En effet, en nous intéressant à l'état de l'art du domaine de la segmentation en intelligence artificielle, nous avons pu comprendre que la variation lors de la segmentation pour un même expert pouvait varier de 15% à 20% [8]. De ce fait, bien que nous tentions d'améliorer la classification, elle finira -en l'état actuel de la recherche [8]- par être limitée par l'étape de la segmentation manuelle.

Conclusion

En conclusion, nous avons réussi à nous approprier le code et à comprendre les tenants et les aboutissants du projet initial. Nous avons ensuite exploré plusieurs pistes d'améliorations du projet possibles et avons décidé d'essayer de résoudre les erreurs de classifications inhérentes aux éventuels plis présents sur les lames en effectuant une amélioration globale de la classification. Nous avons également réussi à nous approprier les divers outils mis à notre disposition pour mener à bien le projet. Ainsi, nous avons réussi après beaucoup d'acharnement à configurer le serveur et à faire tourner nos codes dessus. Nous avons également réussi à mettre en place des indicateurs pour mesurer nos résultats. Cependant, dû au temps considérable que prend l'exécution des programmes (plusieurs jours) nous ne sommes pas encore en capacité de fournir des réultats. À l'heure actuelle, nous sommes dans la phase d'apprentissage de *U-Net* et *MultiResUNet*. Comme dit précedemment, nous actualiserons le rapport dès que possible. Enfin, si nous avions eu encore plus de temps, nous aurions essayé de programmer les indicateurs concernant directement la mauvaise classification des pixels sur les plis et nous aurions généré une base de données plus grande encore.

References

[1] A. Morin and C. Alliot et al, "Actualisation de la revue de la littérature d'une recommandation en santé publique sur la "détection précoce du mélanome cutané"," Haute Autorité de la Santé (HAS), Tech. Rep., juillet 2012. [Online]. Available: https://www.has-sante.fr/upload/docs/application/pdf/2013-01/synthese_de_la_recommandation_actualisation_de_la_revue_de_la_litterature_dune_recommandation_en_sante_publique_sur_la_detection_precoce_du_melanome_cutane.pdf.

[2] A.-V. Guizard, A.-S. Woronoff, and S. Plouvier et al, "Survie des personnes atteintes de cancer en france métropolitaine 1989-2018, Mélanome Cutané," Santé Publique France, report, avril 2021. [Online]. Available: https://www.santepubliquefrance.fr/docs/survie-des-personnes-atteintes-de-cancer-en-france-metropolitaine-1989-2018-melanome-de-la-peau.

[3] K. Mohammed-Amine, "Combinaison de l'intelligence artificielle et l'analyse numérique pour la modélisation et la caractérisation du mélanome - Rapport du stage," M.S. thesis, École Centrale de Lyon, 2020-2021.

[4] R. Olaf, F. Philipp, and B. Thomas, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597.

[5] N. Ibtehaz and M. S. Rahman, "MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation," *CoRR*, vol. abs/1902.04049, 2019. arXiv: 1902.04049. [Online]. Available: http://arxiv.org/abs/1902.04049.

[6] Ibtehaz, Nabil and Rahman, M Sohel, *MultiResUNet code (2D Model)*, https://github.com/nibtehaz/MultiResUNet, Accessed: 2018-06-07, 2019.

[7] j-sripad, *Pytorch implementation of multiresunet : Rethinking the u-net architecture for multimodal biomedical image segmentation*, https://github.com/j-sripad/mulitresunet-pytorch, Accessed: 2021-06-10, 2021.

[8] Z. Akkus, A. Galimzianova, and A. Hoogi et al., "Deep Learning for Brain MRI Segmentation: State of the Art and Future Directions," *Journal of Digital Imaging*, vol. 30, pp. 449–459, 2017. DOI: https://doi.org/10.1007/s10278-017-9983-4. [Online]. Available: https://link.springer.com/article/10.1007/s10278-017-9983-4.

APPENDIX

Guide d'utilisation du serveur de Montpellier 2

Afin de correctement utiliser le serveur, plusieurs étapes de configuration et de déplacement de dossiers sont nécéssaires.

1) **Accès au serveur**

- **Par le terminal**
  Que ce soit pour Windows ou Linux, le serveur est accessible depuis le terminal de n'importe quel ordinateur. Cependant, pour Mac, il est nécessaire d'installer au préalable le logiciel **xQuartz** qui donnera accès au serveur. Nous conseillons à l'utilisateur de Mac d'utiliser ensuite le Terminal (toujours sans désinstaller xQuartz).

- **Par l'interface graphique Filezilla**
  Une interface graphique peut s'avérer pratique pour envoyer ses fichiers et dossiers mais aussi visualiser l'arborescence du serveur. L'interface privilégiée pour tous les systèmes d'exploitation est **Filezilla**.

- **Étape d'identification**
  Afin d'accéder au serveur, il est obligatoire de passer par une étape d'identification et ce, peu importe la nature de l'interface utilisée.

```
ssh -X <nom et première lettre du prénom collée>@muse-login.hpc-lr.univ-montp2.fr
```

Fig. 5. Commande d'accès au serveur par le terminal

Pour y accéder depuis FileZilla, vous aurez besoin du nom du cluster, de l'identifiant utilisateur, de son mot de passe. Le port à rentrer est 22.

2) **Déplacement de fichier et dossier vers le serveur**

- **Droit d'accès**
  Dans le serveur, n'est accessible que ce qui est au nom d'utilisateur (en général du professeur donnant les codes d'accès), les fichiers et dossiers à importer doivent donc l'être dans les dossiers du professeur. Deux branches se distinguent : celle de `/home` dont l'espace de stockage alloué par personne est plus réduit mais permanent et `/lustre` dans lequel l'espace alloué est plus grand mais où tout document entreposé n'y reste que deux mois.

- **Quelques fonctions de l'interface**
  L'interface offre en plus de la visualisation de l'arborescence, la possibilité de charger des fichiers et des dossiers facilement. Elle est constituée de deux fenêtres : celle de gauche est l'arborescence de l'ordinateur courant et celle de droite l'arborescence du serveur.
  Pour envoyer un fichier ou dossier de son propre ordinateur au serveur, il suffit de se placer à l'endroit d'emplacement désiré dans le côté droit de l'interface. Puis une fois cette action effectuée, il faut dans la partie gauche, cliquer droit sur le fichier ou dossier à transmettre et cliquer sur "Envoyer".

3) **Configuration de l'environnement d'exécution**

- **Installation de logiciel**
  L'installation de logiciel sur le serveur est possible à condition d'avoir les droits d'autorisation de l'administrateur réseau.
  Une fois que cela est garantie, il suffit d'aller chercher la commande en ligne par exemple sur `GitHub`.
  Voici un exemple de commande à entrer pour le logiciel **OpenSlide** :

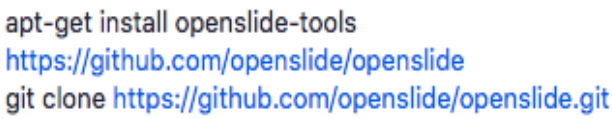

Fig. 6. Commande d'installation de **OpenSlide**

- **Installation de bibliothèques sur Python sur le serveur**
  Pour installer une bibliothèque *Python*, deux méthodes sont possibles en fonction des modalités d'installation souhaitées : ou bien, on souhaite travailler directement avec *Python* et dans ce cas la commande est
  `pip install <nom du module>`
  ou bien on désire travailler depuis *Anaconda* et dans ce cas on utilise la commande
  `conda install <nom du module>`.
  Pour installer avec `pip`, il est important de rentrer la commande `module load python<version>`.

- **Vérification de la présence d'un module, d'une bibliothèque, etc.**
  Pour vérifier la bonne installation d'un module, il suffit de taper la commande suivante dans le terminal :
  `module avail`

4) **Scripts de lancement d'un programme**

- **Rédaction du fichier bash**
  Le fichier bash est le fichier initialisant l'exécution d'un programme en language *C*, *C++*, *Python*, etc. Il doit impérativement commencer par la ligne suivante :
  `#!/bin/sh`
  Voyons un exemple que nous détaillerons plus en détails en dessous :

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 20
#SBATCH --account=<nom de
l'account>
#SBATCH --partition=<nom de la
partition>

module load openslide/3.4.1
module load openjpeg/2.4.0
module load python/3.8.2
python make_dataset.py
```

  Chacune des lignes correspondant aux paramètres d'éxecution doit commencer par `#SBATCH`. -N fixe le nombre de tâches à exécuter en parallèle, -n règle le nombre de coeurs. Ensuite, pour exécuter un programme python, il suffit d'ajouter une ligne avec dessus `python <nom du programme>`.
  Si l'on souhaite préciser la version d'un module en particulier, il faut taper la ligne suivante : `module load <module>` comme dans le fichier ci-dessus.
  Pour éditer le fichier, il faut se placer dans le dossier où se trouve le programme et créer un fichier bash avec la commande suivante :
  `nano <nom du fichier>.sh`
  et ensuite faire ctrl-x (ou cmd-x) pour sortir du fichier en enregistrant les modifications.

- **Exécution, suivi d'exécution et gestion des erreurs du programme**
  En se plaçant dans le dossier contenant le fichier bash et en tapant la commande suivante :
  `sbatch <nom du fichier bash>.sh`
  le fichier devrait s'exécuter.
  Pour ensuite vérifier sa place dans la file d'attente des tâches lancées sur le serveur, il suffit de taper dans le terminal :
  `squeue`
  Une fois que le fichier a disparu de la file d'attente, il y a deux possibilités. Soit le programme s'est exécuté sans erreur, dans ce cas on obtient le fruit du programme (par exemple une image dans le dossier voulue), soit le fichier a rencontré une erreur et dans ce cas un fichier slurm.out se trouve dans le même dossier que le fichier bash. Dans ce cas, il faut consulter son contenu avec la commande suivante :
  `nano slurm.out`
  et alors, l'erreur d'exécution y sera notée.