Stage chez Cardiff University Brain Research Imaging Center

Tristan Renaud
4A Génie biomédical
Année universitaire 2021/2022

Tuteur laboratoire : Pr. Mara Cercignani

Tuteur école : Dr. Carine Guivier-Curien

Remerciements

Pour commencer je voudrais remercier **CUBRIC** pour leur accueil chaleureux au sein de leur équipe. Je remercie toutes les personnes qui ont contribué à la réussite de ce stage, notamment les personnes suivantes :

Mara CERCIGNANI, pour l'intérêt qu'elle a porté à mon profil mais également pour sa disponibilité, son accompagnement et ses conseils tout au long du stage.

Matteo MANCINI, pour m'avoir accompagné dans toutes les étapes de mon projet, mais également pour sa disponibilité et sa bonne humeur.

Cariad SEALAY, pour son aide continu pour les recours auprès de l'équipe informatique et de l'équipe administrative.

Carine GUIVIER-CURIEN, ma tutrice pédagogique, pour son soutien constant.

Et enfin tous mes collègues avec qui j'ai pu discuter et apprendre durant ce stage.

Sommaire

Abstract

MRI is a primary imaging technique for the study of the brain. Its ability to differentiate between soft tissues makes it a key technology for understanding brain functions, both healthy and pathological. A lot of acquisition parameters are adjustable : ones could get an image focused on anatomy or tissue function, removing signal from various tissues etc.

The downside of MRI is its high cost and long acquisition time. Moreover, research teams in psychology, cognitive sciences (etc.) usually need MRI images different than those usually in daily clinical practice at hospital. This is an obstacle for those researchers who definitely could get used of the MRI images acquired in numbers at hospital.

One of the solutions to these issues would be to compute different MRI images (T1-, T2- weighted, FLAIR, MWF, diffusion...) from only a few acquisitions. My project aimed to develop a deep learning based algorithm capable of producing a T2-weighted image and a MWF (Myelin Water Fraction) image from a single T1-weighted image. By leveraging convolutional neural networks (CNNs), we achieved promising results in image reconstruction, potentially reducing the need for multiple MRI scans and enhancing clinical efficiency.

This work did not aim to create the most capable CNN architecture. Therefore, a handmade and quite simple architecture was employed using a downscaling part followed by an upscaling one. Future works could explore more advanced architectures such as a generative adversarial network like pix2pix, for instance. Also, the database used included healthy patients only. Assessing the performance of such a neural network on pathological brains would be necessary.

1 Cardiff University Brain Research Imaging Center

1.1 Présentation générale

CUBRIC est un centre de recherche à portée internationale situé et appartenant au campus de l'Université de Cardiff. Il est dirigé par le Pr. Dereck et le parc d'IRM est dirigé par le Pr. Mara Cercignani. Près de 200 chercheurs spécialisés en psychologie, neurosciences, électronique, IRM etc. y travaillent quotidiennement. De plus, le centre dispose de 4 IRM : une IRM 7T, 2 IRM 3T et une IRM 3T connectome. L'ensemble de ces IRM peut être utilisé en parallèle de stimulation magnétique transcrânienne ou d'électroencéphalographie.

1.2 Axes de recherche

Il y existe un nombre très important d'axes de recherche comme la recherche sur les interactions entre les différentes aires cérébrales pour mieux comprendre leurs impacts sur les comportements.

Un autre axe de recherche qui occupe énormément de chercheurs et dont mon projet a fait partie est le développement de nouvelles méthodes pour quantifier les différents tissus à l'échelle microscopique. Cela implique en grande partie l'utilisation de l'IRM à diffusion qui permet de mesurer indirectement les flux d'eau dans le cerveau qui reflètent plus ou moins la nature des tissus.

Un dernier exemple de recherche en cours que j'ai trouvé particulièrement intéressant est le travail avec la société Hyperfine. Il a pour but de développer de petits IRMs portatifs à faibles champs magnétique (mT) pour permettre un coût d'achat réduit et donc permettre de démocratiser son utilisation. Ce ne sont que quelques exemples parmi un large éventail de projets en cours.

1.3 Ambiance de travail

Par ailleurs, on m'a fait remarquer un très fort développement du télétravail à la suite de la pandémie du COVID-19 et les personnes ont du mal à revenir travailler sur site. De ce fait, le bâtiment reste assez peu foisonnant malgré les près de 200 chercheurs. L'ambiance y reste très agréable avec des personnes de nombreuses origines géographiques (allemands, italiens, espagnols, chinois, français, américains...) ce qui permet un enrichissement culturel très intéressant lors des moments de regroupement.

En effet, toutes les 2 à 3 semaines, un quizz est organisé, basé sur des thèmes propres au laboratoire mais appliqués à la pop culture etc. Cela permet d'échanger facilement avec toutes les personnes, peu importe leur statut ou d'où elles viennent. A la suite de ce quizz, une partie des participants se rejoignent dans un pub à proximité, pour faire vivre la culture anglaise !

2 Projet

2.1 Présentation du projet

Le projet sur lequel j'ai travaillé pourrait s'intituler succinctement : "Estimation d'Imagerie par raisonnance magnétique (IRM) quantitative à partir de scanner de routine à travers un réseau de neurones profond". Plus explicitement, mes tuteurs travaillent activement sur l'IRM à diffusion pour permettre d'analyser quantitativement les tissus au sein du cerveau. Cela peut, par exemple, être la quantité de myéline par voxel estimée grâce au ratio Myelin Water Fraction (MWF). C'est-à-dire le ratio entre l'aire de la réponse en T2 à un temps court (20ms) sur l'aire de la réponse totale en T2. En effet, la fraction d'eau présente entre les différentes couches de myéline ne présente que très peu de flux et donc son temps de relaxation en T2 est très court. Cependant, ce type d'analyse quantitative nécessite des séquences d'IRM particulières ainsi que du post-traitement important. C'est pour cela que le Dr. Mancini m'a proposé d'essayer de mettre au point un réseau de neurones profond afin de pouvoir estimer une image MWF à partir d'une image faite en routine à l'hôpital (T1-MPRAGE, par exemple).

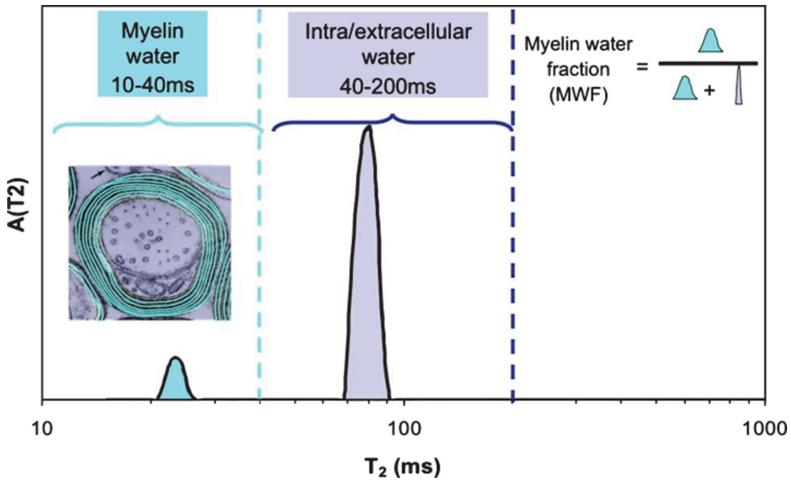

FIGURE 1 – Illustration du ratio MWF

Source : MacKay AL, Laule C. Magnetic Resonance of Myelin Water : An in vivo Marker for Myelin. DOI : 10.3233/bpl-160033.

2.2 Analyse de la littérature et première approche des images IRM

Dans un premier temps, afin de pouvoir comprendre au mieux les images auxquelles j'allais faire face, la Pr. Cercignani m'a prêté le livre "Quantitative MRI of the Brain Principles of Physical Measurement, Second edition" qu'elle a co-écrit. Il comprend près de 400 pages détaillant les dernières techniques utilisées dans l'IRM quantitative. Elle m'a proposé de m'intéresser particulièrement aux mesures du T2, de la diffusion de l'eau et du transfert de magnétisation. A cette lecture, souvent complexe car poussée jusque dans les équations d'analyse des ondes radiofréquence, le Dr. Mancini m'a proposé de commencer à travailler sur des images IRM. Ces deux exercices m'ont permis, conjointement, de pouvoir aborder plus sereinement la suite du projet.

2.2.1 Référentiel des images

Un consensus international a été créé concernant la neuroimagerie. Ainsi, toutes les images d'IRM se présentent sous le format .nifti qui permet de stocker l'image 3D mais également d'autres informations comme le plan de référence de l'IRM, le plan utilisé lors de l'imagerie (tête tournée, par exemple) etc. La librairie Nibabel permet d'importer facilement ces fichiers dans un notebook python afin d'avoir accès à l'ensemble des données.

Concrètement, sur le début, j'ai pu importer deux images d'IRM pondérées T1. Ces deux images ont été prises sur deux sujets différents donc, comme on peut le voir, pour les mêmes coordonnées, les deux coupes ne correspondent pas du tout.

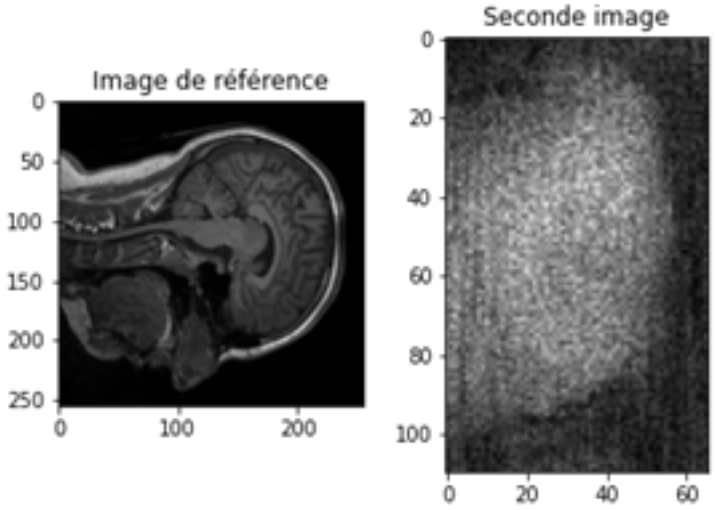

FIGURE 2 – Coupe 2D avant changement de plan

Pour y remédier, j'ai écrit un algorithme qui, étant donnée la matrice de transformation par rapport au plan de référence du scanner, change les coordonnées (i,j,k) de chaque pixel afin de pouvoir obtenir deux images dans le même plan. En effet, on a la matrice affine A présente dans le fichier .nifti telle que :

$$A = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & a \\ m_{2,1} & m_{2,2} & m_{2,3} & b \\ m_{3,1} & m_{3,2} & m_{3,3} & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

avec $m_{(i,j)}$ les composantes de la matrice de rotation M et a,b,c les composantes de la

matrice de translation. On peut ainsi obtenir, dans notre nouveau plan (x, y, z), :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Ainsi, nous obtenons deux images dans le même plan avec les différentes régions anatomiques qui correspondent pour chaque point. Cela permet donc de normaliser le plan de références des images obtenues entre différents patients mais également entre deux séquences d'un même patient dont la position n'aurait pas été exactement la même.

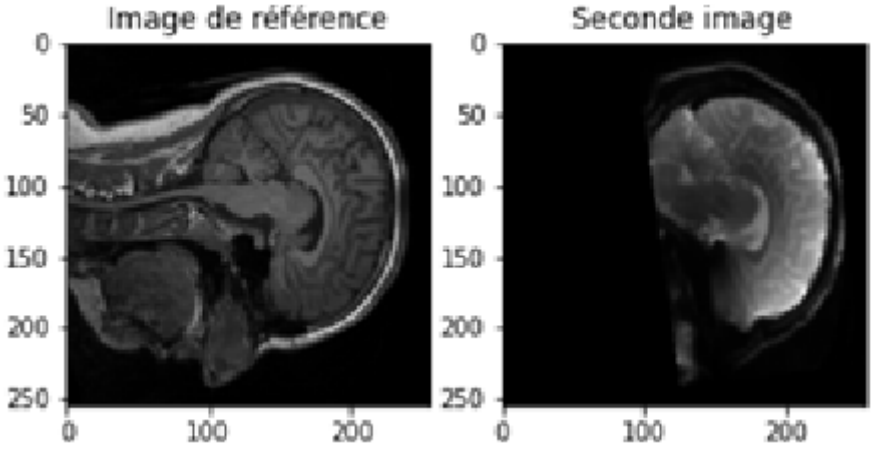

FIGURE 3 – Coupe 2D après changement de plan

2.2.2 Segmentation

Une fois cette normalisation effectuée, je me suis intéressé à la segmentation des différentes aires cérébrales :

— Le **Liquide cérébrospinal** (CSF).
— La **Matière grise** (GM).
— La **Matière blanche** (WM).

Le tout sur différentes séquences d'IRM :

— **T1map**, le temps de relaxation T1 par voxels.
— **T1w**, le temps de relaxation T1 par voxels.
— **MWF**, le ratio entre le temps de relaxation T2 de l'eau dans la myéline vs le temps de relaxation T2 total.
— **MTsat**, le transfert de magnétisation.

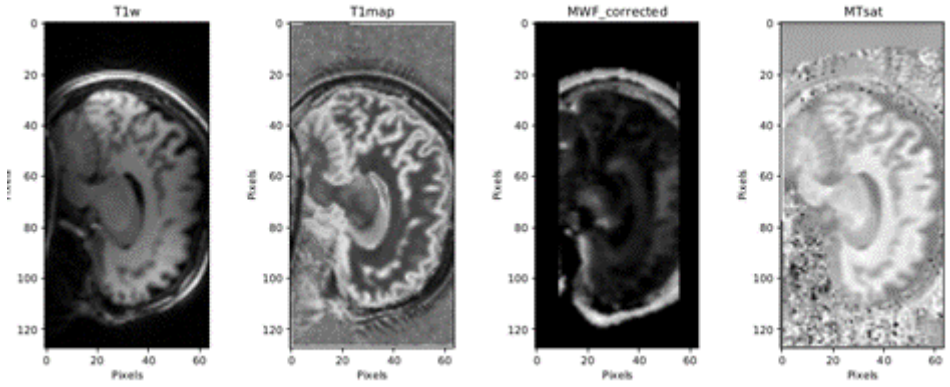

FIGURE 4 – Cerveau sous différentes séquences

Pour cela, différents outils utilisés par les chercheurs de CUBRIC permettent de faciliter les opérations. Ainsi, un premier algorithme, BET, m'a permis d'isoler uniquement les tissus de l'encéphale, en enlevant notamment la boîte crânienne mais aussi la dure mère etc. Dans un second temps, l'algorithme FAST m'a permis de segmenter les trois différents tissus. Sachant que nos images T1w, T1map, MWF et MTsat sont parfaitement alignées, il a suffi de faire cette opération sur T1w afin d'obtenir un masque des différents tissus à appliquer sur chaque image.

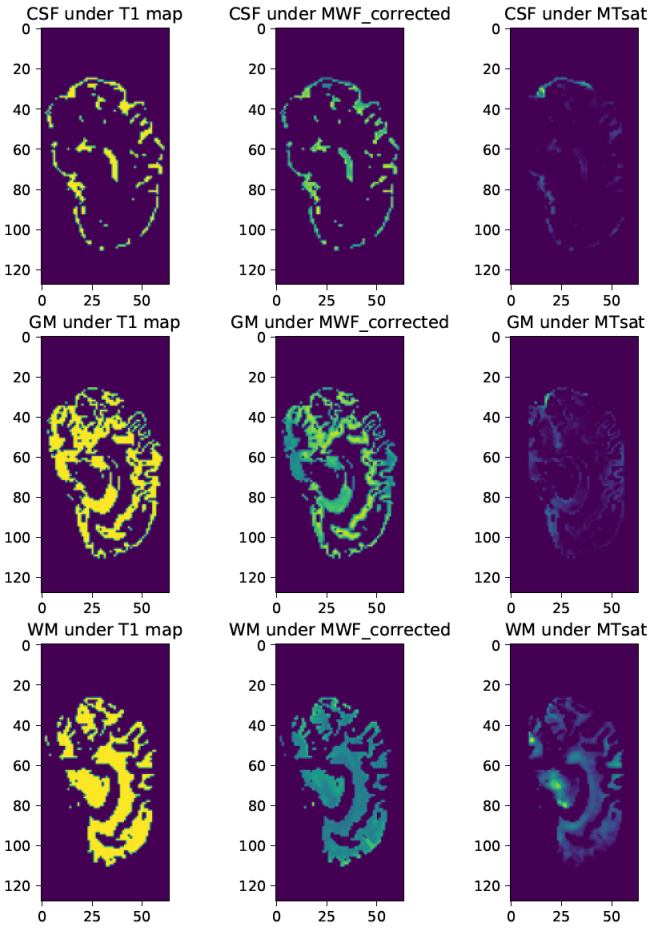

FIGURE 5 – Cerveau sous différentes séquences par type de tissu

De ce fait, j'ai pu obtenir, pour les 3 sujets, un histogramme de la répartition des valeurs de chaque pixel en fonction du tissu. On peut notamment observer un pic de la matière blanche autour de 750ms qui correspond bien au temps de relaxation de celle-ci. On peut aussi, par exemple, remarquer que l'aire sous la courbe pour la matière blanche est la plus élevée sur les images MWF : en effet, c'est bien celle-ci qui comporte le plus de myéline.

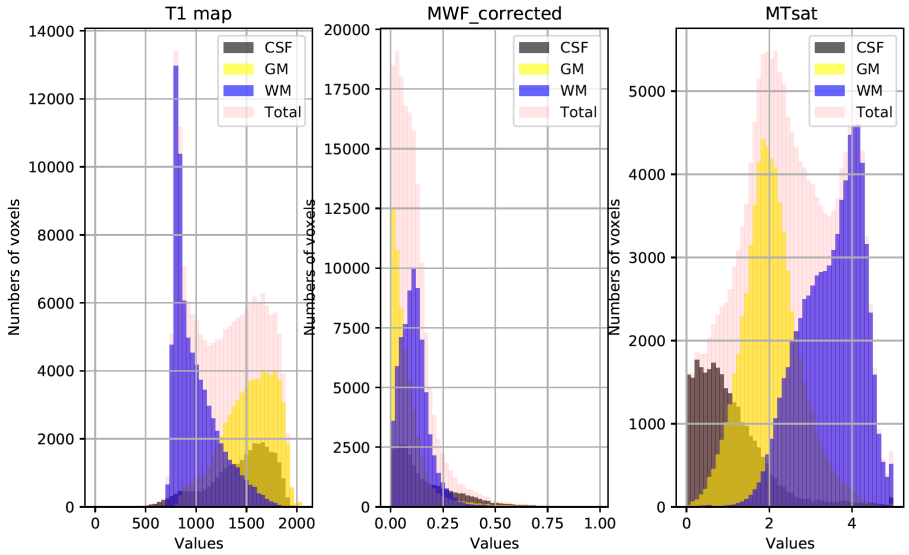

FIGURE 6 – Histogramme des tissus par séquence

2.3 Appréhension des réseaux de neurone

Après m'être familiarisé avec le fonctionnement des fichiers .nifti et des différents outils de segmentation à ma disposition ainsi que des langages de programmation python et bash, j'ai pu commencer à me plonger dans l'univers des réseaux de neurone.

2.3.1 Neuromatch Academy

Neuromatch Academy est une organisation de bénévoles qui mettent à disposition de nombreux cours en ligne sur les sciences de l'informatique. Parmi ces cours, mon tuteur m'a proposé de suivre ceux orientés sur le deep learning. J'ai, comme ça, pu commencer à appréhender le fonctionnement des réseaux de neurone profonds. Par profond on entend que le réseau est composé de plusieurs couches "cachées" auxquelles on n'a pas accès : on a juste une entrée et une sortie. Pour comprendre le fonctionnement à la base, j'ai commencé avec un seul neurone entre la sortie et l'entrée. Ce type de structure, très basique, m'a permis de comprendre le fonctionnement des poids (weight) ainsi que de certains hyperparamètres qui constituent un des points majeurs sur lesquels on modifie l'apprentissage d'un réseau.

Un réseau de neurone par apprentissage profond repose sur certains éléments de base :

— L'initialisation manuelle des **hyperparamètres** (epochs, learning rate, momentum etc.) et initialisation aléatoire des **poids**.

— Un **optimiseur (optimizer)**, par exemple l'algorithme de descente de gradient, qui vise à minimiser la fonction de perte à chaque itération.

— Le nombre de **couches de neurones** et le nombre de **neurones par couche**.

— Une **fonction d'activation**, par exemple la fonction ReLu (fonction rampe : $f(x) = max(0, x)$), qui définit la sortie d'un neurone.

Ainsi, pour un cas assez simple (par exemple pour une régression linéaire), on peut avoir un réseau assez simple de ce type :

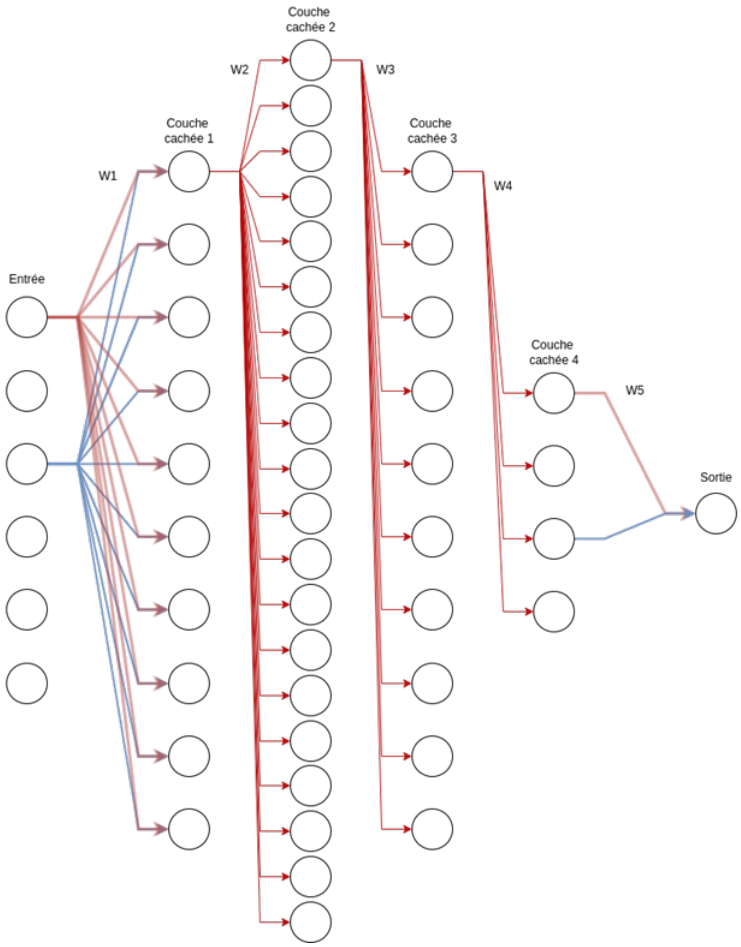

FIGURE 7 – Illustration d'un réseau de neurone connecté

On a un vecteur d'entrée correspondant aux 6 points dont on souhaite obtenir une régression linéaire. On a ensuite une succession de couches cachées et finalement la sortie a. Chaque cellule est interconnectée à chacune des cellules de la couche suivante. A chaque connexion est un associé un poids (weight), stockés dans une matrice W.

Ainsi, $W1$ sera une matrice de 6 lignes par 10 colonnes, soit 60 poids. Pour une couche entièrement connectée K, on aura, en entrée d'un neurone : $\sum_{j=0}^{W.len} w_{K,j}x_j + b_j$. Ce scalaire est ensuite passé à travers la fonction d'activation du neurone (pour la fonction ReLu, la sortie est égale à l'entrée si elle est positive sinon elle vaut 0). Dans cet exemple, la sortie finale vaudra donc la somme du produit $W5$ des 4 sorties des 4 neurones précédents par leur poids plus une constante B.

2.3.2 Réseaux de neurones par convolution

Après avoir compris les concepts autour des réseaux de neurones profonds, je me suis intéressé aux réseaux de neurones par convolution. En effet, dans le cadre de mon projet, je dois utiliser des images à travers un réseau de neurones. Notamment pouvoir entraîner un réseau de neurones à classifier le type d'images IRM en entrée (T1, MWF, MTsat). Pour cela, les réseaux de neurones par convolution sont particulièrement efficaces par l'apprentissage de filtres.

2.3.2.a Convolution

En effet, avant de passer notre image par les couches de neurones connectées, on va d'abord les faire passer par différentes étapes de convolution et sous-échantillonnage. Pour cela, on va définir des noyaux/filtre d'une certaine taille (généralement de 3x3 à 7x7). Pendant la phase d'apprentissage, le filtre sera convolué sur toute l'image le résultat étant le produit du filtre sur chaque segment de l'image. Plusieurs dizaines de filtres vont pouvoir être utilisés. Ainsi, au fil de l'apprentissage, chaque noyau deviendra spécifique à une information sur l'image (par exemple la présence de tracés diagonaux, verticaux ou alors des formes plus complexes). Il est important de noter que pour une image en entrée et 16 filtres, on aura l'équivalent de 16 images en entrée des couches connectées.

2.3.2.b Pooling

Or, une des problématiques des réseaux de neurone, c'est le nombre de paramètres (poids et biais). En effet, plus le nombre de paramètres est grand, plus le calcul de l'optimiseur sera long. Dans le cas des images, cela monte très vite. Les images d'IRM 2D que j'utilise après découpe par un script font une taille de 128*88 pixels. Avec l'ajout des filtres, on peut alors avoir en entrée 128*88*16 pixels, soit plus de 180.000 paramètres simplement à l'entrée du réseau. Le simple ajout d'une couche complètement connectée de 10.000 neurones amène à près d'un milliard le nombre de paramètres. Pour cela, les réseaux par convolution utilisent une seconde étape de pooling, qui va permettre de sous-échantillonner graduellement les images dans le réseau. Pour cela on définit la taille du noyau et le pas. Le pooling le plus utilisé est le maxpooling : sur une fenêtre donnée (noyau), on va sélectionner la valeur la plus élevée parmi les pixels. Pour une image de 128*88 à laquelle on applique un maxpooling d'une fenêtre de 2x2 et un pas de 2x2, la dimension est divisée par deux : 64*44. On peut appliquer le pooling à plusieurs moments (par exemple entre chaque étape de convolution) pour arriver à des images de taille très réduites.

2.3.2.c Augmentation

Un autre problème auquel on fait face dans un réseau de neurone par apprentissage, c'est que le modèle va apprendre sur des données qualitatives et calibrées. Or, dans la réalité, une image d'IRM peut se trouver différemment orientée, plus ou moins floue, avec peut-être même des zones noires dues à des artéfacts (composants ferromagnétiques dans les dents, dans du maquillage etc.). Pour cela, j'ai procédé à une étape d'augmentation des données. C'est-à-dire qu'avant de donner les images à mon modèle pour l'apprentissage, le programme va, aléatoirement :

— Plus ou moins **flouter** l'image.

— Effectuer une **rotation** et/ou **translation**.

— **Rogner** une partie de l'image.

De ce fait, le modèle obtenu sera plus résilient face à des images de la vraie vie.

2.3.3 Construction d'un premier modèle de classification

Pour construire notre modèle, on a donc à intégrer une section dédiée à l'augmentation puis créer le réseau de neurone par convolution en plaçant plusieurs couches de convolution et de pooling. Cette première étape permet de déterminer, au fil de l'apprentissage, des gradients permettant d'identifier certaines formes. Elles sont suivie par plusieurs couches de neurones complètement connectés. Ces couches permettent, à partir d'un ensemble de noyaux de convolution, de déterminer s'il s'agit d'une image pondérée T1, une image MWF ou une image MTsat.

2.3.3.a Augmentation

Comme expliqué plus haut, le première étape fut d'augmenter les données (en l'occurence des images d'IRM). Elles ont été floutées, rognées, translatées, tournées.

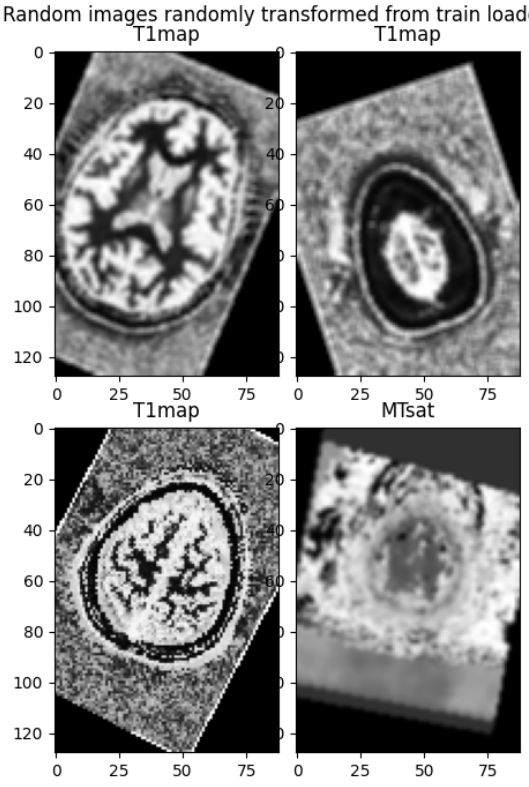

FIGURE 8 – Augmentation des images d'IRM

2.3.3.b Fonction d'entraînement

Une des étapes les plus importantes pour l'algorithme est de définir la fonction d'entraînement. Elle va permettre de passer notre ensemble de jeu de données dans

les fonctions d'optimisation et de perte. Le tout le nombre de fois qu'on aura défini (epochs). En fonction d'optimisation, j'ai choisi la fonction ADAM dont le calcul des poids évolue en fonction de l'apprentissage. Elle est reconnue pour son efficacité, son besoin en mémoire réduit et sa capacité à adapter les hyperparamètres au fil des epochs. Cela permet notamment d'éviter une partie d'overfitting. Ensuite, en fonction de perte, j'ai choisi la fonction de CrossEntropyLoss qui permet de calculer les probabilités d'appartenance à un ensemble de classes, ici : 0 = T1, 1 = MWF et 2 = MTsat.

2.3.3.c Réseau de neurones

Pour ce classifieur, j'ai ajouté 3 couches de convolution (succintement à 8, 16 et 32 noyaux). Après chaque couche, j'ai ajouté la fonction d'activation ReLu ainsi qu'un pooling permettant de divisier par deux le nombre de pixels après chaque. Cela permet d'obtenir en entrée des couches de neurones connectés un total de 4032 pixels contre près de 2 millions sans pooling.

2.3.3.d Entrainement

L'étape finale, et la plus longue, est de lancer l'entraînement du réseau. Avec un learning rate de 0.0005 et 14 epochs, cela a duré 30min (relativement rapide car nombre de données limité). Le résultat final nous donne un taux de réussite de 91%.

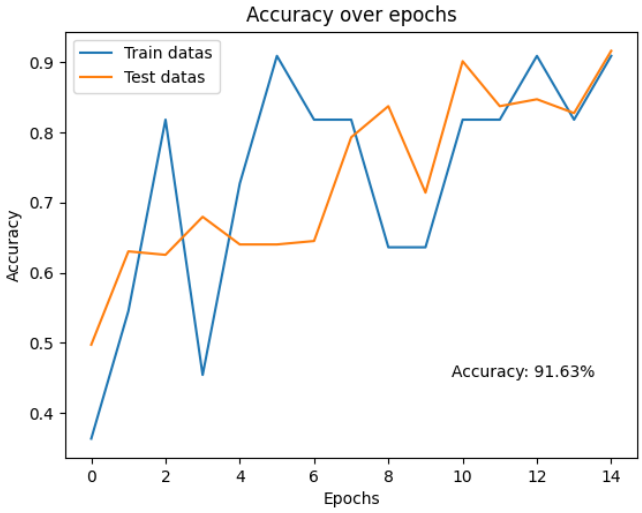

FIGURE 9 – Précision au cours des epochs

On peut calculer la matrice de confusion qui permet de déterminer les points faibles du réseau. Autrement dit, pour quels types d'image il se trompe.

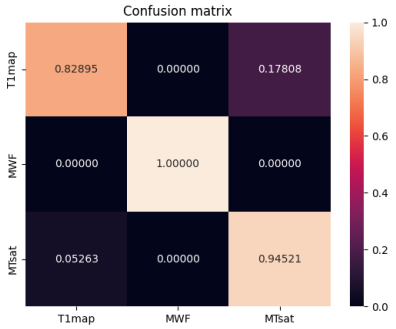

FIGURE 10 – Matrice de confusion

On remarque que le réseau reconnait 100% des images MWF tandis qu'il y a une erreur plus importante pour la détection des images T1 qui sont surtout confondues avec des images MTsat.

Finalement, le résultat de ce classifieur reste assez satisfaisant. En voici quelques échantillons pris au hasard :

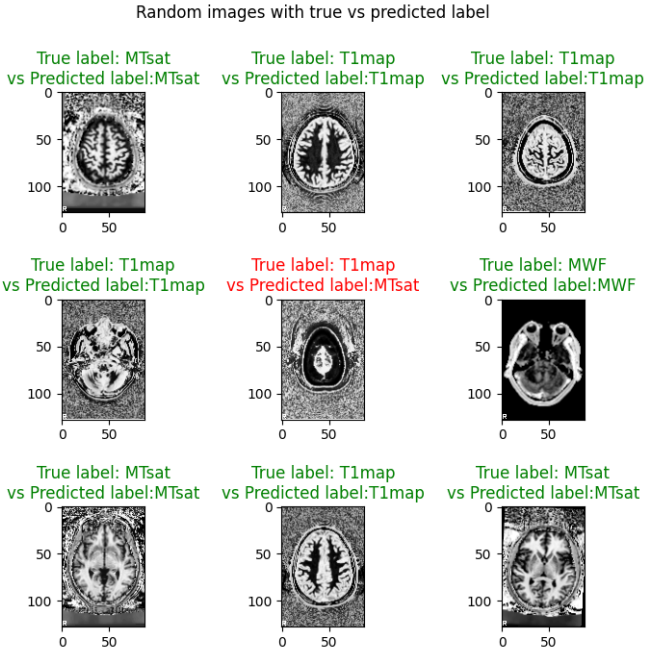

FIGURE 11 – Résultat de quelques classifications

Ce classifieur n'ayant eu pour objectif que de commencer à me familiariser avec les bilbiothèques de deep learning, je ne m'y suis pas attardé plus longtemps. J'ai ainsi pu commencer à m'orienter vers un réseau génératif par la suite.

2.4 Création d'un CNN reproduisant une image T2 à partir d'une image T1

A la suite du classifieur, je me suis lancé dans une tâche plus complexe : recréer une image pondérée T2 à partir d'une image pondérée T1. L'objectif étant de voir la faisabilité d'une telle technique pour pouvoir l'améliorer par la suite et peut-être, un jour, pouvoir l'utiliser en clinique.

Pour ce travail plus complexe, il m'a fallu accéder à un nombre beaucoup plus important de données. Pour cela, j'ai eu la chance de pouvoir accéder à la base de données WAND (Welsh Advanced Neuroimaging Database). Elle est constituée de neuroimageries de 127 participants à l'heure actuelle. Chaque participant a passé des imageries pondérées T1, T2, mais aussi MWF etc. avec des IRM de 3T et 7T.

2.4.1 Préparation des données

Tout d'abord, la première étape fut d'accéder aux dites données. Nous avons donc fait une réunion où nous avons présenté, avec le Dr. Mancini, notre projet aux responsables de WAND. Ils ont trouvé l'idée extrêmement intéressante et nous ont donné accès aux données dont nous avions besoin (imageries pondérées T1, T2 et MWF).

Une fois en possession des données, je les ai toutes alignées entre elles et créé 3 masques CSF, GM et WM à partir des images pondérées T1.

Les masques étant composées de 0 et 1, il m'a fallu normaliser les images pondérées T1 et T2 pour qu'elles se trouvent dans les mêmes valeurs. Pour cela, j'ai chargé l'entiereté du dataset de chaque type d'imagerie (T1 et T2) et, pour chacune j'ai récupéré la moyenne et l'écart type. Par la suite, il m'a suffi d'ajouter ces valeurs dans la fonction de normalisation au moment de la création de mes objets dataset.

2.4.2 Fonction d'entraînement

La fonction d'entraînement reste globalement similaire à celle du classifieur. J'ai utilisé le même optimiseur ADAM. En revanche, pour la fonction de perte j'ai utilisé la fonction MSEloss qui calcul l'erreur quadratique moyenne. Elle est beaucoup plus adaptée au calcul de l'erreur entre la vraie image T2 et celle prédite.

2.4.3 Réseau de neurone

Pour pouvoir recréer une image à partir d'une image en entrée, on n'utilise plus de couches de neurones connectés. En effet, on va seulement utiliser des couches de convolution successives de grandes dimensions. Une fonction d'activation sera placée entre chaque couche de convolution. J'ai donc utilisé 8 couches de convolution de 64 noyaux, dont une pour l'entrée et une pour la sortie.

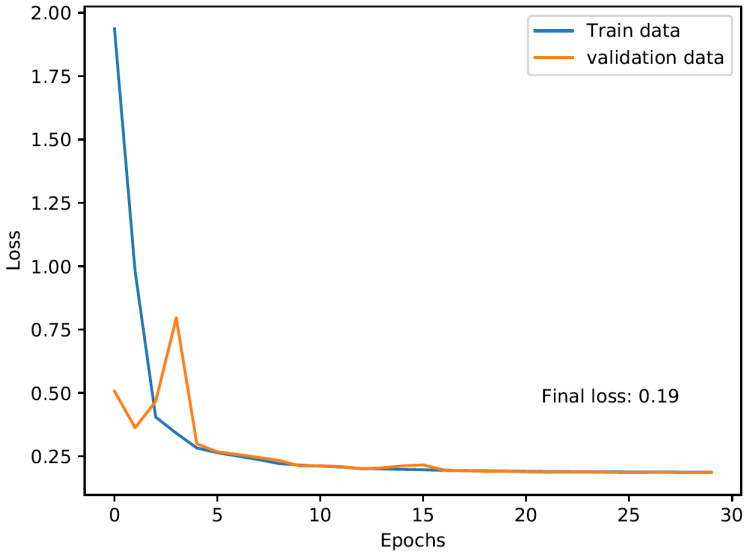

FIGURE 12 – Perte au cours des epochs

On peut remarquer un apprentissage très net sur les 5 premières epochs puis rapidement une stabilisation à partir de l'epoch 15. En l'état, il ne serait donc pas utile de jouer sur le nombre d'epochs pour espérer améliorer l'algorithme.

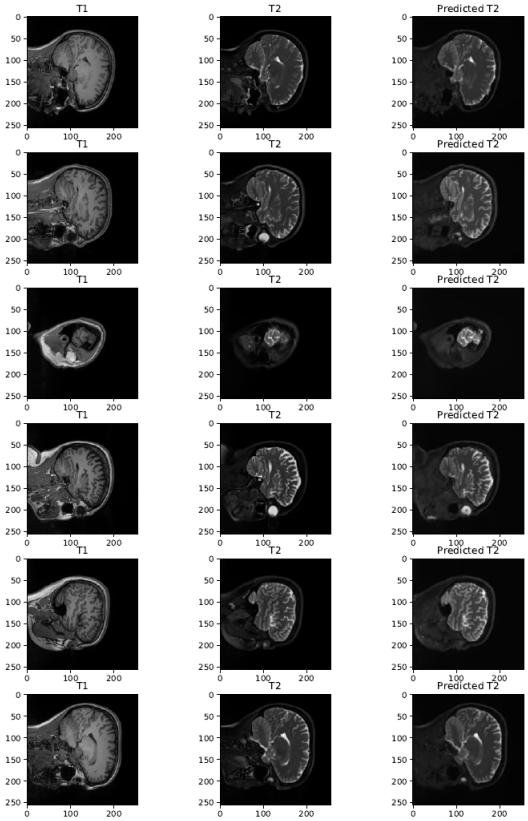

FIGURE 13 – Prédiction des images T2

On peut voir, à l'oeil nu, une ressemblance assez marquée entre les images T2 réelles et celles prédites.

Plus objectivement, on a une similarité moyenne de 86%.

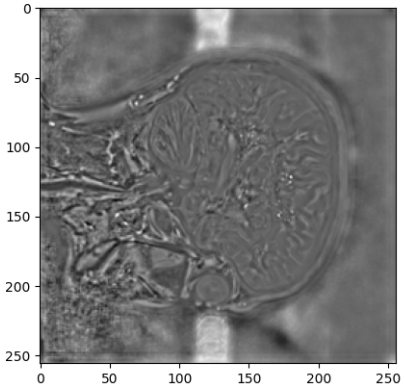

FIGURE 14 – Similarité des structures (T2)

Sur cette figure on peut observer que les points qui posent problème au niveau de la précision sont les parties de circonvolution mais aussi pour beaucoup les zones situées au niveau de la sphère ORL ce qui n'est pas le sujet de ce travail.

2.5 Création d'un CNN reproduisant une image MWF à partir d'images T1 & T2

Après avoir travaillé sur le CNN prédisant une image T2, j'ai fait de même pour reproduire une image MWF. En effet, c'est le principal objectif de mon projet. Les séquences d'IRM MWF sont très peu utilisées en clinique même si elles représentent un intéret certain. L'objectif est donc d'obtenir une image MWF avec un examen de routine en T1 & T2.

2.5.1 Réseau de neurones et données

Pour ce cas, les données étaient donc des images T1 & T2 que j'ai assemblées dans une image à deux channels. Cela fait, j'ai calculé la moyenne et l'écart-type de chaque dataset (train, validation & test) afin de les normaliser. L'entrée du CNN se retrouve être une image 3D composée de T1 et T2 et la sortie une image 2D MWF. Le réseau de neurones que j'ai utilisé ressemble énormément au précédent à la différence que j'ai augmenté le nombre de noyaux à 128 sinon l'image obtenue était de mauvaise qualité (similarité d'environ 30%).

2.5.2 Résultats

Les résultats préléminaires obtenus par le réseau sont semblables à celui du précédent cas. On peut remarquer sur la figure 15 que les résultats semblent visuellement pertinents pour une coupe comprenant une certaine quantité de myéline mais lorsqu'il n'y en a pas ou peu, l'image donne un résultat assez mauvais.

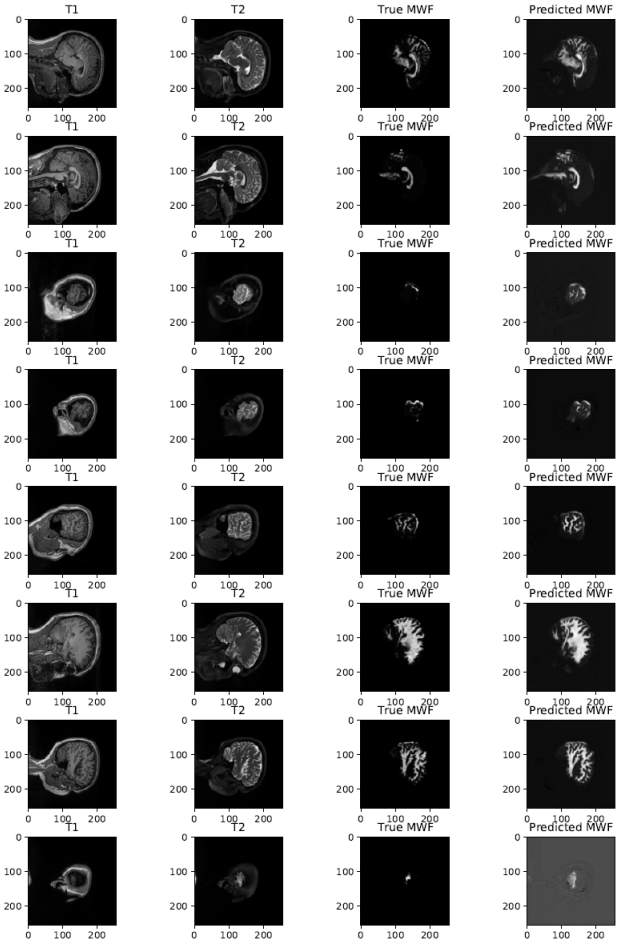

FIGURE 15 – Prédiction des images MWF

Plus quantiativement, on obtient une similarité d'environ 87.2%. La figure 16 représente l'index de similarité entre la vérité et la prédiction d'un échantillon aléatoire du dataset de test. Les couleurs foncées indiquent une forte similarité à l'inverse des zones claires (vertes-jaunes) qui indiquent des différences. Comme sur le cas T1 vers T2, le réseau rencontre le plus de difficulté au niveau de la sphère ORL, des méninges et du crâne.

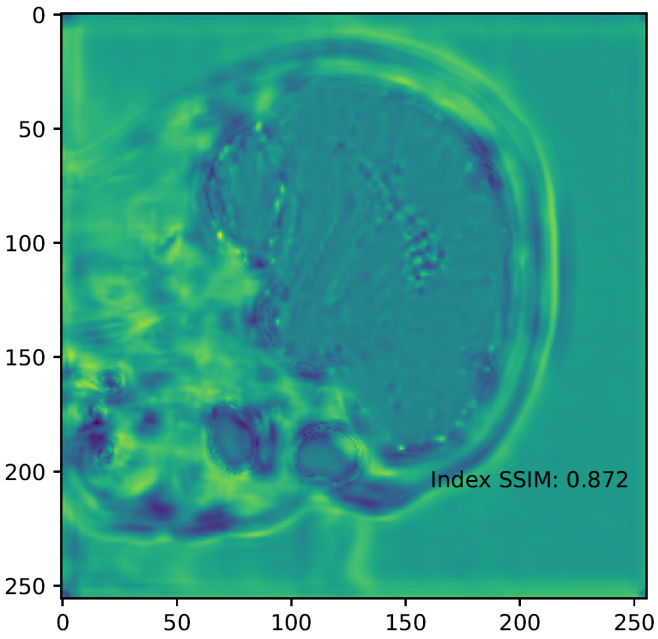

FIGURE 16 – Similarité des structures (MWF)

2.5.3 Discussion

A posteriori, je pense qu'il aurait été intéressant d'évaluer, par l'index SSIM, les performances des réseaux génératifs uniquement sur le tissu cérébral puisque c'est ce qui nous intéressait. Voire même réalier un pré-traitement pour extraire uniquement le cerveau des images. Puis entrainer le réseau sur ces images dont les sources d'erreur les plus imporantes sont supprimées.

De manière plus générale, sur la base de données, nous avions à disposition les images 3D de 127 patients. Ce nombre n'étant pas suffisant pour entraîner un réseau, on a "artificiellement" augmenté ce nombre en les transformant en images 2D (donc $127 * 255$ images). Cela permet d'avoir un dataset beaucoup plus conséquent. Mais on perd une dimension qui pourrait être utile au réseau lors de l'entraînement (peut-être sur les zones où la myéline est peu ou pas présente).

Aussi, cette base de données WAND, au moment de mon stage, ne contenait que des images de patients sains. Or, si on fait des IRM, c'est généralement pour étudier/diagnostiquer des pathologies (en recherche/clinique). Donc en l'état, on ne peut pas savoir si un réseau "généraliste" serait en mesure de prédire des images à partir d'images initiales de patient aussi bien sain que pathologique.

Enfin, la suite du travail aurait été de pouvoir explorer des architectures plus complexes (comme pix2pix). Cependant, mon stage ayant dû s'arrêter de façon anticipée, je n'ai pas eu le temps d'obtenir suffisament de données pour en parler plus en profondeur.

Conclusion

Ce stage au sein du Cardiff University Brain Research Imaging Center (CUBRIC) m'a permis de me plonger au cœur de la recherche en neuroimagerie et d'y apporter ma contribution. J'y ai adoré les collaborations et discussions avec des chercheurs venant d'horizons scientifiques très différents. J'ai pu y développer mes premières compétences en traitement d'images et en apprentissage profond que j'espère poursuivre dans la suite de mon cursus.

Le projet principal, visant à reproduire des images pondérées T2 et des images MWF à partir d'images T1, a permis d'explorer les capacités des réseaux de neurones convolutifs (CNN) dans ce type de tache. Les résultats obtenus sont prometteurs malgré certaines limites dans les zones spécifiques du cerveau, comme les circonvolutions et les régions ORL.

Ce travail n'avait pas pour objectif de créer l'architecture de réseau de neurones la plus avancée. Nous avons utilisé une architecture simple, comprenant une partie de downscaling suivie d'une partie d'upscaling. Les futures recherches pourraient explorer des architectures plus sophistiquées, telles que les réseaux adverses génératifs (GAN) comme pix2pix, pour améliorer davantage la qualité des reconstructions d'images. De plus, la base de données utilisée ne comportait que des images de cerveaux sains. Il sera essentiel d'évaluer les performances de ce réseau de neurones sur des cerveaux pathologiques.

En conclusion, cette expérience a été extrêmement enrichissante tant sur le plan scientifique que personnel. Elle m'a offert une vision approfondie des défis actuels en neuroimagerie. Je tiens à remercier chaleureusement tous ceux qui m'ont accompagné et soutenu durant ce stage, en particulier le Pr. Mara Cercignani, le Dr. Matteo Mancini, et le Dr. Carine Guivier-Curien. Leur expertise et leur conseils ont été essentiels à la réussite de ce projet.

Table des figures

Bibliographie

"Neuromatch Academy." n.d. Accessed May 25, 2022. https://academy.neuromatch.io/.

"BET - FslWiki." n.d. Accessed May 18, 2022. https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/BET.

"Cardiff University Brain Research Imaging Centre." n.d. *Cardiff University*. Accessed July 28, 2022. https://www.cardiff.ac.uk/cardiff-university-brain-research-imaging-centre.

"Contenu de La Documentation Python — Documentation Python 3.10.7." n.d. Accessed May 16, 2022. https://docs.python.org/fr/3/contents.html.

"FAST - FslWiki." n.d. Accessed May 18, 2022. https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FAST.

"FLIRT - FslWiki." n.d. Accessed May 18, 2022. https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT.

Jenkinson, Mark, Peter Bannister, Michael Brady, and Stephen Smith. 2002. "Improved Optimization for the Robust and Accurate Linear Registration and Motion Correction of Brain Images." *NeuroImage* 17 (2) : 825–41. https://doi.org/10.1006/nimg.2002.1132.

Jenkinson, Mark, and Stephen Smith. 2001. "A Global Optimisation Method for Robust Affine Registration of Brain Images." *Medical Image Analysis* 5 (2) : 143–56. https://doi.org/10.1016/S1361-8415(01)00036-6.

"PyTorch Documentation — PyTorch 1.12 Documentation." n.d. Accessed May 25, 2022. https://pytorch.org/docs/stable/index.html.

Quantitative MRI of the Brain : Principles of Physical Measurement, Second Edition. 2018. CRC Press. https://doi.org/10.1201/b21837.

"Scikit-Image 0.19.2 Docs — Skimage V0.19.2 Docs." n.d. Accessed June 11, 2022. https://scikit-image.org/docs/stable/.

"Scikit-Learn : Machine Learning in Python — Scikit-Learn 1.1.2 Documentation." n.d. Accessed June 11, 2022. https://scikit-learn.org/stable/index.html.

Smith, Stephen M. 2002. "Fast Robust Automated Brain Extraction." *Human Brain Mapping* 17 (3) : 143–55. https://doi.org/10.1002/hbm.10062.

Zhang, Y., M. Brady, and S. Smith. 2001. "Segmentation of Brain MR Images Through a Hidden Markov Random Field Model and the Expectation-Maximization Algorithm." *IEEE Transactions on Medical Imaging* 20 (1) : 45–57. https://doi.org/10.1109/42.906424.

Annexes

Les extraits de code proviennent du CNN prédisant une image MWF à partir d'images T1 & T2.

A Dataset

A.A Définition d'une class dataset

```
class MWF_pred_Dataset(Dataset):
    def __init__(self, match, img_dir, transform=None, target_transform=None):
        self.img_match = match
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_match)

    def __getitem__(self, idx):

        T1 = io.imread(self.img_dir + "T1/Reoriented/Slices/" +
        ↪  str(self.img_match.iloc[idx, 0]))

        T2 = io.imread(self.img_dir + "T2/Reoriented/Aligned/Slices/" +
        ↪  str(self.img_match.iloc[idx, 1]))

        MWF = io.imread(self.img_dir + "MWF/Reoriented/Aligned/Slices/" +
        ↪  str(self.img_match.iloc[idx, 2]))

        T1_T2 = np.array([T1, T2])
        T1_T2 = np.transpose(T1_T2, (1, 2, 0))

        if self.transform:
            T1_T2_transformed = self.transform(T1_T2)
        else:
            T1_T2_transformed = T1_T2
        if self.target_transform:
            MWF_transformed = self.target_transform(MWF)
        else:
            MWF_transformed = MWF

        return T1_T2_transformed, MWF_transformed
```

A.B Chargement d'un dataset

```
train_data = MWF_pred_Dataset(
    img_dir="",
    match=train_match,
    transform=transforms.ToTensor(),
    target_transform=transforms.ToTensor()
)

train_loader = DataLoader(train_data, batch_size=50, shuffle=True)
```

B Définition de la fonction d'entraînement

```
def train(model, device, train_loader, validation_loader, epochs, lr=0.001):
    criterion = nn.MSELoss()

    train_loss, validation_loss = [], []

    for epoch in range(epochs):
        if epoch % 3 == 0 and epoch != 0:
            lr = lr / 2
        running_loss = 0.
        optimizer = torch.optim.Adam(model.parameters(), lr=lr, betas=(0.9, 0.999),
        ↪  eps=1e-08, weight_decay=0)
        model.train()
        with tqdm(train_loader, unit='batch') as tepoch:
            for data, target in tepoch:

                if device == "cuda":
                    data, target = data.to(device).type(torch.cuda.FloatTensor),
                    ↪  target.to(device).to(device).type(
                        torch.cuda.FloatTensor)
                else:
                    data, target = data.to(device).float(), target.to(device).float()

                target = torch.unsqueeze(target, 1)
                optimizer.zero_grad()
                output = model(data)
                loss = criterion(output.reshape(-1), target.reshape(-1))
                loss.backward()
                optimizer.step()

                tepoch.set_postfix(lr=lr, loss=loss.item(), epoch=str(epoch + 1) + "/"
                ↪  + str(epochs))
                time.sleep(0.1)
                running_loss += loss.item()

        train_loss.append(running_loss / len(train_loader))

        model.eval()
        running_loss = 0.

        with torch.no_grad():
            for data, target in validation_loader:

                if device == "cuda":
                    data, target = data.to(device).type(torch.cuda.FloatTensor),
                    ↪  target.to(device).to(device).type(
                        torch.cuda.FloatTensor)
                else:
                    data, target = data.to(device).float(), target.to(device).float()

                target = torch.unsqueeze(target, 1)

                optimizer.zero_grad()
                output = model(data)
                loss = criterion(output.reshape(-1), target.reshape(-1))
```

```
                tepoch.set_postfix(loss=loss.item())
                running_loss += loss.item()
                time.sleep(0.1)

            validation_loss.append(running_loss / len(validation_loader))

    return train_loss, validation_loss
```

C Fonction ADAM

La fonction ADAM est définie telle que :

```
class torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08,
↪  weight_decay=0, amsgrad=False, *, foreach=None, maximize=False, capturable=False)
```

Avec :

input : γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective), λ (weight decay) , $amsgrad$, $maximize$
initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$
for $t = 1$ **to** ... **do**
    if $maximize$:
        $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
    else
        $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    if $\lambda \neq 0$
        $g_t \leftarrow g_t + \lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
    $\widehat{m_t} \leftarrow m_t / (1-\beta_1^t)$
    $\widehat{v_t} \leftarrow v_t / (1-\beta_2^t)$
    if $amsgrad$:
        $\widehat{v_t}^{max} \leftarrow max(\widehat{v_t}^{max}, \widehat{v_t})$
        $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / (\sqrt{\widehat{v_t}^{max}} + \epsilon)$
    else
        $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / (\sqrt{\widehat{v_t}} + \epsilon)$
return θ_t