الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Projet de Fin d'Études

Filière Télécommunication

Spécialité Réseaux & Télécoms

Présenté par

Aymen BOUGUERRA

Utilisation du Deep Learning pour la classification des images IRM pour la détection des AVC

Proposé par : Mr. Yacine Kabir

Année Universitaire 2020-2021

ملخص:

السكتة الدماغية هي السبب الرئيسي لإعاقة البالغين في العالم، حيث يعاني أكثر من ثلثي الأفراد من إعاقات طويلة الأمد عادةً بسبب التأخير في العلاج، لأنه أثناء السكتة الدماغية، يُحرم الخلايا العصبية من الدم، ويموت الملايين منهم كل دقيقة. مما يجعل تشخيص السكتة الدماغية سباقًا مع الوقت. من خلال تطبيق تقنية تصنيف الصور القائمة على التعلم العميق الناشئة "الشبكة العصبية التلافيفية" على قاعدة بيانات كبيرة من صور سكتة دماغية "ATLAS" المقسمة يدويًا، يمكننا الآن تأكيد أو إنكار وجود الآفات في كل مقطع T1- من صورة الدماغ ثلاثية الأبعاد المصورة باستعمال التصوير بالرنين المغناطيسي.

كلمات المفاتيح: الذكاء الاصطناعي، التعلم العميق، كشف السكتة الدماغية، التصوير بالرنين المغناطيسي، ATLAS، تصنيف الصور، VGG16، InceptionV3.

Résumé :

L’accident vasculaire cérébral est la première cause de handicap acquis à l'âge adulte dans le monde, avec plus de deux tiers des individus qui subissent des handicaps à long terme généralement à cause du retard de traitement. En effet, pendant un AVC, privé de sang, des millions de neurones meurent chaque minute, ce qui rend le diagnostic d’un AVC une course contre la montre. En appliquant la technique émergeante de classification d’images à base d’apprentissage profond « réseau neuronal convolutionnel » à une large base de données d’images d’AVC segmentées à la main « ATLAS », nous sommes à présent en mesure de confirmer ou infirmer la présence de lésions dans chaque coupe d’une image 3D cérébrale de type T1 avec une précision de validation de 95% ± 1.5% et un score AUC de 0.996. Nous comparerons aussi notre modèle proposé aux modèles référentiels VGG16 et inception V3.

Mots clés : Réseaux neuronaux convolutionnels, IRM cérébrale, ATLAS, Classification d’images, VGG16, InceptionV3.

Abstract:

Stroke is the leading cause of adult disability worldwide, with up to two-thirds of individuals experiencing long-term disabilities usually because of the treatment delay. During a stroke, billions of brain cells die each minute which makes diagnosing a stroke a time sensitive mission. Applying the promising technique of image classification with the Deep learning 'Convolutional neural network' on A large, open-source dataset of stroke anatomical brain images and manual lesion segmentations "ATLAS" we will now be able to confirm or deny the presence of lesions on each slice of a 3D brain MRI-T1 weighted image with a validation rate of 95% ± 1.5% and an AUC score of 0.996. We will also compare our proposed model to other referential models like VGG16 and InceptionV3.

Key words: Convolutional neural network, Deep Learning, Stroke detection, MRI, ATLAS, Image Classification, VGG16, InceptionV3.

Table des matières

Liste des abréviations

AVC Accident vasculaire cérébral

IRM Image par résonance magnétique

IA Intelligence artificielle

CNN Convolutional neural network ou Réseau neuronal convolutif ou convolutionnel

ReLu Unité de rectification linéaire

CONV Couche de convolution

POOL Couche de pooling (MaxPooling dans la majorité des cas)

FC Fully connected layer ou couche entièrement connectée

ROC (receiver operating characteristic)

AUC Aire sous la courbe ROC

TPR True Positive Rate

FPR False Positive Rate

Liste des figures

Introduction

Inspiré du cortex visuel animal, et utilisant l'apprentissage profond, le réseau neuronal convolutionnel nous permet de classifier des images de plusieurs catégories en identifiant les caractéristiques propres à chaque classe. Même s'il existe plusieurs domaines où nous pourrions appliquer ces méthodes, le domaine qui reçoit le plus d'attention est bien celui de l'imagerie médicale.

L'AVC est une urgence médicale qui nécessite un diagnostic rapide [3] afin de limiter les éventuelles séquelles liées à la privation des neurones du sang qui véhicule l'oxygène et le glucose nécessaire à la survie de ces derniers. Chaque minute perdue entre l'apparition des symptômes et le traitement de l'AVC risque d'accroitre gravement les futurs handicaps causés par la mort des neurones.

Même si le scanner CT est plus communément utilisé pour l'imagerie médicale en cas de suspicion d'AVC à cause de sa disponibilité, l'imagerie par IRM reste le meilleur choix à cause de sa meilleure sensibilité à la détection de l'ischémie cérébrale comme le démontre l'intéressante étude réalisée par [1].

Plusieurs chercheurs ont déjà relevé ce défi et ont déjà réussi à segmenter les zones lésées suite à un AVC grâce à une autre architecture utilisant l'apprentissage profond nommée U-net et cela avec une précision très satisfaisante, cependant cette architecture non seulement a besoin d'une base de données plus complète (images et leurs masques respectifs) mais fournit comme résultat un masque au lieu d'un choix binaire.

Dans cette étude, nous allons utiliser une architecture CNN classique avec le DATASET ATLAS [2]. Le but est de prédire la présence ou non d'une lésion sans la localisation de cette dernière.

Le but de ce travail est non-seulement de comparer plusieurs approches et architectures CNN différentes mais de les comparer aussi avec des méthodes plus conventionnellement utilisé comme le U-net, et des modèles de référence comme VGG16 et InceptionV3

Ce mémoire est divisé en quatre chapitres. Nous nous intéresserons dans le premier chapitre à donner un aperçu sur les AVC, ainsi qu'au fonctionnement de l'IRM utilisé pour l'imagerie de nos données. Le deuxième chapitre quant à lui va nous initier au réseau neuronal convolutionnel, afin de donner une idée du fonctionnement de ce dernier. Le troisième chapitre concernera la manière d'utilisation de la base de données ATLAS, pour créer une base de données adaptée à notre utilisation. On trouvera aussi dans ce chapitre les quatre approches utilisées ainsi que diverses hypothèses et commentaires, les obstacles rencontrés et la solution que nous proposerons. Le quatrième chapitre sera consacré quant à lui à l'analyse et comparaison des résultats et nous clôturerons ce travail par une conclusion et quelques perspectives.

Chapitre 1

Contexte Médical

1.1 Introduction

Ce chapitre a pour but de donner un niveau minimal dans la compréhension des AVC, car cette maladie touchant le cerveau est très complexe. Nous nous intéresserons aussi au fonctionnement de l'IRM qui est utilisée pour l'imagerie médicale des patients atteints.

1.2 Généralités sur les AVC

Mieux connu sous le nom commun «attaque cérébrale», l'accident vasculaire cérébral (AVC) est une perte brutale de la fonction cérébrale due à un infarctus ou à une hémorragie [3].

	AVC ischémique	AVC hémorragique
Cause	Infarctus cérébral : caillot de sang bouchant une artère et interrompant la circulation du sang dans le cerveau	Hémorragie dans le cerveau
Fonctionnement	Le caillot peut être véhiculé par la circulation sanguine Il est alors question d'embolie cérébrale. Dans d'autres cas, l'artère se bouche seule.	L'hémorragie est due à la rupture d'une artère ou d'un anévrysme
Prévalence	80 % des cas	20 %

Figure 1.1 Généralités sur l'AVC [3].

Les séquelles d'un AVC vont dépendre de l'endroit du cerveau touché et de l'étendue des dommages. Un AVC peut affecter une ou plusieurs fonctions cérébrales, notamment la capacité de :

- Se déplacer.
- Voir.

- Se souvenir.
- Raisonner.

Un AVC peut également entraîner une hémiplégie ou une aphasie.

En fonction des lésions subies, il existe de nombreuses méthodes de rééducation, et il est parfois possible de retrouver ses capacités, dans une certaine mesure. Il est indispensable d'avoir un suivi médical adapté, d'avoir un soutien social et psychologique au quotidien pour aider la personne à retrouver ses marques, et éventuellement adapter son mode de vie : aménagement du foyer, assistance à domicile, etc.

Chaque année, environ 60.000 Algériens sont victimes d'AVC et seulement un cas sur 200 bénéfice des traitements d'urgence et on estime que plus de 60% des victimes d'AVC gardent des séquelles à vie : il peut s'agir d'une paralysie, de troubles de la mémoire ou encore des difficultés pour parler. Pourtant, une prise en charge très précoce peut éviter les complications et en limiter les séquelles [4].

1.3 Les causes de l'AVC

Qu'il soit dû à l'occlusion d'une artère cérébrale (AVC ischémique) ou à la rupture d'un vaisseau (AVC hémorragique), l'accident vasculaire cérébral survient toujours sans prévenir, subitement. Il s'agit d'une urgence médicale et il faut faire vite.

Le plus souvent, c'est un caillot de sang bouchant une artère qui est à l'origine d'un AVC. Il peut être véhiculé par la circulation sanguine Il est alors question d'embolie cérébrale. Dans d'autres cas, l'artère se bouche seule.

La cause majeure de l'AVC reste toutefois l'hypertension artérielle. Un AVC peut également survenir du fait de la malformation d'un vaisseau ou par suite d'un traitement anticoagulant.

1.4 Prévention et conséquences

Alors que la rupture d'un vaisseau sanguin due à une malformation n'est absolument pas prévisible, on peut cependant agir sur les autres causes possibles de l'AVC :

- L'hypertension artérielle doit être prise en charge avec sérieux,
- Le taux de cholestérol doit également être surveillé,
- Il ne faut jamais modifier sans avis médical l'un de ces traitements.

Les personnes qui survivent à un AVC souffrent parfois de séquelles irréparables. La plupart du temps, elles peuvent néanmoins récupérer en partie leurs capacités. Des changements de comportement peuvent aussi être observés. Ces modifications ont souvent un retentissement négatif sur le quotidien du patient et celui de la famille.

Pour éviter les récidives, lorsque l'athérosclérose ou l'hypertension artérielle ont été à l'origine de l'AVC, les traitements médicaux doivent être scrupuleusement suivis**.**

1.5 Avantages de l'IRM par rapport au Scanner CT

A la phase aiguë, l'intérêt du scanner sans injection repose sur sa grande accessibilité, son utilisation comme examen de première intention lors des essais thérapeutiques qui ont montré l'efficacité des thrombotiques, et sa valeur pronostic (une hypodensité scénographique précocement étendue est associée à un très mauvais pronostic). Cependant, dans les 6 premières heures, le scanner est souvent normal (ou subnormal), ce qui peut conduire à des erreurs diagnostiques. Le scanner de perfusion est une technique très prometteuse, en particulier pour imager la zone de pénombre ischémique à risque d'extension de l'infarctus à la phase aiguë.

La mise au point de nouvelles séquences d'IRM, diffusion, perfusion et angio-IRM, a bouleversé la prise en charge en imagerie des patients suspects d'AVC [1].

L'IRM avec des séquences dites conventionnelles (T1, T2, T2*) permet de distinguer sans ambiguïté une hémorragie cérébrale d'un infarctus cérébral non hémorragique. En effet, dans les premières heures, les hémorragies sont en hypersignal sur les séquences FLAIR ou T2, en hyposignal discret en T1, et la présence d'un hyposignal sur les séquences en écho de gradients (T2*) signe la présence d'un saignement récent.

Une imagerie cérébrale est indispensable en urgence pour :

– confirmer le diagnostic d'AVC,

– distinguer une hémorragie cérébrale d'un infarctus,

– poser l'indication d'un traitement thrombolytique,

– orienter le bilan étiologique.

Dans les premières heures, l'IRM est préférée au scanner chaque fois que possible en raison de sa meilleure sensibilité à la détection de l'ischémie cérébrale [1].

1.6 Informations générales sur l'IRM

L'imagerie par résonance magnétique (**IRM**) est une technique d'imagerie médicale permettant d'obtenir des vues en deux ou en trois dimensions de l'intérieur du corps de façon non invasive avec une résolution en contraste relativement élevée.

L'IRM repose sur le principe de la résonance magnétique nucléaire (RMN) qui utilise les propriétés quantiques des noyaux atomiques pour la spectroscopie en analyse chimique. L'IRM nécessite un champ magnétique puissant et stable produit par un aimant supraconducteur qui crée une magnétisation des tissus par alignement des moments magnétiques de spin. Des champs magnétiques oscillants plus faibles, dits « radiofréquence », sont alors appliqués de façon à légèrement modifier cet alignement et produire un phénomène de précession qui donne lieu à un signal électromagnétique mesurable. La spécificité de l'IRM consiste à localiser précisément dans l'espace l'origine de ce signal RMN en appliquant des champs magnétiques non uniformes, des « gradients », qui vont induire des fréquences de précession légèrement différentes en fonction de la position des atomes dans ces gradients. Sur ce principe qui a valu à ses inventeurs, Paul Lauterbur et Peter Mansfield le prix Nobel de physiologie ou médecine en 2003, il est alors possible de reconstruire une image en deux dimensions puis en trois dimensions de la composition chimique et donc de la nature des tissus biologiques explorés [5].

En imagerie médicale, l'IRM est principalement dédiée à l'imagerie du système nerveux central (cerveau et moelle épinière), des muscles, du cœur et des tumeurs. Grâce aux différentes séquences, on peut observer les tissus mous avec des contrastes plus élevés qu'avec la tomodensitométrie . En revanche, l'IRM ne permet pas l'étude des

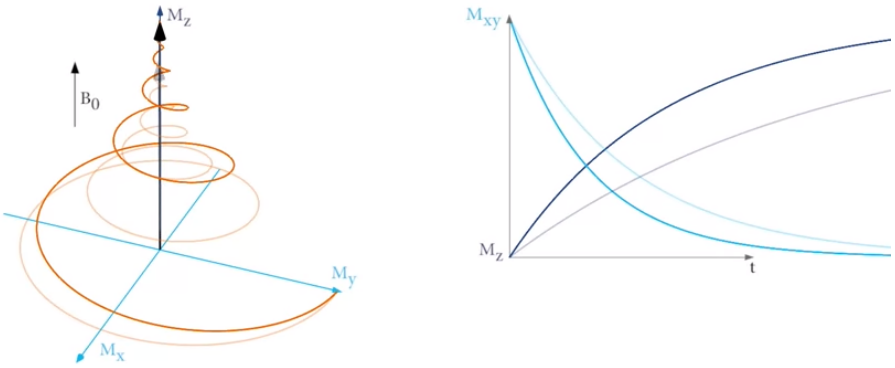
corticales osseuses (tissus « durs ») trop pauvres en hydrogène, ni donc la recherche fine de fractures où seul l'œdème péri-lésionnel pourra être observé.

L'appareil IRM est parfois désigné sous le nom de « *scanner* », ce qui en français prête à confusion avec le tomodensitomètre. Contrairement à ce dernier (et à d'autres techniques d'imagerie comme la TEP), l'examen IRM n'est pas invasif et n'irradie pas le sujet. Cela en fait donc un outil de prédilection pour la recherche impliquant la personne humaine, et notamment en neurosciences cognitives. À partir des années 1990, la technique d'IRM fonctionnelle, qui permet de mesurer l'activité des différentes zones du cerveau, a en effet permis des progrès importants dans l'étude des fondements neurobiologiques de la pensée [6].

1.7 Principe de fonctionnement

La résonance magnétique (RMN) nucléaire exploite le fait que les noyaux de certains atomes (ou plutôt isotopes atomiques) possèdent un moment magnétique de spin. C'est en particulier le cas de l'atome d'hydrogène 1 que l'on retrouve en grande quantité dans les molécules qui composent les tissus biologiques comme l'eau (H_2O) et les molécules organiques. En RMN (tout comme en IRM), on place les atomes que l'on veut étudier dans un champ magnétique constant. On peut alors imaginer les spins des noyaux atomiques comme des toupies tournant sur elles-mêmes autour de leur axe et effectuant un mouvement rapide de précession autour de l'axe du champ magnétique (mouvement appelé précession de Larmor). Cette fréquence de précession est exactement proportionnelle à l'intensité du champ magnétique (qui est de quelques teslas pour les appareils d'IRM actuels). On applique alors à ces atomes une onde électromagnétique à une fréquence bien particulière dite fréquence de résonance ou fréquence de Larmor. En effet, pour que le champ oscillant de l'onde électromagnétique puisse avoir un effet notable sur les spins, il faut que sa fréquence soit ajustée au mouvement de précession de ces spins (phénomène de résonance). La fréquence de Larmor étant différente pour des isotopes atomiques différents (à cause d'un rapport gyromagnétique différent), un choix judicieux de cette fréquence permet de cibler quels atomes on va détecter. En IRM, on utilise principalement les atomes d'hydrogène dont la fréquence de résonance est autour de 42 MHz/T, ce qui correspond à la gamme

des ondes radio. En effet, l'atome d'hydrogène qui est constitué d'un seul proton, est très abondant dans les tissus biologiques et en outre, son moment magnétique nucléaire est relativement fort, ce qui fait que la résonance magnétique de l'hydrogène donne lieu à un phénomène de résonance très net et facile à détecter [7].



Spin d'un proton dans à un champ magnétique constant B_0 puis soumis à une onde radio fréquentielle B_1. Visualisation des temps de relaxation T_1 et T_2 [8].

Même s'il s'agit en réalité de phénomènes quantiques, on peut se représenter, de façon imagée, que sous l'effet du champ magnétique statique, les moments magnétiques de spin vont progressivement s'aligner dans une direction initialement parallèle à celui-ci et donner lieu à une aimantation globale dans la direction du champ B_0, dite *direction longitudinale*. Par habitude, on note cette direction de la lettre *Z*. et on note l'aimantation longitudinale résultant de l'addition de tous ces moments magnétiques, *M* En fait, seule une très faible proportion (environ 0,001 %) des moments magnétiques nucléaires s'aligne dans la direction *Z*, la très grande majorité ne possède pas une orientation stable en raison de l'agitation thermique, néanmoins cette petite proportion de spins qui « s'alignent » est suffisante pour être détectée, c'est pourquoi on néglige le reste des moments magnétiques des 99,999 % restant qui statistiquement se compensent les uns les autres.

Lorsque l'on applique l'onde magnétique radiofréquence oscillante à la fréquence de Larmor, on va entraîner les moments magnétiques qui vont alors s'écarter progressivement de l'axe *Z* pour aller se placer perpendiculairement à leur axe de départ un peu comme un parapluie qui

s'ouvrirait mais en plus les spins continuent leur rotation autour de l'axe. C'est ce qu'on appelle un mouvement de précession.

L'onde magnétique oscillante, notée B_1 va donc avoir comme rôle de faire « basculer » les moments magnétiques de spin pour les placer dans un plan perpendiculaire à la direction du champ statique B_0. C'est ce qu'on appelle l'excitation : plus celle-ci dure longtemps et plus la proportion de moments magnétiques qui auront basculé sera importante et donc plus l'aimantation longitudinale (dans la direction Z) diminuera.

Lorsqu'on interrompt le champ oscillant, les moments magnétiques qui se sont écartés de leur axe initial vont revenir vers la direction sans cesser de tourner. On peut alors mesurer ce mouvement de rotation des spins sous la forme d'un signal oscillant qui a la même fréquence que l'onde excitatrice. C'est ce signal, dit de précession, qu'on mesure en RMN et en IRM au moyen d'une antenne réceptrice [9].

1.8 Pondération

En modifiant les paramètres d'acquisition IRM, notamment le temps de répétition entre deux excitations et le temps d'écho, temps entre le signal d'excitation et la réception de l'écho, l'utilisateur peut modifier la pondération de l'image, c’est-à-dire faire apparaître les différences de temps T1 et de temps T2 des différents tissus d'un organisme. Les tissus ayant des temps T1 et T2 différents en fonction de leur richesse en atome d'hydrogène et en fonction du milieu dans lequel ces derniers évoluent, peuvent renvoyer des signaux différents si l'on arrive à mettre en évidence ces différences de temps. Pour cela, on teste la réponse des atomes après des excitations particulières.

Des tissus différents ont des T1 différents. Après stimulation de radiofréquence avec un temps de répétition court, on ne laisse pas le temps aux atomes d'hydrogène de certains tissus de revenir en position d'équilibre alors que, pour d'autres atomes d'hydrogène d'autres tissus, le temps est suffisamment long pour qu'il y ait un retour à l'équilibre. Lorsque l'on mesure l'état d'énergie des atomes des tissus, on note des écarts d'état entre ces différents atomes. Si on laissait un temps trop long, tous les atomes auraient le temps de revenir en position d'équilibre et l'on ne noterait plus de différences entre différents tissus.

De même, des tissus différents ont des T2 différents. Après stimulation par un temps d'écho long, on retrouve des décroissances d'énergie d'amplitude plus importante entre les tissus. Les différences de T2 étant plus discriminants si le temps d'écho est long [10].

1.8.1 Pondération T1

En utilisant un temps de répétition court et un temps d'écho court (neutralise les différences de temps T2), on obtient un contraste d'image pondérée en T1, pondération dite « anatomique » : en pondération T1 sur le cerveau, la substance blanche apparaît plus claire que la substance grise. Le liquide céphalorachidien, situé entre la substance grise et l'os apparaît lui nettement plus foncé.

Ces séquences sont également utilisées après injection de produit de contraste, pour caractériser une anomalie.

1.8.2 Pondération T2

En utilisant un temps de répétition long (neutralise les différences de temps T1) et un temps d'écho long, on obtient un contraste d'image dite pondérée en T2, dite aussi pondération « tissulaire » : L'eau et l'œdème apparaissent en hyper signal.

1.9 Conclusion

L'AVC et l'IRM restent des domaines vastes et complexes et même si nous nous sommes intéressés à leurs fonctionnement, nous restons loin de les approfondir et c'est pour cela que nous laisserons de côté les problèmes liés, comme par exemples les artéfacts de l'IRM, pouvons-nous déformer les lésions afin d'améliorer la généralisation ? Nous nous intéresserons donc uniquement à la partie CNN du problème : on utilisera les images de l'AVC comme n'importe quelles autres images appliquées au CNN.

Chapitre 2

Le Réseau Neuronal Convolutionnel

2.1 Introduction

Grace à ce chapitre, nous aurons une vague idée sur le fonctionnement des réseaux neuronaux convolutionnels, car en effet, la discipline est bien trop compliquée pour être entièrement expliquée en quelques pages. On utilise cette méthode de classification « CNN » car elle très similaire à comment le cerveau des animaux y compris les humains reconnais les objets et les formes.

2.2 Le perceptron

Un neurone artificiel est un ensemble d'opérations mathématiques. Tout d'abord un poids et un biais sont appliqués de manière affine à une valeur d'entrée : en analyse d'images celle-ci est la valeur d'un pixel. Puis, une fonction d'activation est appliquée au résultat intermédiaire pour représenter les données dans l'espace des données de cette fonction. Souvent, cette fonction d'activation est non-linéaire, car elle permet de représenter des données complexes où la combinaison linéaire ne fonctionne pas [11].

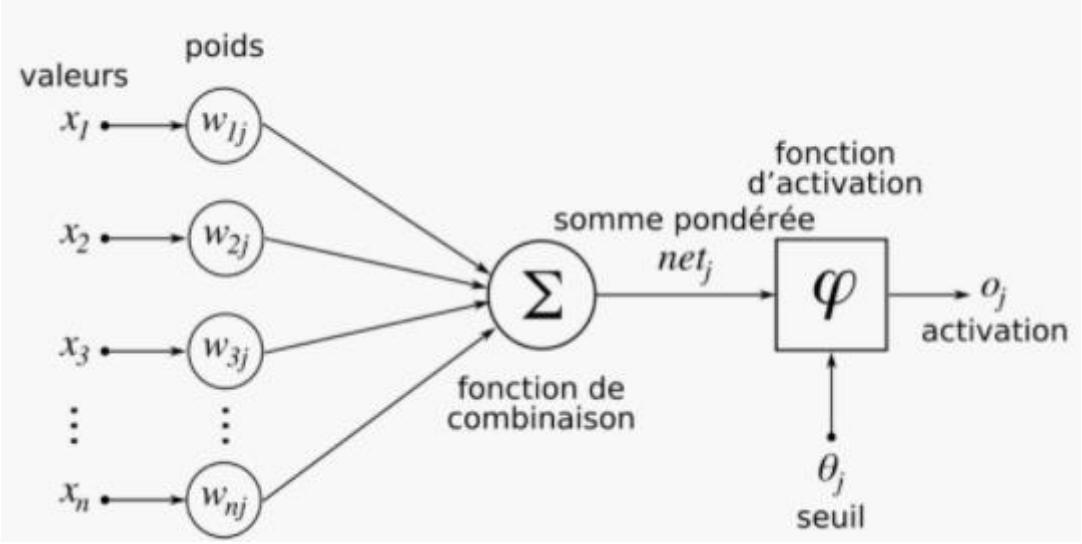


Figure 2.1 : Le perceptron [11].

La sortie est calculée par la formule suivante :

$$\boldsymbol{o_j(t) = -\theta_j + \sum x_n . w_{nj}} \quad (1)$$

Pour la mise à jour des poids :

$$\boldsymbol{w_n(t+1) = w_n(t) + r.\left(d_j - o_j(t)\right) x_{n,j}} \quad (2)$$

Où *r* est le pas d'apprentissage et *d* est la vérité absolue.

Cette architecture de réseaux est inspirée du fonctionnement du cortex visuel des animaux. L'analyse du champ visuel est faite au travers d'un ensemble de sous-régions se chevauchant et pavant l'image. Chaque sous-région est analysée par un neurone du cerveau de l'animal, afin de prétraiter de petites quantités d'information. C'est ce que l'on nomme le traitement convolutif.

L'architecture d'un réseau de neurones convolutifs est formée par une succession de blocs de traitement pour extraire les caractéristiques discriminant la classe d'appartenance de l'image des autres. Un bloc de traitement se compose d'une à plusieurs:

- Couches de convolution (CONV) qui traitent les données d'un champ récepteur ;
- Couches de correction (ReLU), souvent appelée par abus « ReLU » en référence à la fonction d'activation (Unité de rectification linéaire) ;
- Couches de pooling (POOL), qui permet de compresser l'information en réduisant la taille de l'image intermédiaire (souvent par sous-échantillonnage).

Les blocs de traitement s'enchaînent jusqu'aux couches finales du réseau qui réalisent la classification de l'image et le calcul de l'erreur entre la prédiction et la valeur cible :

- Couche « entièrement connectée » (FC), qui est une couche de type perceptron ;
- Couche de perte (LOSS).

La façon dont s'enchaînent les couches de convolution, de correction et de pooling dans les blocs de traitement, ainsi que les blocs de traitement entre eux, font la particularité de l'architecture du réseau. Celle-ci est définie à la suite d'un travail de recherche appliquée.

2.3 La couche de convolution

L'image est :

1. Découpée en sous-régions,
2. Analysée par un noyau de convolution.

Ce noyau de convolution a la taille d'une tuile, souvent 3*3 ou 5*5. La zone analysée (champ réceptif) est légèrement plus grande que le noyau, car un pas est ajouté de façon à ce que les champs réceptifs se chevauchent. Cette astuce permet d'obtenir une meilleure représentation de l'image et d'améliorer la cohérence du traitement de celle-ci.

L'analyse des caractéristiques de l'image par le noyau de convolution est une opération de filtrage avec une association de poids à chaque pixel. L'application du filtre à l'image est appelée une convolution.

Après une convolution, une carte de caractéristiques (en anglais features map) est obtenue, c'est une représentation abstraite de l'image. Ses valeurs dépendent des paramètres du noyau de convolution appliqué et des valeurs de pixels de l'image d'entrée.

Une couche de convolution est un empilement de convolutions. En effet, l'image est parcourue par plusieurs noyaux de convolution qui donnent lieu à plusieurs cartes de caractéristiques de sorties. Chaque noyau de convolution possède des paramètres spécifiques à l'information qui est recherchée dans l'image (par exemple : un noyau de convolution de type filtre Sobel a des paramètres permettant de rechercher les contours dans l'image).

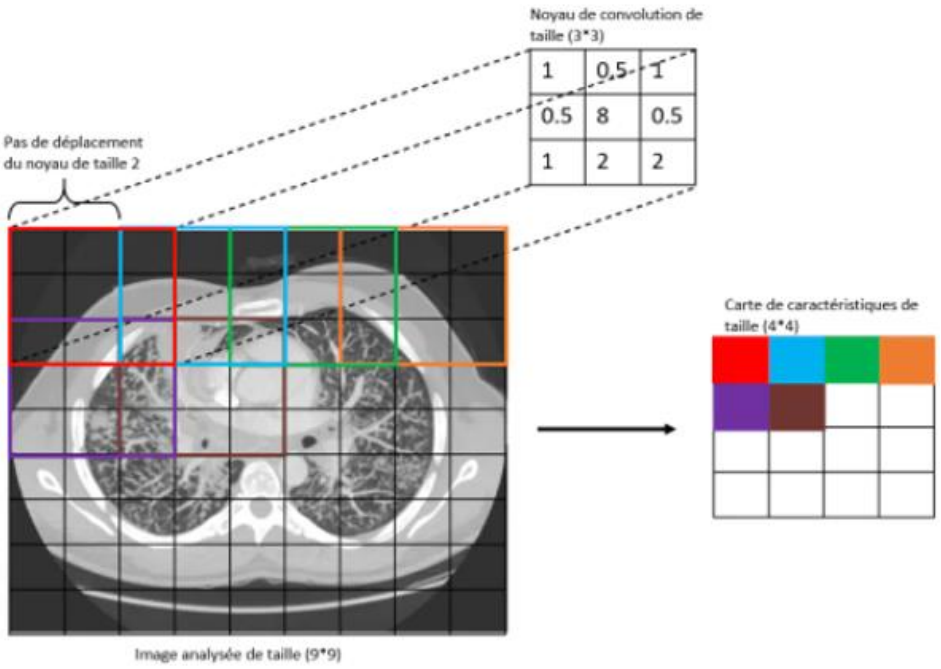


Figure 2.2 Schéma d'une convolution avec un noyau de taille 3x3 et pas de 2 [11].

Le choix des paramètres du noyau de convolution dépend de la tâche à résoudre. Avec les méthodes deep learning[11], ces paramètres sont automatiquement appris par l'algorithme à partir des données d'entraînement notamment grâce à la technique de rétropropagation du gradient, qui permet l'ajustement des paramètres en fonction de la valeur du gradient de la fonction de perte. La fonction de perte calcule l'erreur entre la valeur prédite et la valeur cible.

$$G[m,n] = (f * h)[m,n] = \sum_{j} \sum_{k} h[j,k] f[m-j, n-k] \quad (3)$$

2.4 La couche d'activation

La couche de correction ou d'activation est l'application d'une fonction non-linéaire aux cartes de caractéristiques en sortie de la couche de convolution. En rendant les données non-linéaires, elle facilite l'extraction des caractéristiques complexes qui ne peuvent pas être modélisées par une combinaison linéaire d'un algorithme de régression.

Les fonctions non-linéaires les plus utilisées sont :

-*Softmax* $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ *z est un vecteur* (4)

-*Sigmoide* $\boldsymbol{f(x) = \frac{1}{1+e^{-x}}}$ (5)

-*Tangente hyperbolique* $\boldsymbol{f(x) = \frac{2}{1+e^{-2x}} - 1}$ (6)

- *Unité de rectification linéaire (ReLu)* $\boldsymbol{f(x) = \begin{cases} 0 \; si \; x < 0 \\ x \; si \; x \geq 0 \end{cases}}$ (7)

Souvent, la correction Relu est préférable, car il en résulte la formation de réseau neuronal plusieurs fois plus rapide, sans faire une différence significative à la généralisation de précision.

2.5 La couche de Pooling

L'étape de pooling est une technique de sous-échantillonnage. Généralement, une couche de pooling est insérée régulièrement entre les couches de correction et de convolution. En

réduisant la taille des cartes de caractéristiques, donc le nombre de paramètres du réseau, cela accélère le temps de calcul et diminue le risque de sur-apprentissage.

L'opération de pooling la plus courante est celle du maximum : MaxPool (2*2, 2). Elle est plus efficace que la moyenne, car elle maximise le poids des activations fortes. Elle est appliquée à la sortie de la couche précédente comme un filtre de convolution de taille (2*2) et se déplace avec un pas de 2. En sortie de la couche de pooling est obtenue une carte de caractéristique compressée par un facteur de 4.

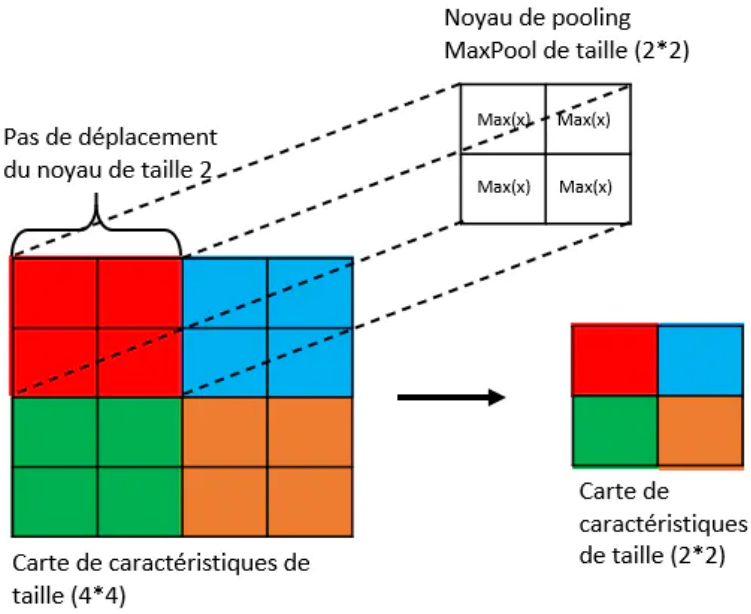


Figure 2.3 Schéma d'une opération de pooling avec un noyau MaxPool de taille 2*2 et d'un pas de 2 [11].

2.6 La couche entièrement connectée

Cette couche est à la fin du réseau. Elle permet la classification de l'image à partir des caractéristiques extraites par la succession de bloc de traitement. Elle est entièrement connectée, car toutes les entrées de la couche sont connectées aux neurones de sorties de celle-ci. Ils ont accès à la totalité des informations d'entrée. Chaque neurone attribue à l'image une valeur de probabilité d'appartenance à la classe i parmi les C classes possibles.

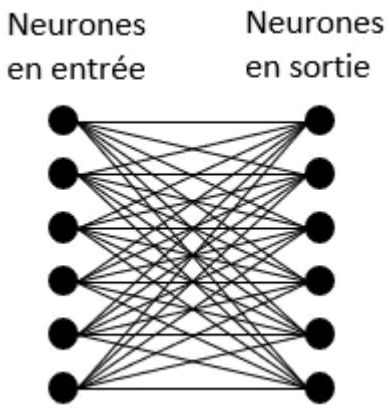


Figure 2.4 Schéma d'une couche entièrement connectée [11].

2.7 La couche de calcul de perte (Loss)

La couche de perte est la dernière couche du réseau. Elle calcule l'erreur entre la prévision du réseau et la valeur réelle. Lors d'une tâche de classification, la variable aléatoire est discrète, car elle peut prendre uniquement la valeur 0 ou 1, représentant l'appartenance (1) ou non (0) à une classe. C'est pourquoi la fonction de perte la plus courante et la plus adaptée est la fonction d'entropie croisée (en anglais cross-entropy).

Comme notre classification sera binaire, nous utiliserons la fonction de calcul de pertes binary crossentropy et sa fonction est la suivante

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} - (y_i * \log(p_i) + (1-y_i) * \log(1-p_i)) \quad (8)$$

Où P est la probabilité de la classe 1 et 1-p la probabilité de la classe 0

Celle-ci est issue du domaine de la théorie de l'information et mesure la différence globale entre deux distributions de probabilité (celle de la prévision du modèle, celle du réel) pour une variable aléatoire ou un ensemble d'évènements [11].

2.8 L'apprentissage du modèle

2.8.1 La rétropropagation du gradient

Les réseaux de neurones convolutifs ont deux sens de propagation de l'information :

- En avant (forward pass) et
- En arrière (backward pass) ou rétropropagation.

La propagation avant est la phase d'entraînement du modèle, où l'estimation des paramètres est calculée. La rétropropagation est la phase de validation du modèle, où les paramètres sont mis à jour en fonction de la valeur du gradient de l'erreur entre l'estimation et l'observation.

Cette mise à jour a pour objectif la minimisation du gradient de l'erreur pour chaque neurone du réseau. Pour cela, une fois le gradient calculé, les valeurs les plus importantes entraînent une forte pénalisation des paramètres qui leurs sont associés (i.e. poids). L'apprentissage profond est un processus itératif, aussi la mise à jour régulière des paramètres du modèle par la minimisation du gradient de l'erreur est appelée descente de gradient.

2.8.2 Le gradient descendant

La descente de gradient est un algorithme itératif de minimisation du gradient de la fonction de perte du modèle. Une itération de la méthode de descente de gradient stochastique (en anglais Stochastic Gradient Descent)

$$w := w - \eta \nabla Q(w) = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(w) \tag{9}$$

$$Q(w) = \frac{1}{n} \sum_{i=1}^{n} Q_i(w) \tag{10}$$

La fonction de coût qui minimise la fonction de perte du modèle, *n* le nombre d'observation du jeu de données, *w* le paramètre qui minimise la fonction de coût, *η* le taux d'apprentissage.
La descente de gradient associée à l'algorithme de rétropropagation permet le processus d'apprentissage automatique des réseaux de neurones.

2.9 Les hyper-paramètres

Les hyper-paramètres sont les paramètres des modèles deep learning, donc des réseaux de neurones. Ils sont de deux types:

2.9.1 Les hyper-paramètres du modèle

Les hyper-paramètres du modèle sont principalement liés à la topologie et la taille du réseau de neurones, souvent prédéfinis par l'architecture de celui choisi (e.g. CNN, RNN, auto-encodeur, GAN). Généralement, ces paramètres sont peu ou pas changés, car l'architecture est souvent utilisée telle qu'elle est proposée par son auteur, voire modifiée à la marge dans le cas d'une stratégie de fine-tuning (détaillée plus loin).

2.9.2 Les hyper-paramètres d'algorithme

Les hyper-paramètres d'algorithme sont ceux sur lesquels le nous allons le plus jouer pour contrôler la vitesse d'apprentissage du modèle. En deep learning, le modèle est encapsulé dans un algorithme d'apprentissage qui définit :

1. le chargement des données en entrée du modèle,
2. le déroulement de la phase d'entraînement,
3. le déroulement de la phase de validation.

Pour le chargement des données, il y a :

- La taille du lot de données (en anglais batch size) fournit en entrée du réseau,
- La méthode de chargement des données (en anglais data loader) avec ou sans échantillonnage.

Pour les phases d'entraînement et de validation, il y a :

- Le nombre d'itérations (en anglais epoch) qui définit le nombre de boucles d'apprentissage (i.e. entraînement-validation) que l'algorithme va réaliser pour permettre au modèle d'améliorer ses estimations,
- La fonction de perte qui calcule la valeur de l'erreur existante entre l'estimation et l'observation,
- L'optimiseur est la fonction d'optimisation utilisée pour la descente de gradient,
- Le taux d'apprentissage (en anglais learning rate) est le pas dans l'algorithme de descente de gradient.

2.10 Les optimiseurs

Les optimiseurs sont la pierre angulaire de l'algorithme de descente de gradient. La méthode principale est celle de la descente de gradient stochastique (Stochastic Gradient Descent) dont sont dérivées d'autres méthodes visant à l'améliorer.

Dans le cas d'un jeu de données très grand, le calcul de la somme des gradients de chaque fonction de coût (une par observation du jeu de données) devient très coûteux. La méthode de descente de gradient stochastique utilise la technique de l'échantillonnage pour calculer la somme des gradients sur un échantillon aléatoire des fonctions de coûts du jeu de données à chaque itération. La taille de l'échantillon aléatoire est de 1 observation pour la méthode standard, mais une méthode dite "en mini lots" est aussi employée où l'échantillon aléatoire est composé de plusieurs observations du jeu de données.

Le taux d'apprentissage (ou le pas d'apprentissage) est le paramètre définissant la "vitesse" de la descente de gradient par l'algorithme d'optimisation. Le choix de sa valeur est difficile, car si elle est trop grande l'algorithme d'optimisation diverge, et si elle est trop petite la vitesse de convergence est trop faible.

Les différents algorithmes d'optimisation présentés ensuite tendent tous à améliorer l'estimation du taux d'apprentissage en fonction des itérations passées de la descente de gradient.

Les optimiseurs les plus courants sont :

SGD moment qui utilise les moments mathématiques pour conserver l'information de la valeur du taux d'apprentissage de l'itération précédente pour définir une nouvelle valeur de celui-ci pour l'itération actuelle ;

$$\begin{aligned} V_t &= \beta V_{t-1} + (1-\beta) S_t \\ V_{t-1} &= \beta V_{t-2} + (1-\beta) S_{t-1} \\ V_{t-2} &= \beta V_{t-3} + (1-\beta) S_{t-2} \end{aligned} \tag{11}$$

AdaGrad qui applique un facteur multiplicatif (la somme des carrés des gradients antérieurs de chaque paramètre du réseau) au taux d'apprentissage de façon à l'adapter à chaque paramètre du réseau pour lisser les mises à jour de ceux-ci entre elles, en amplifiant

les mises à jour trop faibles et atténuant les mises à jour trop fortes. Ce lissage permet d'éviter de définir comme minimum global un minimum local, et stopper la descente de gradient prématurément. Son inconvénient majeur est la trop rapide accumulation des carrés des gradients diminuant drastiquement le taux d'apprentissage à chaque itération ;

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon I + diag(G_t)}} \cdot g_t, \tag{12}$$

Avec

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t), \tag{13}$$

Et

$$G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^\top. \tag{14}$$

RMSprop, qui est une amélioration d'AdaGrad pour ralentir la diminution du taux d'apprentissage à chaque nouvelle itération, et qui calcule le facteur multiplicatif du taux d'apprentissage par une moyenne des carrés des gradients antérieurs de chaque paramètre du réseau ;

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$
$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w} \tag{15}$$

ADAM qui est une amélioration de RMSprop, et qui calcule le facteur multiplicateur du taux d'apprentissage par une combinaison de la moyenne des carrés et de la moyenne des moments d'ordre 2 (soit la variance) des gradients antérieurs. Comme l'optimiseur SGD, l'optimiseur ADAM a également de nombreux dérivés.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{16}$$

2.11 Les stratégies d'apprentissage spécifiques du deep learning

Parmi les stratégies d'apprentissage automatiques certaines sont couramment employées par les algorithmes de deep learning :

2.11.1 L'augmentation des données

Il s'agit d'une technique permettant de contourner le manque de données qui réduit les chances du modèle d'obtenir de bonnes performances en généralisation. Il y existe deux types augmentation:

- Par transformation des données.
- Par création de données synthétiques.

La transformation des données se fait souvent par application de transformations géométriques (e.g. retournement horizontal, rotation), modification du contraste, découpage aléatoire, ajout de bruit blanc, effacement aléatoire des valeurs de pixel d'une zone de l'image. La création de données synthétiques est notamment possible par l'utilisation des réseaux de neurones de type antagoniste génératif (Generative Adversarial Network).

Généralement, l'augmentation des données est utilisée pendant la phase d'entraînement du modèle, donc à partir du jeu de données d'entraînement. Les jeux de données de validation et de test ne sont pas soumis à cette technique.

Lorsqu'un manque de données est observé, il est important d'identifier les caractéristiques de celui-ci : mon jeu de données est-il déséquilibré ? Dans quelles proportions ? Il faut également étudier la variance intra et inter-classes du jeu de données. Ces analyses préparatoires permettent de définir quels types d'augmentation appliquée, en quelle proportion et sur quelles données du jeu de données d'entraînement [11].

2.11.2 Le dropout

La couche Dropout définit de manière aléatoire les unités d'entrée sur 0 avec une fréquence de taux à chaque étape pendant le temps d'entraînement, ce qui permet d'éviter le surajustement. Les entrées non définies sur 0 sont augmentées de 1/ (1 - taux) de sorte que la somme de toutes les entrées reste inchangée.

Il s'agit de délibérément supprimer des connexions de manière aléatoire entre les neurones pour trouver ceux qui sont significatives et ont un impact car ils représentent une

caractéristique réelle et ceux qui n'ont pas d'incidence sur le modèle mais peuvent en revanche prendre un bruit comme caractéristique et fausser l'apprentissage [11].

2.11.3 La régularisation L2

Nous adoptons la régularisation pour éviter le surajustement, un bon modèle d'apprentissage automatique devrait atteindre un faible taux d'erreur sur les données d'entraînement. Mathématiquement, cela équivaut à minimiser la fonction de perte sur les données d'apprentissage étant donné le modèle, cependant, cela pourrait ne pas suffire. Un modèle peut devenir excessivement complexe afin de capturer toutes les relations intrinsèquement exprimées par les données d'apprentissage. Cette augmentation de la complexité pourrait avoir deux conséquences négatives.

Premièrement, un modèle complexe peut nécessiter beaucoup de temps pour être exécuté. Deuxièmement, un modèle complexe peut atteindre de très bonnes performances sur les données d'entraînement, mais très mal sur les données de validation. En effet, le modèle est capable de créer des relations entre de nombreux paramètres dans le contexte de formation spécifique, mais ces relations n'existent en fait pas dans un contexte plus généralisé.

Faire perdre à un modèle sa capacité à généraliser de cette manière est appelé « surapprentissage ou surajustement ». Encore une fois, l'apprentissage est plus une question de généralisation que de mémorisation [11].

2.11.4 L'apprentissage par transfert ou transfer learning

La première technique a également pour objectif de tirer profit des apprentissages antérieurs d'un modèle pré-entraîné sur un jeu de données de taille importante. Les modèles avec un très grand nombre de paramètres, comme les réseaux de neurones profonds, doivent être entraînés sur des jeux de données de taille tout aussi importante pour réduire le risque de sur-apprentissage.

A partir d'un modèle pré-entraîné sur un jeu de données de n classes, sa couche de classification finale est modifiée pour correspondre à la tâche à effectuer, soit la classification de m classes (e.g. m < n, avec le jeu de données spécifique à la tâche). Désormais, le modèle

pré-entraîné possède maintenant une couche de classification finale de m sorties. Les paramètres des couches d'extraction des caractéristiques du modèle sont exclus de la phase de rétropropagation (i.e. figés), afin de conserver les valeurs estimées lors du pré-entraînement avec un jeu de données de taille plus importante (e.g. ImageNet). Seuls les paramètres des couches de classification du modèle seront entraînés sur le jeu de données spécifique.

L'ajustement fin (fine-tuning) permet de réduire le temps d'apprentissage et le risque de surajustement du modèle inhérent à un jeu de données de taille inférieure au nombre de paramètres du modèle.

L'idée principale de la deuxième technique est de tirer profit des apprentissages antérieurs d'un algorithme tiers. Celui-ci aura été entraîné pour une tâche similaire (e.g. classification d'images naturelles) à celle de notre réseau (e.g. classification d'images TDM des différents stades du cancer des poumons), mais à partir d'un jeu de données de taille très supérieure (e.g. ImageNet) au jeu de données disponible pour notre projet. Cette approche réduit le temps d'apprentissage du modèle pour obtenir de bonnes performances, car cette phase de pré-entraînement sur un jeu de données similaires permet d'obtenir les valeurs des paramètres associées à l'extraction de descripteurs d'image conventionnels, tels que l'invariance d'échelle (SIFT) et l'histogramme des gradients orientés (HOG). De ce fait, plutôt que d'initialiser les paramètres de notre réseau aléatoirement, ils sont initialisés à partir des valeurs finales des paramètres de cet algorithme tiers. Ensuite l'entraînement est fait à partir du jeu de données spécifique à notre projet. Et pour cela nous utiliserons les deux prochains modèles :

a VGG16

VGG16 est un modèle de réseau de neurones convolutif proposé par K. Simonyan et A. Zisserman de l'Université d'Oxford dans le document « Very Deep Convolutional Networks for Large-Scale Image Recognition ». Le modèle atteint une précision de test de 92,7% dans le top 5 dans ImageNet, qui est un ensemble de données de plus de 14 millions d'images appartenant à 1000 classes. C'était l'un des modèles célèbres soumis à l'ILSVRC-2014. Il apporte une amélioration par rapport à AlexNet en remplaçant les grands filtres de la taille d'un noyau (11 et 5 dans la première et la deuxième couche convolutive, respectivement) par

plusieurs filtres 3 × 3 de la taille d'un noyau l'un après l'autre. VGG16 a été formé pendant des semaines et utilisait des GPU NVIDIA Titan Black [12].

L'entrée de la couche cov1 est d'une image RVB de taille fixe de 224 x 224. L'image est passée à travers un empilement de couches convolutives (conv.), Où les filtres ont été utilisés avec un très petit champ récepteur : 3 × 3 (qui est la plus petite taille pour capturer la notion de gauche / droite, haut / bas, centre). Dans l'une des configurations, il utilise également des filtres de convolution 1 × 1, qui peuvent être considérés comme une transformation linéaire des canaux d'entrée (suivie d'une non-linéarité). La foulée de convolution est fixée à 1 pixel; le remplissage spatial de conv. L'entrée de couche est telle que la résolution spatiale est préservée après convolution, c'est-à-dire que le remplissage est de 1 pixel pour 3 × 3 conv. couches. La mise en commun spatiale est effectuée par cinq couches de pooling max, qui suivent une partie de la conv. couches (toutes les couches de conv. ne sont pas suivies de la mise en pool maximale). La mise en pool maximale est effectuée sur une fenêtre de 2 × 2 pixels, avec foulée 2 [12].

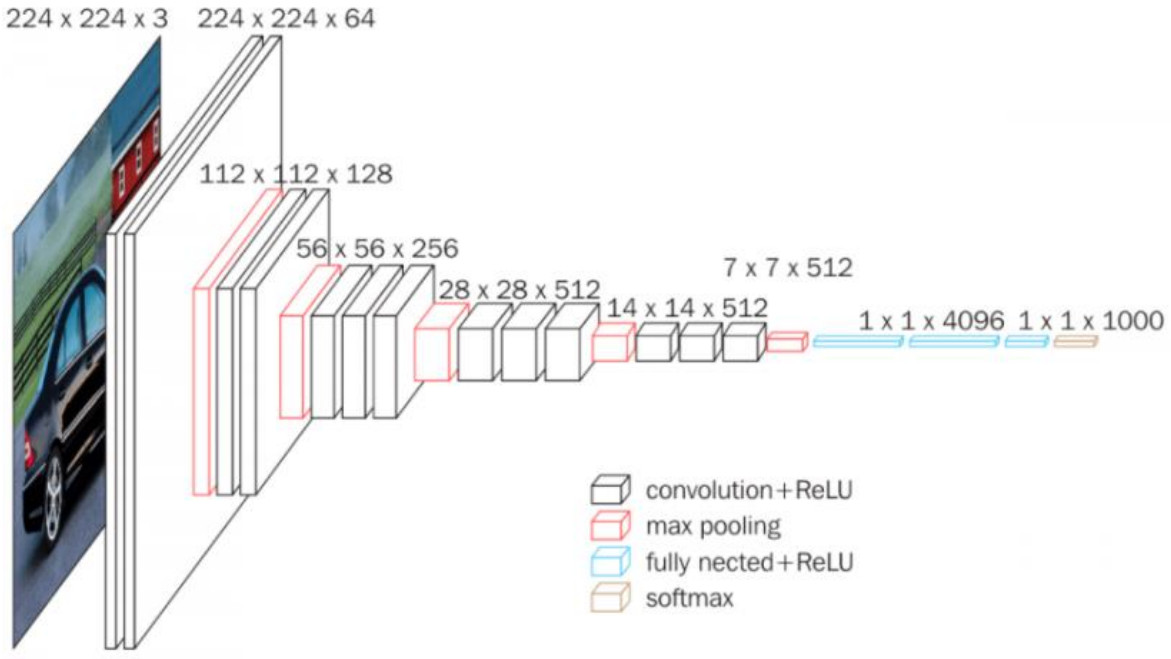


Figure 2.5 Architecture du modèle VGG16 [12].

Trois couches entièrement connectées (FC) suivent un empilement de couches convolutives (qui a une profondeur différente dans différentes architectures) : les deux

premières ont 4096 canaux chacune, la troisième effectue une classification ILSVRC à 1000 voies et contient donc 1000 canaux (un pour chaque classe). La dernière couche est la couche soft-max. La configuration des couches entièrement connectées est la même dans tous les réseaux.

Toutes les couches cachées sont équipées de la non-linéarité de rectification (ReLU). Il est également noté qu'aucun des réseaux (sauf un) ne contient la normalisation de réponse locale (LRN), une telle normalisation n'améliore pas les performances sur l'ensemble de données ILSVRC, mais conduit à une augmentation de la consommation de mémoire et du temps de calcul.

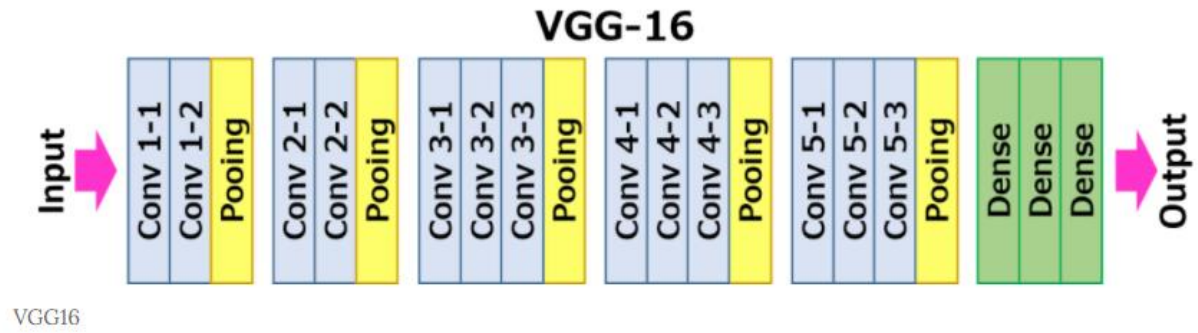


Figure 2.6 Sommaire du modèle VGG16 [12].

b InceptionV3

Inception v3 est un modèle de reconnaissance d'image largement utilisé qui atteint une précision supérieure à 78,1% sur l'ensemble de données ImageNet. Le modèle est l'aboutissement de nombreuses idées développées par plusieurs chercheurs au fil des ans.

Le modèle lui-même est composé de blocs de construction symétriques et asymétriques, y compris les convolutions, la mise en commun moyenne, la mise en commun maximale, les concats, les abandons et les couches entièrement connectées. Batchnorm est largement utilisé dans tout le modèle et appliqué aux entrées d'activation. La perte est calculée via Softmax. [13]

Un diagramme de haut niveau du modèle est présenté ci-dessous :

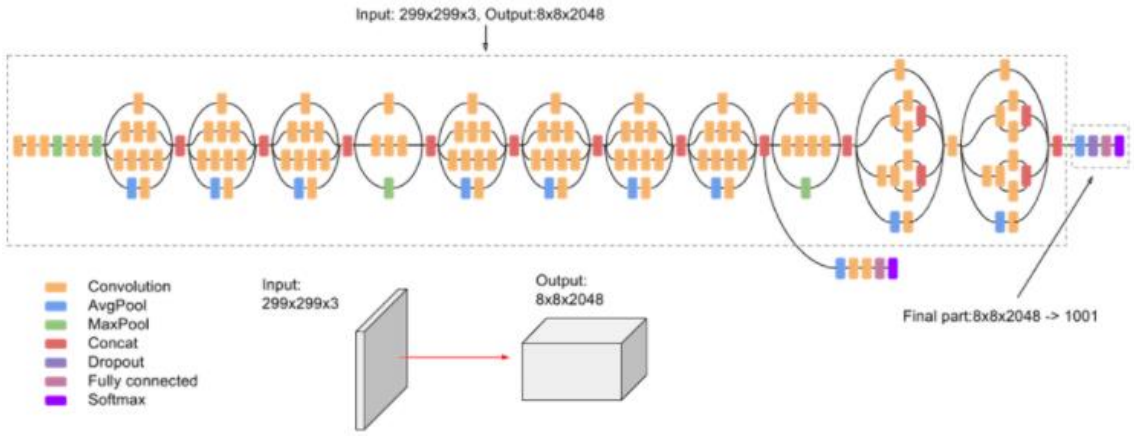


Figure 2.7 Architecture du modèle InceptionV3 [13].

Définition des termes utilisés dans les courbes AUC et ROC.

Une courbe **ROC** (**receiver operating characteristic**) est un graphique représentant les performances d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs en fonction du taux de faux positifs [14].

2.12 Courbe ROC

Une courbe ROC trace les valeurs TVP taux de vrais positifs et TFP taux de faux positifs pour différents seuils de classification. Diminuer la valeur du seuil de classification permet de classer plus d'éléments comme positifs, ce qui augmente le nombre de faux positifs et de vrais positifs. La figure ci-dessous représente une courbe ROC classique.[14]

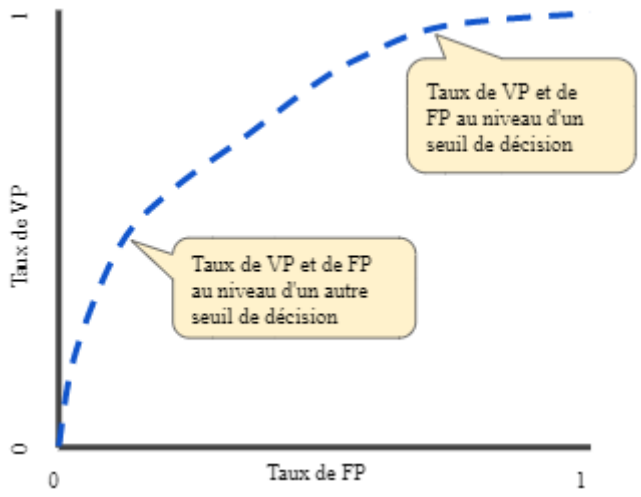


Figure 4.1 Exemple d'une courbe ROC [14].

Pour calculer les points d'une courbe ROC, nous pourrions effectuer plusieurs évaluations d'un modèle en variant les seuils de classification, mais ce serait inefficace. Nous pouvons en revanche calculer efficacement l'aire sous cette courbe, ou AUC, grâce à un algorithme de tri.

2.13 Score AUC :

AUC signifie "aire sous la courbe ROC". Cette valeur mesure l'intégralité de l'aire à deux dimensions situées sous l'ensemble de la courbe ROC (par calculs d'intégrales) de (0,0) à (1,1).

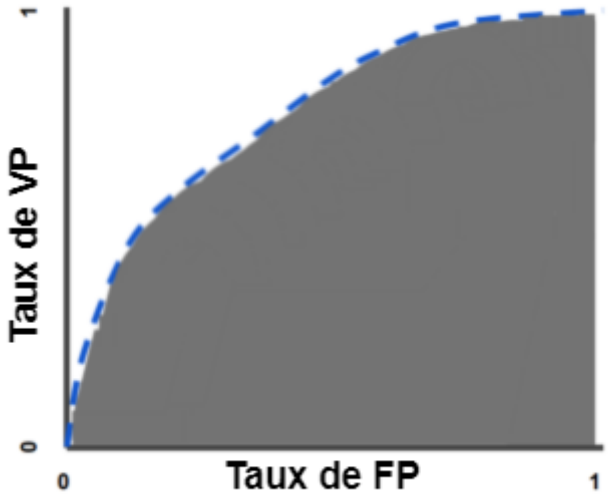


Figure 4.2 Représentation graphique de l'aire sous la courbe 'AUC' [14].

L'AUC fournit une mesure agrégée des performances pour tous les seuils de classification possible. Nous pourrons interpréter l'AUC comme une mesure de la probabilité pour que le modèle classe un exemple positif aléatoire au-dessus d'un exemple négatif aléatoire.

2.14 Conclusion

Même si l'intelligence artificielle nous permet de crée des algorithmes polyvalents, capables de correctement prédire l'appartenance à une classe, il reste cependant relativement difficile d'entrainer ces modèles, afin d'avoir un taux de précision satisfaisant.

Chapitre 3

Etude Expérimentale

3.1 Introduction

Les méthodes des différentes approches étudiées montrent non seulement des expériences différentes mais expliquent le cheminement d'idées qui nous a permis d'aboutir à un résultat satisfaisant.

3.2 Matériel utilisé

Un ordinateur bureau équipé d'un système d'exploitation Microsoft Windows 10 x64 doté d'un processeur I3-3220 à 3.3 GHz avec 8Go de mémoire vive.

3.3 Environnement

Interpréteur Python 3.7 sous pycharm. En utilisant une variété de bibliothèques connus et conventionnelles comme numpy, matplotlib, openCV, scipy, Tnesorflow, keras, sklearn, os, ...

Liste détaillée de toutes les bibliothéques utilisées :

absl-py0.12.0, altgraph0.17, astunparse1.6.3, auto-py-to-exe2.9.0, bottle0.12.19, bottle-websocket0.2.9, cachetools4.2.1, certifi2020.12.5, cffi1.14.5, chardet4.0.0, cycler0.10.0, decorator4.4.2, Eel0.12.4, flatbuffers1.12, future0.18.2, gast0.3.3, gevent21.1.2, gevent-websocket0.10.1, google-auth1.27.1, google-auth-oauthlib0.4.3, google-pasta0.2.0, greenlet1.1.0, grpcio1.32.0, gviz-api1.9.0, h5py2.10.0, idna2.10, imageio2.9.0, importlib-metadata3.7.3, joblib1.0.1, Keras-Preprocessing1.1.2, kiwisolver1.3.1, Markdown3.3.4, matplotlib3.3.4, med2image2.2.10, networkx2.5.1, nibabel3.2.1, numpy1.19.5, oauthlib3.1.0, opencv-python4.5.1.48, opt-einsum3.3.0, packaging20.9, pandas1.2.4, pefile2021.5.24, pfmisc2.2.2, Pillow8.1.0, pip20.1.1, protobuf3.15.6, pudb 2020.1, pyasn10.4.8, pyasn1-modules0.2.8, pycparser2.20, pydicom2.1.2, Pygments2.8.0, pyinstaller4.3, pyinstaller-hooks-contrib2021.1, pyparsing2.4.7, python-dateutil2.8.1, pytz2021.1, PyWavelets1.1.1, pywin32-ctypes0.2.0, PyYAML5.4.1, requests2.25.1, requests-oauthlib1.3.0, rsa4.7.2, scikit-image0.18.1, scikit-learn0.24.1, scipy1.6.1, seaborn0.11.1, setuptools47.1.0, six1.15.0, sklearn0.0, tensorboard2.4.1, tensorboard-plugin-profile 2.4.0 ,tensorboard-plugin-wit1.8.0, tensorflow2.4.1, tensorflow-estimator2.4.0, termcolor1.1.0, threadpoolctl2.1.0, tifffile2021.3.31, typing-extensions3.7.4.3, urllib31.26.4, urwid2.1.2, Werkzeug1.0.1, wheel0.36.2, whichcraft0.6.1 ,wrapt1.12.1, zipp3.4.1, zope.event4.5.0, zope.interface5.4.0

3.4 Première approche

Dans un premier temps, nous voulons observer les résultats d'un algorithme basique, sans aucune amélioration, ce programme sera le fondement de notre programme final. Cela nous permettra de mieux mener les futures expériences et de mieux juger les méthodes d'augmentation de précision que nous utiliserons par la suite.

3.4.1 Traitement d'image et base de données

Le résultat final dépend fortement des données qui nourrissent le modèle, leurs nombre, la qualité ainsi que la diversité sont tous des paramètres très importants qui influeront non seulement sur le bon apprentissage de notre modèle, mais aussi à sa capacité de généralisation ; de bien classifier les images qu'il n'a jamais vu auparavant.

Ci-dessus on trouve les détails et le cheminement du traitement d'image de la base de données initial vers l'ensemble des images finales utilisé par le programme V1.

a La base de données utilisée

« *A large, open source dataset of stroke anatomical brain images and manual lesion segmentations »* abrégé **ATLAS [2]** es une base de données qui comporte 229 images 3D d'IRM type « T1 » qui sont déjà centrées, normalisées (intensité lumineuse) et standardisées à la norme « MNI-152 » et chaque image 3D est accompagné d'un ou de plusieurs (entre un et cinq) masques 3D qui peuvent être superposés sur l'image 3D initial pour distinguer les lésions. Certains patients présentent jusqu'à cinq différentes lésions.

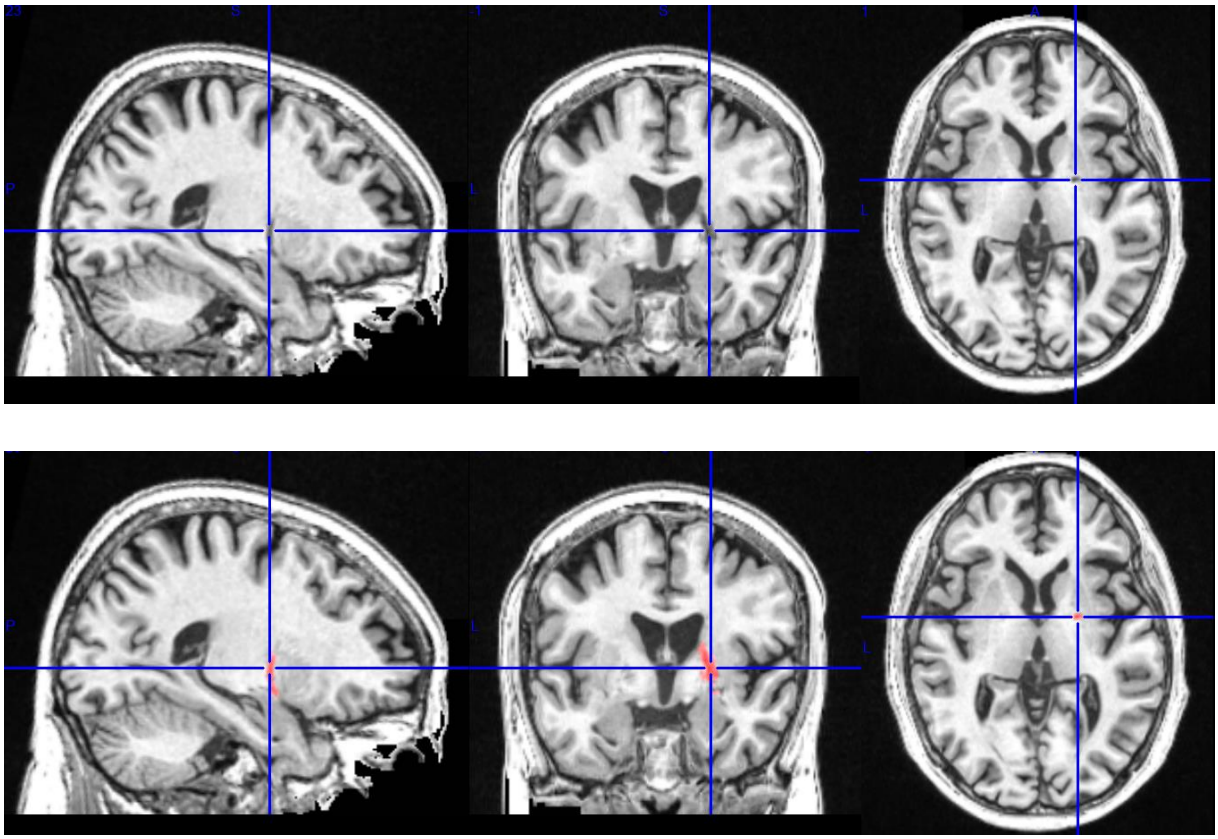


Figure 3.1 Localisation d'une lésion à l'aide d'un masque sur MRIcron.

b Localisation probabiliste des lésions

Les chercheurs qui ont constitué la base de données ATLAS nous ont également fourni une illustration qui montre la probabilité de présence de lésion dans le volume, rouge étant élevé et le bleu inversement. Cette illustration ne concerne que les 229 patients présents dans la base de données.

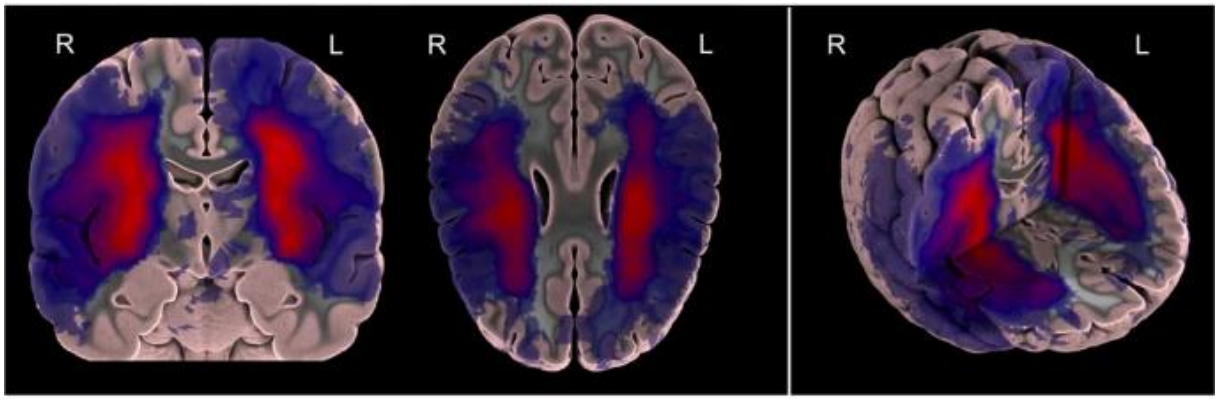


Figure 3.2 Distribution Probabiliste des lésions du dataset ATLAS [2].

c Limitations de la base de données

- Premier obstacle :

Cette base de données ne comporte que des cerveaux lésés, aucun patient sain n'est présent. L'apprentissage supervisé a impérativement besoin d'un grand nombre d'échantillon des deux classes (saines et lésées).

- Deuxième obstacle :

Le nombre d'échantillons total est très faible dans le contexte de l'apprentissage profond (n=229), il faut aussi noter que seul 70% sont réservés à l'entrainement alors que les 30% restants sont dédiés à la validation.

- Troisième obstacle :

Les images fournis par ATLAS ont subi un pré-traitement, nous ne pouvons donc pas ajouter d'autres images même si le traitement se fait d'une manière similaire; l'ajout d'une image dans la base de données alors qu'elle n'a pas subi un traitement parfaitement identique aux autres créera un biais qui agira sur l'apprentissage de notre modèle et par la même occasion faussera nos résultats.

Exemple : Les différences de traitement entre le traitement de ATLAS et le nôtre sera interprété comme des caractéristiques de l'image ; une différence de normalisation de la luminosité entre les deux traitements sera interprétée par notre modèle comme la différence de couleur entre un citron et une orange (en gray scale).

Conclusion : L'ajout d'échantillons à partir d'une autre source autre qu'ATLAS est déconseillé.

d Solution proposée

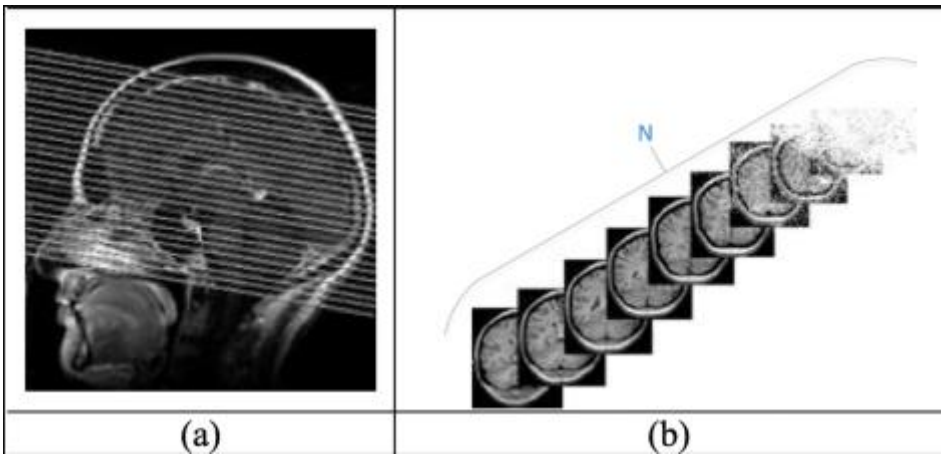


Figure 3.3 Découpage d'une image IRM 3D (a) en coupes d'images IRM 2D (b) [15].

Une image IRM 3D cérébrale est composée d'un ensemble de N (190 dans notre cas) coupes, l'idée est de travailler avec l'ensemble des images 2D qui composent l'image 3D.

Avantages

- Augmente grandement notre nombre d'échantillons total. En effet, même si plusieurs coupes proviennent du même patient, aucun slice n'est identique à un autre. D'ailleurs, même les images 2D d'une même lésion sont différentes, sauf dans le cas d'une lésion parfaitement cylindrique et parallèle à notre axe. Notre modèle interprètera deux coupes d'une lésion provenant du même patient comme deux lésions sans rapport.

- En absence d'image 3D d'un cerveau sain, notre modèle ne pourra pas apprendre les caractéristiques de cette classe, mais en récupérant les coupes 2D saines de chaque cerveau et en les combinant, notre modèle pourra apprendre les caractéristiques d'une coupe saine du cerveau.

- L'introduction des images 3D biaisées dans notre base de données n'est plus nécessaire, la base de données ATLAS pourrait nous suffit à présent.

- Inconvénients :

- Le nouveaux nombre d'images 2D est beaucoup trop élevée et prendra énormément de temps à itérer. Car 229 images X 3 axes X 190 coupes = 130530 images 2D, il faut trouver

un compromis entre le nombre d'images et les informations qu'elles donnent afin de réduire le temps de calcul du programme.

e Elimination des images non ou peu informatives

Afin de réduire le nombre d'images dans la base de données, on ne va considérer que les coupes horizontales.

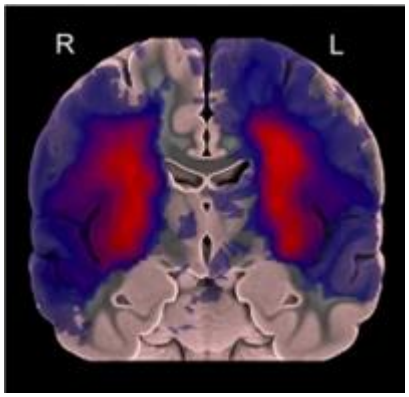


Figure 3.30. Distribution Probabiliste des lésions du dataset ATLAS [2].

D'après l'illustration ci-dessus de la figure 3.30, on remarque que certaines régions n'ont pas ou ont une très faible probabilité d'occurrence des lésions (la couleur violette exprime une occurrence d'une seule fois seulement). Elles peuvent donc être négligées dans le cadre de cette étude d'essai. L'élimination de ces coupes va non seulement réduire le temps de traitement et apprentissage pour chaque itération mais réduira aussi le nombre de caractéristiques à apprendre.

- Les coupes de 0 à 14 sont éliminées pour aucune ou peu de probabilité de lésion.
- Les coupes de 15 à 149 sont utilisées.
- Les coupes de 150 à 160 sont éliminées pour aucune ou peu de probabilité de lésion et aussi car la surface du cerveau est très petite.
- Les coupes de 160 à 189 sont éliminées car ils ne représentent que des images noires (les coupes sont en dehors du cerveau)

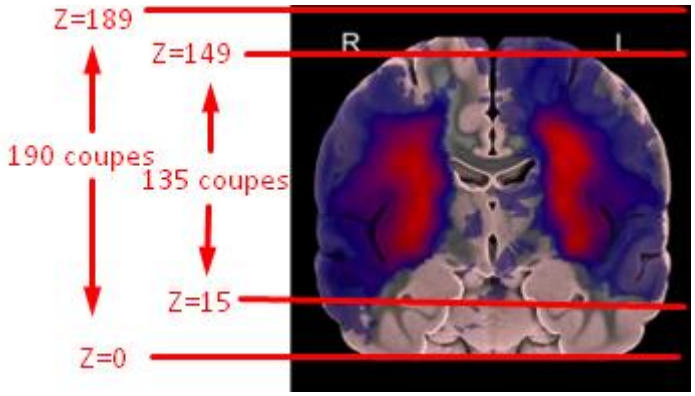


Figure 3.4 Représentation des coupes du cerveau.

Le nombre de coupes par cerveau passe de 189 à 135, et le nombre d'images 2D total passe de 130530 images pour les trois axes à 135 X 229 **= 30915** images 2D pour un seul axes.

f Classification manuelle des images

L'apprentissage supervisé est binaire dans notre cas, soit l'image ne représente aucune lésion est donc saine, soit l'image représente une ou plusieurs lésions est donc lésée.

L'ensemble des images fournies au modèle doivent être donc séparées en ces deux classes, la procédure manuelle est la suivante :

- Découpage de l'image 3D initial en 189 coupes.
- Suppression des coupes 0 à 14 et 150 à 189.
- Observation de l'image 3D initial ainsi que son ou ses masques.
- Attribution des coupes qui ne représente aucune lésion à la classe « sains ».
- Attribution des coupes qui représente une ou plusieurs lésions à la classe « lésés ».

Note : Si au moins un seul pixel est marqué comme lésé dans le masque fournis par ATLAS la coupe est considérée comme lésé.

g Finition de la base de données

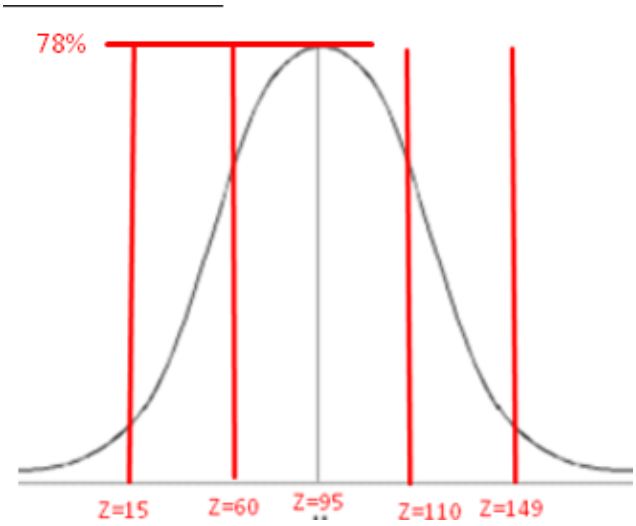


Figure 3.5 Représentation Gaussienne de présence de lésion par coupe.

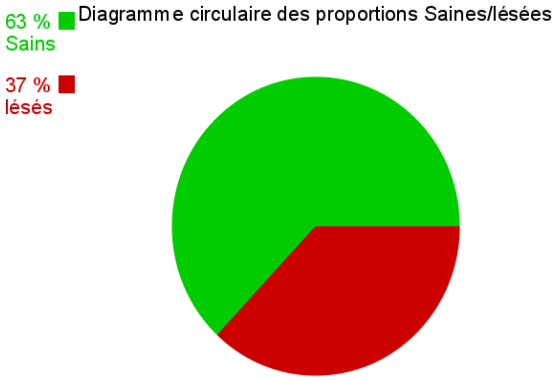


Figure 3.6 Diagramme circulaire des proportions saines / lésées.

Afin d'éviter que le point faible de ces expériences soit la base de données, Il faut s'assurer que les différents cas et classes soient bien représentées ; il faut qu'ils soient statistiquement équiprobables. Pour cela, la procédure utilisée est la suivante :

- Suppression des images saines trop redondantes (exemple de Z=15 à Z= 45) ; pour réduire la proportion des images 2D saines et en même temps réduire le temps de calcul et éviter le surajustement.

- Multiplication des images saines peu présentes dans l'ensemble (exemple de Z=80 à Z=105) ; pour contrebalancer la présence dominante des images lésés à ces cordonnées.
- Multiplication des images lésées peu présentes dans l'ensemble (exemple de Z=15 à Z=50 et de Z=120 à Z=149.

Note : nous ne pouvons pas supprimer les images lésées très présentes de Z=80 à Z=105 car chacune est unique et apporte une différente information (taille, coordonnées X/Y) ; les supprimer pénalisera énormément la capacité de notre modèle à généraliser.

Afin de maximiser la généralisation du modèle, nous avons ajouté à l'ensemble une image inversée par rapport à l'ase verticale (miroitée) de chaque image. Le but est de simuler l'ajour de patients ayant des lésions dans le côté opposé du cerveau.

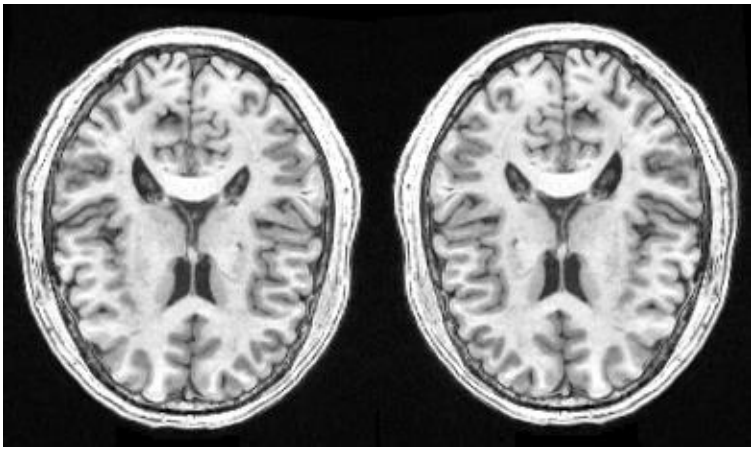


Figure 3.7 Exemple d'une image miroitée.

Enfin, dans le but de maximiser la netteté des images, nous avons utilisé un outil informatique de réduction du bruit **cv.fastNlMeansDenoising()** de la bibliothèque OpenCV (dé-bruitage rapide par moyenne). Comme son nom l'indique, cette fonction réduit le bruit en moyennant les pixels dans des fenêtres données en éliminant les détails fins qui sont souvent du bruit. Il est à noter que cet outil peut très légèrement altérer les lésions, et dans les cas extrêmes, en masquer une (on donnera plus de détails dans hypothèses).

La nouvelle base de données est à présent prête à être utilisée. Cette dernière comporte un total de 24115 images 2D saines et 24114 lésées. Elle sera devisée en 40000 images d'entrainement et 8229 images de validation. Soit un ratio de 80% entrainement et 20% validation. Chaque ensemble compose 50% d'images saines et 50% d'images lésées.

3.4.2 Le Programme

a Initialisation du programme

Bibliothèques utilisées :

Numpy, tensorflow, os, tensorboard et sklearn.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten,\
    BatchNormalization, Conv2D, MaxPool2D, Dropout,\
    AveragePooling2D, LeakyReLU, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import regularizers
import os
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Aiguillage des répertoires de la base de données :

```
train_path = 'C:/Users/21366/Desktop/PYTHON PROJECT/train3'
valid_path = 'C:/Users/21366/Desktop/PYTHON PROJECT/valid3'
```

Ci-dessous, la valeur numérique de chaque pixel est transformée de (0-255) à (0-1).

(Cela est conventionnel et permet un meilleur calcul des pertes pas le modèle)

```
train_batches = ImageDataGenerator(
    rescale=1./255,
)
valid_batches = ImageDataGenerator(
    rescale=1./255,
)
```

Chargement des images dans le programme :

```
valid_generator = valid_batches.flow_from_directory(
    directory=valid_path,
    class_mode="binary",
    classes=['healthy', 'sickly'],
    color_mode="grayscale",
    batch_size=32,
    target_size=(107, 89),
    shuffle=True,)
train_generator = train_batches.flow_from_directory(
    directory=train_path,
    class_mode="binary",
    classes=['healthy', 'sickly'],
    color_mode="grayscale",
    batch_size=32,
    target_size=(107, 89),
    shuffle=True,)
```

Les images sont nativement dimensionnées en 233 pixels de hauteurs par 197 pixels de largeur, afin de réduire le temps de calcul qui augmente avec le nombre de fenêtres de calcules, les images se font redimensionnées en 107x89 pixels.

Le paramètre shuffle mélange les images présentes dans l'ensemble, afin d'éviter les séries d'images appartenant au même patient et/ou la même classe.

Le paramètre batch-size indique le nombre d'image à traiter simultanément par le modèle. Cela utilise le pipe-line du hardware pour traiter plusieurs images en même temps afin de réduire le temps d'itération.

b Modèle utilisé

Le modèle utilisé est un modèle CNN de 9-couches à 4-blocs classique.

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(107, 89, 1)),
    MaxPool2D(pool_size=(2, 2)),

    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),

    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),

    Flatten(),
    Dense(units=256, activation='relu'),
    Dense(units=1, activation='sigmoid')
])
```

Figure 3.8 Modèle utilisé dans la première approche.

La loi d'activation utilisée pour la dernière couche est « sigmoïde » car elle est optimale dans les classifications binaires.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 107, 89, 32)       320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 53, 44, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 53, 44, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 26, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 22, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 11, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 18304)             0
_________________________________________________________________
dense (Dense)                (None, 256)               4686080
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 4,779,009
Trainable params: 4,779,009
Non-trainable params: 0
```

Figure 3.9 Sommaire de la première approche.

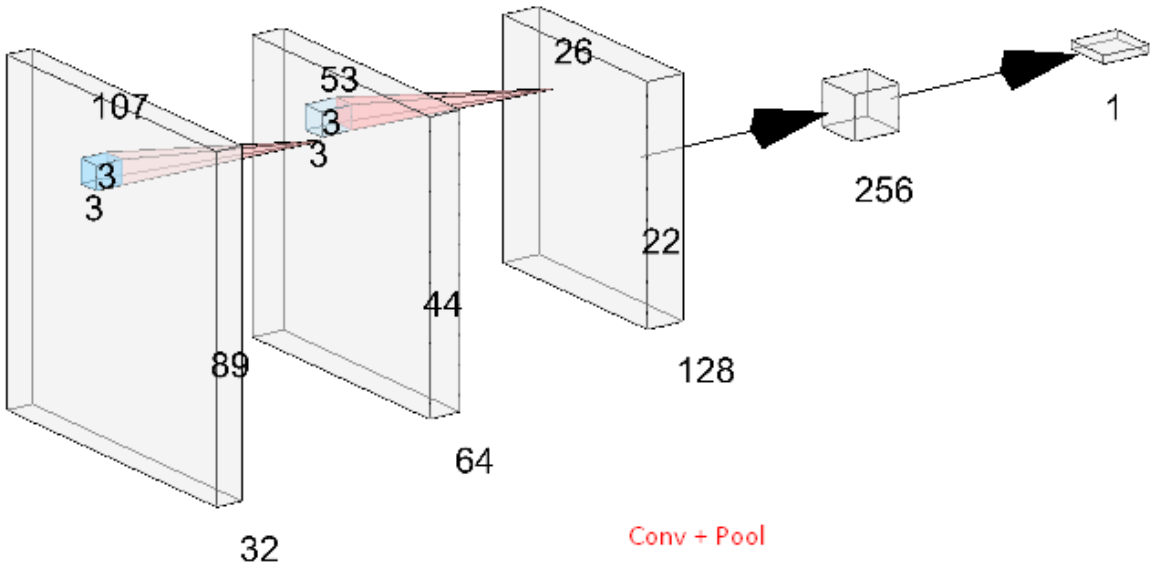


Figure 3.10 Schéma représentatif de l'architecture de la première approche.

c Variables d'apprentissage du modèle

```
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.001), metrics=['accuracy'])
model.fit(
    shuffle=True,
    x=train_generator,
    steps_per_epoch=len(train_generator),
    validation_data=valid_generator,
    validation_steps=len(valid_generator),
    epochs=10,
    verbose=1
)
```

Figure 3.11 Hyper-paramètres de la première approche.

L'optimisateur choisi est **Adam et binary_crossentropy** pour le calcul de pertes car notre classification est binaire.

Tous les paramètres sont conventionnels avec un pas d'apprentissage de 10^{-3}.

Un second mélange des images augmente le désordre pour une meilleur moyenne de lot et pour éviter les séries d'images de même classe et/ou de même patient à l'entrainement.

3.4.3 Discussion et Hypothèses

- Un modèle CNN utilise des caractéristiques clés comme les formes, les angles, le différentiel de couleurs, les bordures etc. pour la classification, en utilisant ces architectures dans une classification binaire où les deux classes sont presque identiques, et dans certains cas à un pixel prés, nous utilisons le modèle dans une application où il n'y est pas adapté. Une bonne performance n'est donc pas attendue.

-Si nous supposons que l'espace volumique d'une lésion dans le cerveau est ressemblant à celle d'une sphère, nous constaterons alors que les coupes tangentielles ou proches des extrémités supérieures et inférieures de la lésion sont caractérisées par des surfaces lésées extrêmement petites :

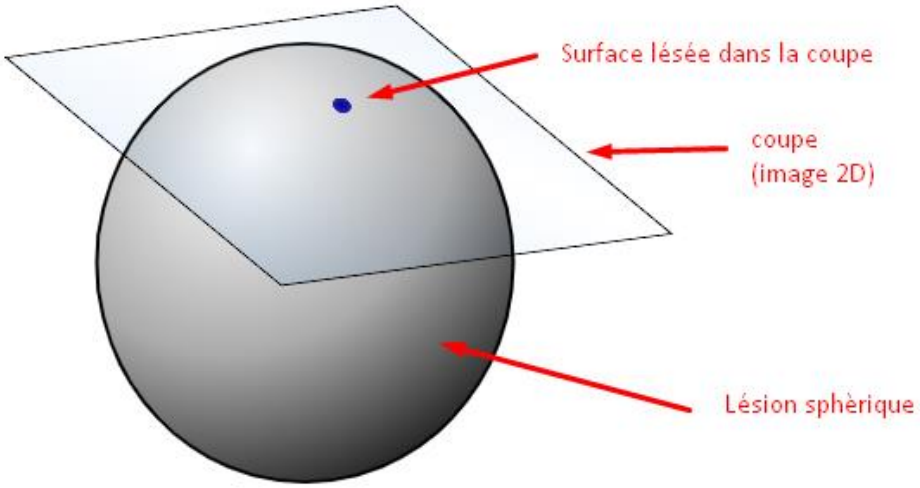


Figure 3.12 Exemple d'une coupe tangentielle d'une lésion sphérique.

Exemple réel :

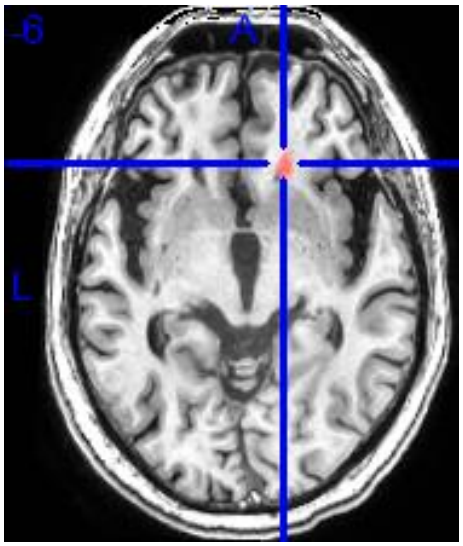


Figure 3.13 Coupe non-tangentielle de la lésion (Z=67).

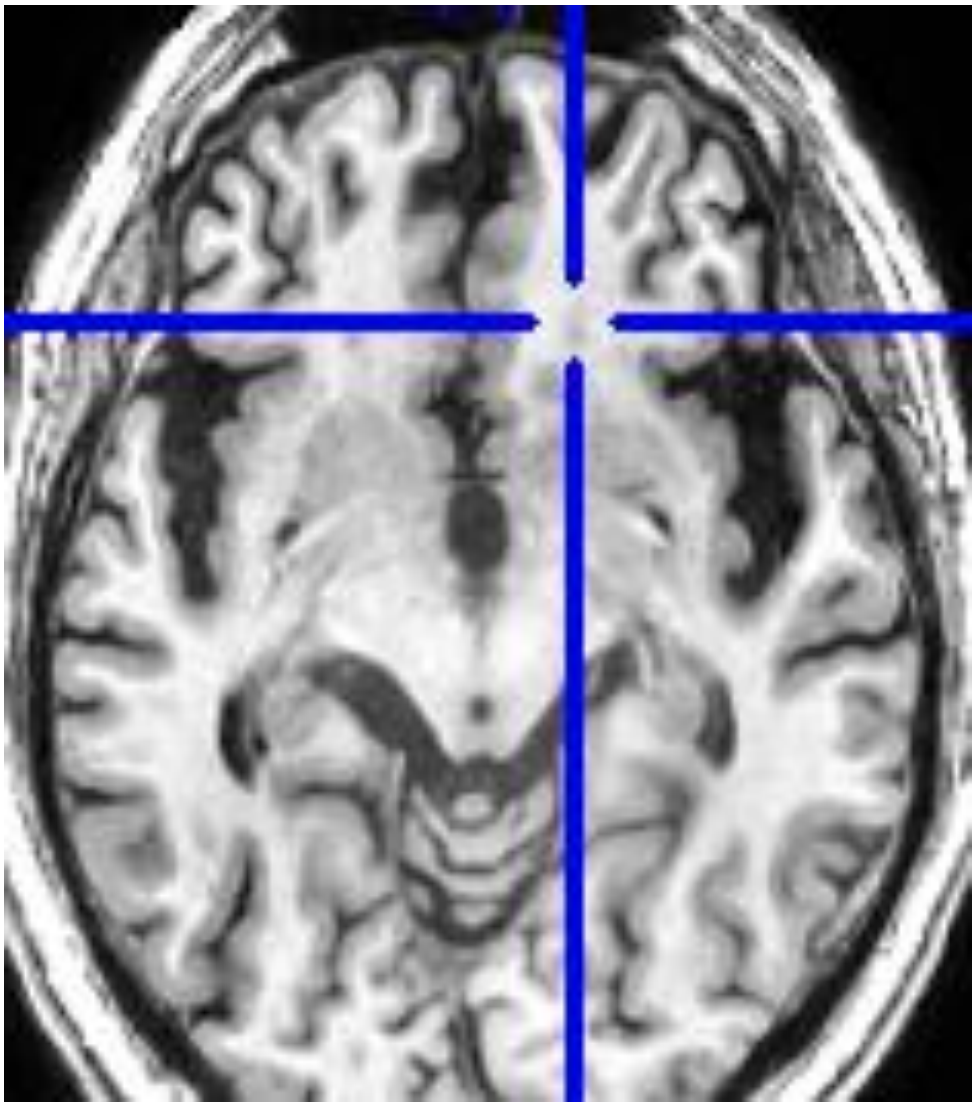


Figure 3.14 Coupe tangentielle de la même lésion (Z=64).

Dans ce cas, la coupe (Z=64) est classée comme lésée dans notre ensemble d'entrainement, même si, la surface lésée est plus petite que la fenêtre de convolution. Non seulement la lésion est quasiment invisible pour le modèle, mais elle contribuera à une baisse non-négligeable dans la précision car au moins deux coupes tangentielles sont présentes dans une image IRM 3D du patient.

Supprimer les coupes tangentielles d'une lésion ou changer leur classe a déjà été envisagé, mais cela non seulement biaisera nos résultats, mais nous donnerait une fausse perception de la capacité de notre modèle à identifier les coupes lésées dans un véritable scénario.

Il est aussi à noter que la procédure de réduction du bruit **cv.fastNlMeansDenoising()** peut aussi affecter les petites surfaces lésées en les considérant comme bruit ou détail fin.

-Les variables du modèle ne sont pas optimaux.

-les images de la base de données sont trop similaires et peuvent faire entrer le modèle en surajustement très rapidement.

-Le modèle n'est pas assez bien structuré.

-Les images de la base de données sont peu diversifiées et ne permettent pas au modèle de généraliser.

3.5 Deuxième approche

3.5.1 Modifications apportées à la base de données

-Les images inversées ont été retirées. Afin d'éviter un éventuel surajustement.

-Chaque image restante a été rendue négative (inversement de la luminosité), l'idée est de simuler les images en T2 plutôt qu'en T1, il est probable que cela ne changera pas la perception de notre modèle aux lésions, mais il est certain que cela le fera pour nous ; il est

plus facile pour les yeux humains de distinguer le blanc dans un plan noir que l'inverse, et c'est d'ailleurs pourquoi les IRM en T2 sont plus populaire que celles en T1 [16].

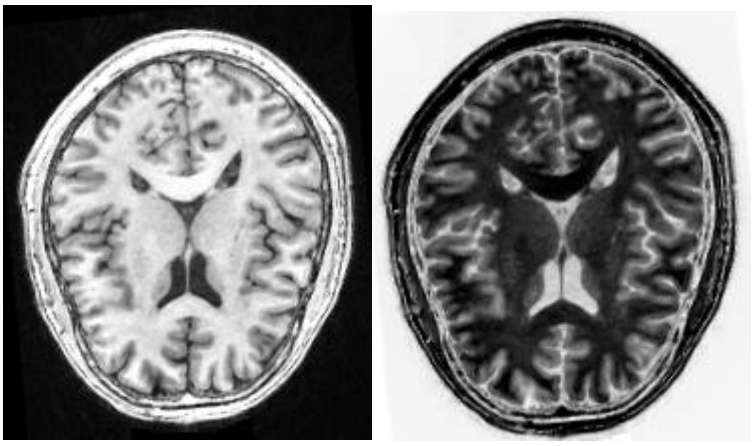


Figure 3.15 Exemple d'une image inversée en luminosité.

3.5.2 Modifications apportées au programme

Ajout de la fonction d'augmentation de données.

```
train_batches = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
valid_batches = ImageDataGenerator(
    rescale=1./255,
)
```

Figure 3.16 Fonction d'augmentation de données.

L'outil de choix contre le surapprentissage et la non-généralisation des prédictions.

3.5.3 Modifications apportées au modèle

Le modèle utilisé est un modèle CNN de 12-couches à 5-blocs classiques.

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(107, 89, 1)),
    MaxPool2D(pool_size=(2, 2)),

    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),

    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),

    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),

    Flatten(),
    Dense(units=512, activation='relu'),
    Dense(units=1, activation='sigmoid')
])
```

Figure 3.17 Modèle utilisé dans la deuxième approche.

Modèle utilisé dans la recherche [17] qui a permis une précision de 97% dans la classification de la maladie d'Alzheimer (base de données OASIS).

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 107, 89, 32)       320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 53, 44, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 53, 44, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 26, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 22, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 11, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 11, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 5, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 7680)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3932672
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,321,025
Trainable params: 4,321,025
Non-trainable params: 0
```

Figure 3.18 Sommaire de la deuxième approche.

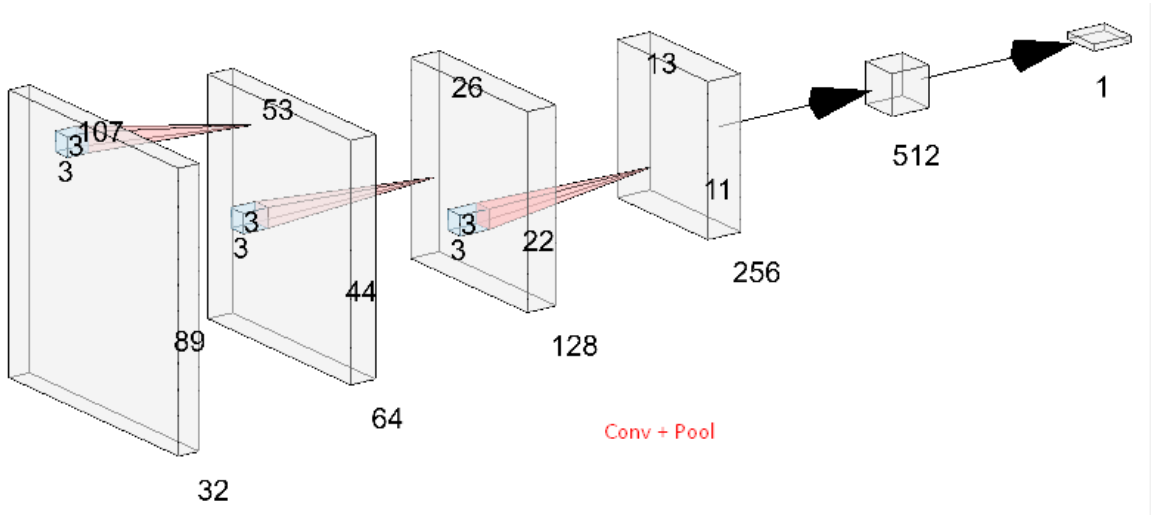


Figure 3.19 Schéma représentatif de l'architecture de la deuxième approche.

3.5.4 Modification apportées aux variables (hyper-paramètres)

```
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.00001), metrics=['accuracy'])
```

Un plus petit pat d'apprentissage permettra un meilleur calcul des minimums locaux et donc une meilleure convergence, au détriment du temps, mais cela signifie aussi que notre modèle prendra plus de temps pour diverger et entrer en surajustement.

3.5.5 Discussions et Hypothèses

-Les variables (hyper-paramètres) du modèle ne sont pas optimaux.

-les images de la base de données sont trop similaires et peuvent faire entrer le modèle en surajustement très rapidement.

3.6 Troisième approche : Tranfert-learning et fine tuning

3.6.1 Avec VGG16

```
from tensorflow.keras.applications.vgg16 import VGG16
```

Prétraitement des images suivant VGG16 (le paramètre le plus important est l'accentuation du contraste).

```
train_batches = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg16.preprocess_input, rescale=1/255.
).flow_from_directory(
    directory=train_path,
    target_size=(107, 89),
    classes=['healthy', 'sickly'],
    batch_size=20,
    shuffle=True,
    class_mode='binary'
)
valid_batches = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg16.preprocess_input, rescale=1/255.
).flow_from_directory(
    directory=valid_path,
    target_size=(107, 89),
    classes=['healthy', 'sickly'],
    batch_size=20,
    shuffle=True,
    class_mode='binary'
)
```

Figure 3.20 Pré-traitement des images pour VGG16.

Apres l'importation du modèle VGG16, il sera chargé sans les couches de l'entrée et sortie, cela permettra l'ajout des couches qui nous convient. Nous désactiverons l'apprentissage des couches car les poids synaptiques sont importés avec le modèle. Le modèle VGG16 utilise les 3 chaines de couleur, les images ne sont donc pas en grayscale.

```
IMAGE_SIZE =[107, 89]
vgg16_model = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
model = Sequential()
for layer in vgg16_model.layers[:-1]:
    model.add(layer)

for layer in model.layers:
    layer.trainable = False

Flatten(),
Dense(units=1, activation='sigmoid')
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.001), metrics=['accuracy'])
model.summary()
model.fit(
    shuffle=True,
    x=train_batches,
    steps_per_epoch=len(train_batches),
    validation_data=valid_batches,
    validation_steps=len(valid_batches),
    epochs=10,
    verbose=1
)
```

Figure 3.21 Chargement du modèle VGG16.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 107, 89, 64)       1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 107, 89, 64)       36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 53, 44, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 53, 44, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 53, 44, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 26, 22, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 26, 22, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 26, 22, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 26, 22, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 13, 11, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 13, 11, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 13, 11, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 13, 11, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 6, 5, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 6, 5, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 6, 5, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 6, 5, 512)         2359808
_________________________________________________________________
flatten (Flatten)            (None, 15360)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 15361
=================================================================
Total params: 14,730,049
Trainable params: 15,361
Non-trainable params: 14,714,688
```

Figure 3.22 Sommaire du modèle VGG16.

3.6.2 Avec InceptionV3

La procédure est quasiment identique à la précédente.

Importation et chargement du modèle InceptionV3

Il faut noter que ces importations comportent l'architecture, les hyper-paramètres et les poids.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
```

Augmentation de données, pré-traitement des images.

```
train_datagen = ImageDataGenerator(
      preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
      rotation_range=20,
      width_shift_range=0.1,
      height_shift_range=0.1,
      shear_range=0.1,
      zoom_range=0.1,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.inception_v3.preprocess_input)

train_generator = train_datagen.flow_from_directory(
      train_path,
      target_size=(107, 89),
      batch_size=32,
      class_mode='binary')

valid_generator = validation_datagen.flow_from_directory(
      valid_path,
      target_size=(107, 89),
      batch_size=32,
      class_mode='binary')
```

Figure 3.23 Pré-traitement des images pour InceptionV3.

Chargement du modèle, désactivation de changement des poids synaptiques, ajout d'une couche d'entrée et de sortie suivant nos besoins.

```
modelv3 = InceptionV3(input_shape=(107, 89, 3), weights='imagenet', include_top=False)

modelv3.trainable = False
x = layers.Flatten()(modelv3.output)
x = layers.Dense(1, activation='sigmoid')(x)

model = Model(modelv3.input, x)

model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0001), metrics=['accuracy'])
model.summary()
model.fit(
    shuffle=True,
    x=train_generator,
    steps_per_epoch=len(train_generator),
    validation_data=valid_generator,
    validation_steps=len(valid_generator),
    epochs=10,
    verbose=1
)
```

Figure 3.24 Chargement du modèle InceptionV3.

Le sommaire du modèle est beaucoup trop long pour l'afficher.

3.6.3 Discussion et Hypothèses

-Les modèles VGG16 et InceptionV3 sont tous les deux entrainés sur des millions d'images en les classifiant en 1000 classes différentes, ces modèles n'ont pas été entrainés pour distinguer entre deux classes similaires avec une différence de quelques pixels mais entre des images qui n'ont absolument aucun lien les unes les autres comme un chat et une voiture.

-Des modèles aussi complexes ne sont peut-être pas adaptés pour des classifications binaires.

-les trois approches précédentes ont bien était testé avec une base de données de contrôle (chiens contre chats) et les résultats était bien satisfaisants (+95% de validation), le problème est donc bien lié à notre base de données.

La caractéristique majeure qui différencie notre base de données des autres est que les images 2D se ressemblent énormément, il est vrai qu'une coupe de la partie supérieure du cerveau est différente de l'inférieur mais la répétition reste présente.

Dans les 30000 images 2D uniques, appartenant à une des 135 coupes possibles, nous déduisons qu'en moyenne, il existe 222 images appartenant à la même coupe, et donc, se ressemble énormément, et ceci, pour chaque hauteur de coupe (pour chaque coordonnée Z).

3.7 Quatrième approche

3.7.1 Modifications apportées à la base de données

-L'inversion des luminosités est annulée, les images sont donc des T1.

-A cause de la similarité entre les images saines et dans le but de réduire le surapprentissage, il a été décidé donc de réduire drastiquement la taille de l'ensemble des images saines d'entrainement tout en gardant les images lésées, ceci est contre-intuitif car il faut généralement respecter l'égalité des proportions des classes sauf qu'à cause de l'application inconventionnel du modèle CNN dans une telle base de données, cela a un certain sens.

La base de données comporte deux ensembles dont 3500 images saines et 8260 images lésées pour l'entrainement ainsi que 1500 images saines et 1500 images lésés pour la validation.

L'ensemble d'entrainement est donc disproportionné (30% images saines contre 70% d'images lésés). Alors que l'ensemble de validation est toujours bien proportionné (50% d'images saines pour 50% d'images lésés). Les résultats donnés pas notre modèle ne seront donc pas biaisés.

3.7.2 Modification apportée au programme

-Accentuation des valeurs des variables (hyper-paramètres) d'augmentation de données.

En effet, donner des valeurs plus importantes pour les propriétés de modification des images permettra d'introduire des images plus 'distinguées' et plus 'différentes'

```
train_batches = ImageDataGenerator(
    rescale=1./255,
    rotation_range=90,
    width_shift_range=0.27,
    height_shift_range=0.27,
    shear_range=0.27,
    zoom_range=0.27,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

Figure 3.25 Augmentation de données utilisée.

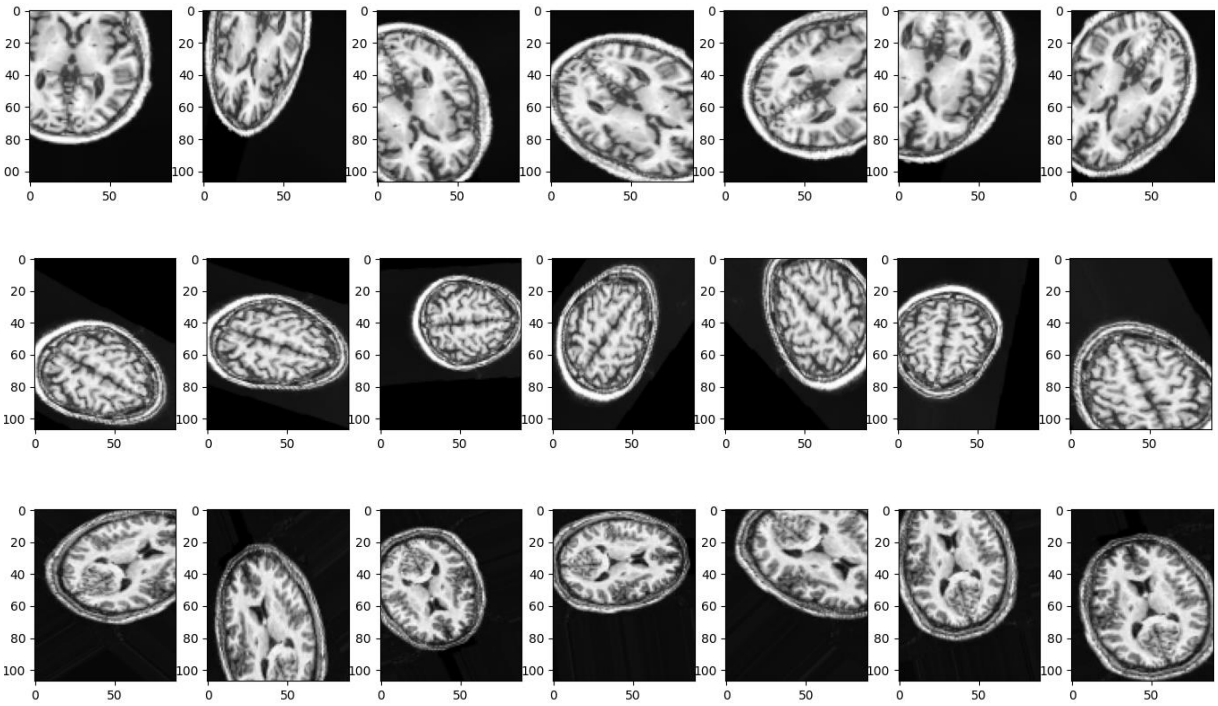


Figure 3.26 Exemple des résultats de l'augmentation de données utilisée.

3.7.3 Modifications apportées au modèle

-L'ajout du Dropout, qui est l'outil de choix pour réduire le surajustement sans augmenter le temps de calcul.

-L'ajout de la régularisation L2 qui elle aussi est un outil de réduction de surapprentissage, mais qui augmente le temps de calcul.

-L'ajout de l'arrêt prématuré, cet outil arrêtera l'apprentissage automatiquement si le paramètre sélectionné augmente ou diminue (dans notre cas, si la perte de validation augmente.

```
model.fit(
    shuffle=True,
    x=train_generator,
    steps_per_epoch=len(train_generator),
    validation_data=valid_generator,
    validation_steps=len(valid_generator),
    epochs=30,
    verbose=1,
    callbacks=[early_stopping]
)
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=1, mode="min")

model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(107, 89, 1)),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(0.1),

    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(0.1),

    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(0.1),

    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(0.1),

    Flatten(),
    Dense(units=512, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.4),

    Dense(units=1, activation='sigmoid')
])
```

Figure 3.27 Modèle utilisé dans la quatrième approche.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 107, 89, 32)       320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 53, 44, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 53, 44, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 53, 44, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 26, 22, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 26, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 22, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 11, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 13, 11, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 11, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 5, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 5, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 7680)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3932672
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,321,025
Trainable params: 4,321,025
Non-trainable params: 0
```

Figure 3.28 Sommaire de la quatrième approche.

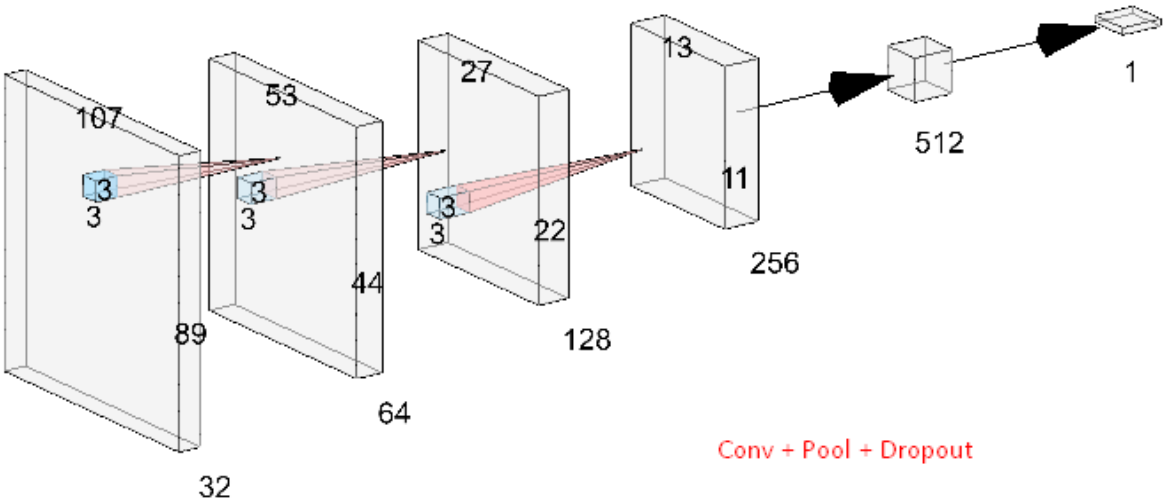


Figure 3.29 Schéma représentatif de l'architecture de la quatrième approche.

3.7.4 Discussion et hypothèses

Comme la convergence d'un modèle est logarithmique, et comme le résultat de cette dernière expérience est satisfaisant, il sera de plus en plus difficile d'augmenter le pourcentage de validation au fur et à mesure que nous nous rapprochons des 100%, il est donc peu probable qu'on puisse dépasser le taux de validation de cette dernière expérience. Et cela probablement toujours à cause des coupes tangentielles toujours présentes dans notre base de données.

3.8 Conclusion

Nous avons utilisé seulement quatre approches dans notre étude expérimentale mais il reste certain qu'aucun de nos modèles n'est optimal, on peut toujours théoriser les hyper-paramètres idéaux et les utiliser, nous pourrons aussi empiriquement chercher les meilleurs variables à rentrer, mais nous finirons toujours par atteindre un certain seuil où la légère amélioration des performances ne justifie pas l'effort nécessaire à fournir.

Chapitre 4

Résultats et discussion

4.1 Introduction

Dans ce chapitre, nous nous intéresserons aux résultats de chaque approche et modèle, nous commenterons aussi leurs propriétés et nous poserons quelques hypothèses qui justifient ces résultats. Nous comparerons par la suite ces derniers dans les deux cas, avec utilisation ou non de la même base de données.

4.2 Résultats et commentaires

4.2.1 Première approche

```
- loss: 0.0772 - accuracy: 0.9713 - auc: 0.9966 - val_loss: 0.8608 - val_accuracy: 0.6572 - val_auc: 0.7336
```

Apres convergence du modèle, nous constatons que ce dernier est en surajustement car la précision de l'ensemble d'entrainement est à 97% alors que celle de la validation (ou test) n'est que de 66%, ce qui est toujours mieux qu'avoir recours à un tirage aléatoire (50% comme notre ensemble de validation est composé de 1500 images saines et de 1500 images lésées). Il est évident qu'une précision aussi basse n'est absolument pas acceptable dans un contexte médical. Ce modèle n'arrive pas à généraliser.

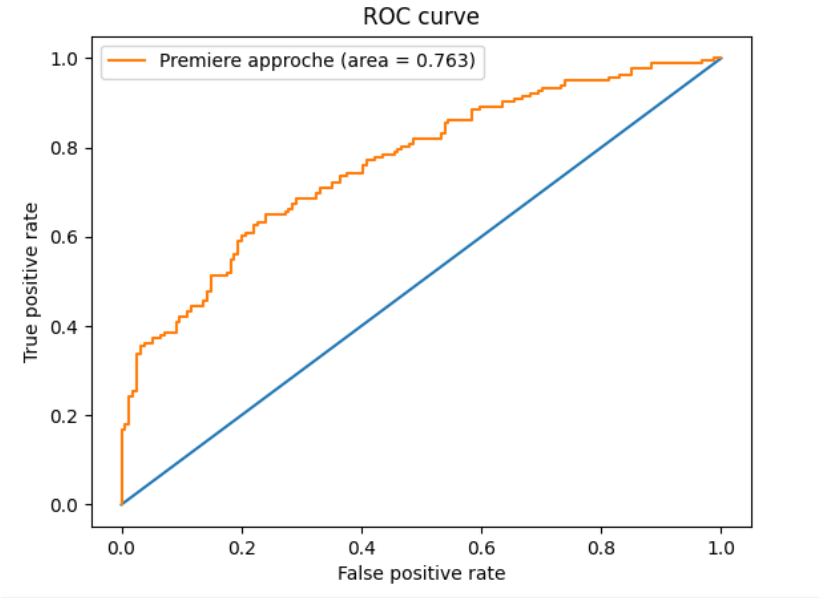


Figure 4.3 Courbe ROC de la première approche.

Les raisons probables et hypothétiques de ce surajustement ont déjà été détaillées dans la partie étude expérimentale et sont résumés par ce qui suit :

- La petite taille des lésions.
- Le mauvais traitement de la base de données.
- Le modèle est trop « simple ».
- Les variables « hyper-paramètres » sont mal sélectionnées.

Note : l'AUC en sortie du modèle et dans le graphique varie car ils sont calculés par deux bibliothèques différentes et en temps différé (tensorflow calcule l'AUC en temps réel avec les autres sorties de mesure comme « accuracy » alors que sklearn intervient en fin de programme après la fin d'exécution du modèle).

4.2.2 Deuxième approche

```
545s 11s/step - loss: 0.2320 - accuracy: 0.9002 - auc: 0.9638 - val_loss: 1.1401 - val_accuracy: 0.6272 - val_auc: 0.7913
```

Après la fin de l'entrainement, nous remarquons que la précision d'entrainement n'est que de 90% cette fois, cela revient forcément à l'ajout de l'augmentation de données qui à chaque fois, modifie les images de l'ensemble d'entrainement pour simuler une base de données plus grande et diversifiée, nous remarquons aussi une baisse (de 66% à 63%) de la précision de validation (de test) suite à cet ajout, ainsi que la modification du modèle.

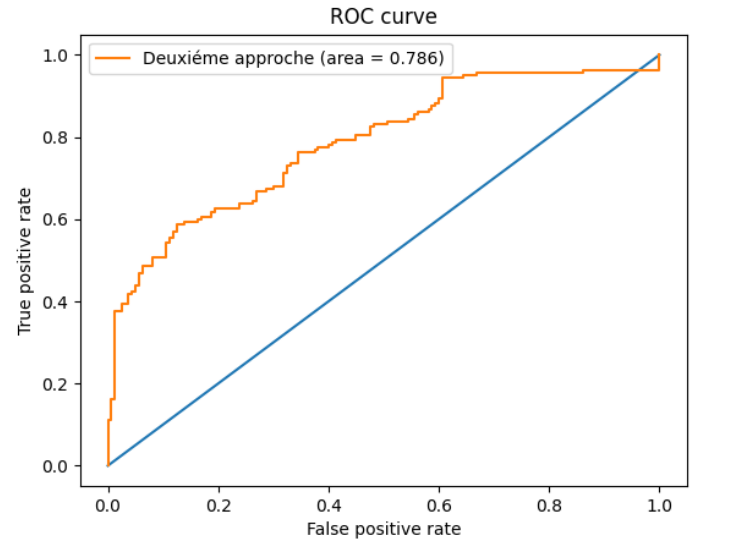


Figure 4.4 Courbe ROC de la deuxième approche.

Une légère amélioration dans les performances ROC et AUC de notre modèle mais toujours pas suffisant. Les raisons qui restent valables sont les suivantes :

- La petite taille des lésions.
- Le mauvais traitement de la base de données.
- Les variables « hyper-paramètres » sont mal sélectionnées.

Vu le principe de fonctionnement de l'architecture CNN (fenêtres de convolution 3x3 + pooling 2x2) et aussi le besoin de savoir comment le modèle réagit à tous les types de lésions, le problème des petites lésions est donc intraitable à ce niveau.

4.2.3 Transerft learning et Fine-tuning

a VGG16

```
loss: 0.2206 - accuracy: 0.9094 - auc: 0.9756 - val_loss: 0.9938 - val_accuracy: 0.6238 - val_auc: 0.7312
```

Le modèle VGG16 est un modèle avec 1000 classes complétement différentes en sortie, c'est un modèle qui a été entrainé pour classifier des centaines de choses sans aucune relation entre elles mais avec une précision très élevée. Il a été donc prévisible que ce modèle allait répondre très pauvrement car il n'y est pas adapté, cependant les résultats et performances sont relativement identiques à nos deux approches précédentes.

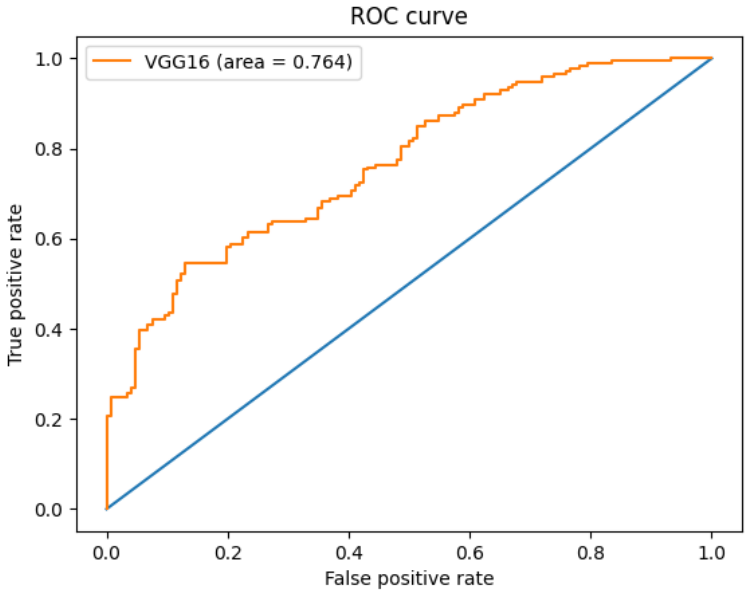


Figure 4.5 Courbe ROC de l'approche VGG16.

b InceptionV3

```
loss: 0.4456 - accuracy: 0.7992 - auc: 0.8755 - val_loss: 0.5719 - val_accuracy: 0.7124 - val_auc: 0.7858
```

Avec une architecture complétement différente de VGG16, InceptionV3 répond généralement mieux que cette dernière.

Nous constatons que tous les modèles et architectures testés jusqu'à présent ne sont pas satisfaisants en termes de résultats et de performances dans un contexte médical, le point commun entre ces derniers est la base de données ; une modification de la base de données s'impose alors.

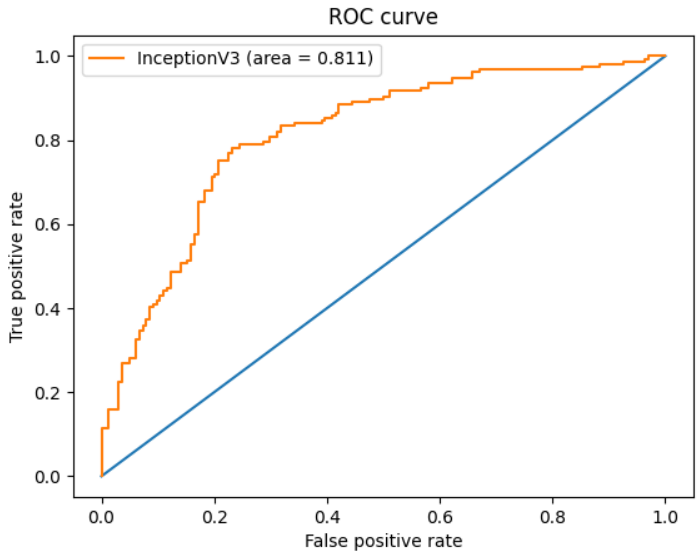


Figure 4.6 Courbe ROC de l'approche InceptionV3.

4.2.4 Quantième approche

```
loss: 0.4498 - accuracy: 0.7553 - auc: 0.8245 - val_loss: 0.3980 - val_accuracy: 0.9507 - val_auc: 0.9964
```

Après une convergence très rapide du modèle (5 epochs seulement), le modèle converge avec une précision d'entrainement de 75% seulement. Cela revient à l'accentuation de l'augmentation de données pour l'ensemble d'entrainement que nous venons d'introduire, mais en contrepartie cette modification a grandement amélioré notre précision de validation (test) ainsi que les performances de notre modèle et sa capacité de généralisation.

Il faut aussi noter que comme notre ensemble d'entrainement est asymétrique (70% images lésées pour 30% de saines), la précision d'entrainement est difficilement interprétable et peut-être même non-significative.

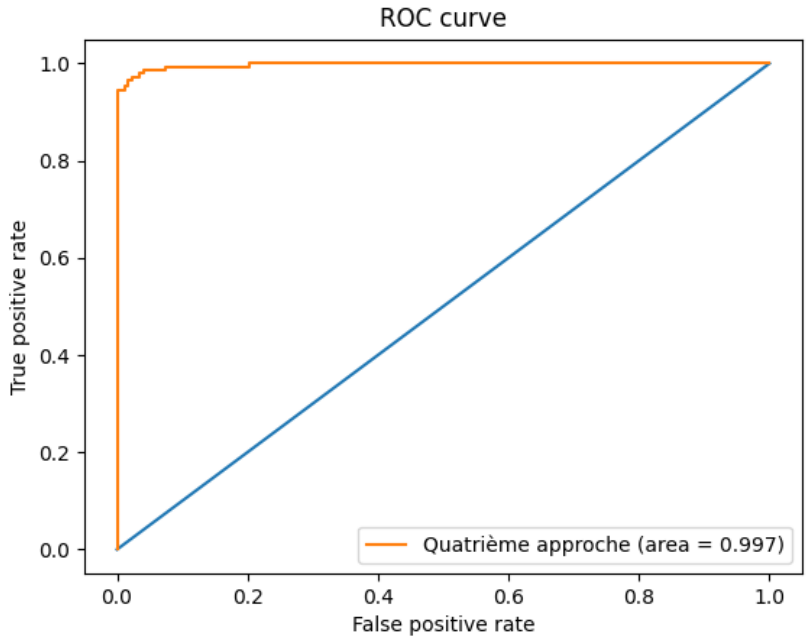


Figure 4.7 Courbe ROC de la quatrième approche.

Cette amélioration est en grande partie due à la réduction des images saines qui se ressemblent et entrainent donc très rapidement un surajustement, il est vrai qu'intuitivement et conventionnellement les proportions des nombres d'images doivent être égaux pour une bonne appréhension de notre modèle des différentes classes, sauf que dans le cas présent le surapprentissage arrive très vite à cause de la grande ressemblance entre les deux classes.

	Temps d'entrainement (par itération)	Précision d'entrainement	Précision de validation	Score AUC
Première approche	14min	97%	66%	0.75
Deuxième approche	17min	90%	63%	0.8
VGG16	32min	90%	62%	0.74
InceptionV3	22min	80%	71%	0.79
Quatrième	6min	76%	95%	0.996

Table 4.8 Table comparative des cinq modèles utilisés.

Note : Les bases de données ne sont pas identiques dans les différentes approches.

Il est clair que la quatrième approche est bien plus efficace, mais comparer les performances des modèles avec des bases de données différentes est presque dénué de sens, la prochaine table compare les modèles en utilisant la base de données de la quatrième approche.

	Temps d'entrainement (par itération)	Précision d'Entrainement	Précision de validation	Score AUC
Première approche	4min	93%	93%	0.98
Deuxième approche	6min	92%	94%	0.99
VGG16	14min	85%	75%	0.9
InceptionV3	8min	85%	82%	0.79
Quatrième approche	6min	76%	95%	0.996

Table 4.9 Table comparative des cinq modèles utilisés avec la même base de données.

Note : l'ensemble d'entrainement étant disproportionné (30/70), implique que la précision d'entrainement des cinq modèles est intuitivement mal comprise ou biaisée.

4.3 Discussion

Nous remarquons que chaque modèle répond mieux en utilisant une base de données asymétrique, nous remarquons surtout que la première et deuxième approche répondent

quasiment aussi bien que la quatrième ce qui indique que la modification de la base de données en la rendant asymétrique pour réduire les images qui se ressemblent est sans doute la cause de ces bons résultats. Les Dropouts ainsi que la régularisation L2 ont eu peu ou pas d'effet.

La performance et les résultats inférieurs des modèles VGG16 et InceptionV3 par rapport aux autres modèles peut être expliqué par la complexité de caractérisation de VGG16 et InceptionV3 car ces dernier ont été développés pour classifier des millions d'images en des centaines de classes, ils sont dont plus optimisés dans la recherche de caractéristiques spécifiques pour chaque image, alors que les autres modèles sont bien plus simples pour des images (coupes d'IRM du cerveau) qui n'ont pas beaucoup de caractéristiques différentes entre les classes (saines et lésées).

4.4 Conclusion

En conclusion notre modèle avec une précision de 95% est compétitif avec les autres modèles qui utilisent l'architecture U-net [18], [19], [20], et qui ont une précision globale de 84% à 97% selon les études suivantes [21], [22], [23]. Comme la sortie de ces architectures (prédiction de masque) sont différentes du CNN (prédiction de classe) les il est difficile de comparer leur résultats.

Conclusion

Même si notre modèle est binaire et donc peu informatif en comparaison avec les modèles utilisant l'architecture U-net (précision globale entre 84% et 97%) qui renvoient un masque de prédiction sur l'étendue de la lésion, il reste cependant clair que notre modèle est un outil utile et compétitif dans le diagnostic rapide d'un patient qui présente des signes ou symptômes d'accident vasculaire cérébral.

Le modèle a prouvé sa robustesse dans la classification de tous types de lésions avec une précision globale de 95%, il faut aussi noter que certaines images de la classe « lésées » présentent des lésions minuscules de quelques pixels uniquement et sont plus petits que la fenêtre de convolution qui sera suivie par un maxpooling qui risque davantage de réduire l'information.

Un taux de précision global de 95% (qui peut atteindre 97% dans certain essais) sur des images aléatoires inconnues par le modèle est relativement satisfaisant si nous considérons les précisions des autres diagnostics et analyses médicaux. Il faut toutefois noter que refaire la classification n'aura aucun impact. Un faux négatif ou positif le restera au deuxième essai. Sauf qu'il reste plus qu'improbable que le modèle ne décèle pas au moins une image lésée parmi les nombreuses images lésées de l'image 3D. En effet, en découpant une image 3D en 190 images 2D, il est clair que la lésion sera présente sur plus d'une coupe et avec des dimensions différentes.

Il est généralement de plus en plus compliqué et difficile de converger vers des taux de précision élevés au fur et à mesure que nous nous rapprochons des 100%, cela est lié au fait que nous aurons besoin d'une augmentation réelle et exponentiel de la base de données pour une augmentation linéaire de la précision. Il est aussi possible que le modèle ait atteint ces limites.

Même si ce le modèle ne peut pas fournir la position de la lésion détectée, il reste toutefois possible de le combiner avec un modèle de localisation probabiliste en fonction des symptômes identifiés par le personnel compétant.

Le bon fonctionnement des modèles dans la dernière expérience est indéniablement lié au non-respect du bon proportionnement équitable des images d'entrainement par classe. Il est donc probable que la réduction du nombre d'images est plus importante que la bonne distribution du nombre d'images par classe dans le cas de ressemblance entre eux.

Il conviendra de donner comme perspectives à ce travail une amélioration du taux de précision de validation et score AUC, car même si ils sont satisfaisant dans un premier temps, il est certain qu'ils peuvent être améliorés d'avantage en jouant sur les hyperparamètres du modèle ainsi que ceux de l'algorithme et en cherchant empiriquement le nombre de couches de convolutions et décisionnels optimaux, nous pourrons difficilement mais certainement dépasser le seuil qu'a atteint cette étude.

Bibliographie

[1] CALVET D, OPPENHEIM C. et MAS J.L., (2009), Infarctus cérébral et AIT: scanner ou IRM en première intention? *Revue Realites cardiologiques*.

[2] Sook-Lei L, Julia M. Anglin et AL, (2018) A large, open source dataset of stroke anatomical brain images and manual lesion segmentations. https://www.nature.com/articles/sdata201811. Consulté le 21 Octobre 2020.

[3] Blanc E, (2014) Les AVC: définition, symptômes, causes et conséquences, https://www.capretraite.fr/prevenir-dependance/sante-grand-age/les-accidents-vasculaires-cerebraux-avc. Consulté le 12 Janvier 2021.

[4] Kidiss B, (2019) Statistiques sur l'AVC en Algérie, https://allodocteurs.africa/maladies/coeur-poumons/avc-60.000-cas-en-algerie. Consulté le 22 Mai 2021.

[5] Kastler B, Vetter D, Patay Z et Germain P, (2006). Comprendre l'IRM Manuel d'auto-apprentissage, ISBN 2-294-05110-6.

[6] Lee SC, Kim K, Kim J, Lee S, Han Yi J, Kim SW, Ha KS et Cheong C, (2001). One micrometer resolution NMR microscopy. *J. Magn. Reson.*, vol. 150, no 2, page 207.

[7] Eiichi F, (1989). NMR in Biomedicine : The Physical Basis, *Springer Science & Business Media*, page 221.

[8] Wikipédia , Rappels de RMN https://fr.wikipedia.org/wiki/Imagerie_par_r%C3%A9sonance_magn%C3%A9tique. Consulté le 1 juillet 2021.

[9] Bernhard B et Winfried K, (1992). Magnetic Resonance Microscopy : Methods and Applications in Materials Science, Agriculture and Biomedicine. *Wiley Ed*, page 604.

[10] Zhi-Pei L et Paul C.L, (1999). Principles of Magnetic Resonance Imaging : A Signal Processing Perspective. *Wiley Ed*, page 416

[11] Chane M, (2017). Classification des images médicales : comprendre le réseau de neurones convolutifs (CNN), https://www.imaios.com/fr/Societe/blog/Classification-des-images-medicales-comprendre-le-reseau-de-neurones-convolutifs-CNN. Consulté le 16 Février 2021.

[12] Hassan U, (2018). VGG16 – Convolutional Network for Classification and Detection, https://neurohive.io/en/popular-networks/vgg16. Consulté le 15 Mars 2021.

[13] Google Cloud, (2021), Advanced Guide to Inception v3 on Cloud TPU https://cloud.google.com/tpu/docs/inception-v3-advanced. Consulté le 25 Avril 2021.

[14] Google Devs, (2017). Developers of google Crash Cours ROC and AUC https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=fr. Consulté le 02 Mai 2021.

[15] Hedieh S, Nastaran P (2019) Age Prediction Based on Brain MRI Image: A Survey, *Springer* article numéro 279

[16] Info-radiologie.ch, (2005). La resonance magnetique, https://www.info-radiologie.ch/resonance_magnetique.php. Consulté le 2 Mai 2021.

[17] Emtiaz H, Mahmudul H, Syed Zafrul H, Tanzina Hassan A, (2020). Deep Learning Based Binary Classification for Alzheimer's Disease Detection using Brain MRI Images. *IEEE* pages 1115-111.

[18] Sinan A, Mohammed A, Abdullah M, Bilal A, (2019). Automatic Brain Tumour Segmentation using fully convolution Network and Transfer Learning (U-net*). IEEE*, pages 188-191.

[19] Castillo, Mary A, (2019). U-ISLES: Ischemic Stroke Lesion Segmentation Using U-Net: Proceedings of the 2018 Intelligent Systems Conference. *IntelliSys* Volume 2, pages 1-5.

[20] Abang M, Kuryati K, Muhammad H, Ahmad T, Siti K, Mohd S, Soon K, Buthainah N, (2020). A Review of MRI Acute Ischemic Stroke Lesion Segmentation. *The International The International Integrated Engineering*, pages 117-127.

[21] Vijayalakshmi M, Sunil Babu V, (2018). A review on acute/sub-acute ischemic stroke lesion segmentation and registration challenges. *Springer*, pages 2484-2503.

[22] Alexand L, Arvid L, (2018). An overview of deep learning in medical imaging. *Science*, pages 102-127.

[23] Yannan Yu, MD et AL, (2020). Use of Deep Learning to Predict Final Ischemic Stroke Lesions from Initial Magnetic Resonance Imaging. *JAMA Network Open*, pages 1-13.