الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث 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 Master

Filière Électronique

Spécialité Électronique des Systèmes Embarqués

présenté par

KHERMAZA ELYES

&

BOUTIARA ABDELOUAHAB

Traitement des images IRM pour la détection des tumeurs cérébrales par les algorithmes de Deep Learning CNN, Faster RCNN, Mask R-CNN et Transfer Learning sous environnement Cloud

Encadré par : Mme. BOUGHERIRA HAMIDA

Année Universitaire 2019-2020

Remerciements

C'est avec grand plaisir qu'on réserve cette page, en signe de gratitude et de reconnaissance à tous ceux qui nous ont aidé à la réalisation de ce travail.

On remercie d'abord notre promotrice Mme. BOUGHERIRA HAMIDA pour sa patience, ses conseils judicieux et pour nous avoir suivi et orienté et toujours poussé à viser plus loin.

On remercie également, notre chef de spécialité Mme. NACEUR DJAMILA pour son encouragement et ses efforts afin de mener à bien notre formation de Master en Electronique des Systèmes Embarqués.

On remercie aussi, notre professeur Mr. NAMANE ABDERRAHMANE qui nous a fait découvrir le monde de la Vision Artificielle et qui nous a mis sur la bonne voie de l'Intelligence Artificielle.

On tient aussi à remercier les membres du jury pour avoir accepté d'examiner et d'évaluer ce travail.

On remercie, enfin, toutes personnes ayant contribué de près ou de loin à la réalisation de ce travail.

Dédicaces

À mes très chers parents

À mon frère le futur ingénieur et ma sœur

À mon cher binôme Elyes

À mes chers amis, Oussama, Saleh Eddine, Zoheir, Manel, Mouhcine et Abderraouf

À tous ceux qui m'ont soutenu moralement

À tous ceux qui ont cru en moi

À nos martyrs

Qu'ils trouvent dans ce modeste travail l'expression de ma reconnaissance et mon estime.

Boutiara Abdelouahab

Dédicaces

À mes très chers parents

À mon frère Yassine et mes sœurs

À mon cher binôme Abdelouahab

À toute la famille Khermaza et Bouzidi

À mes chers amis, Irfane, Ahcene, Walid, Yasmine, Rania, Safwa, Yazid et Rahil.

À tous ceux qui m'ont soutenu moralement

À tous ceux qui ont cru en moi

À nos martyrs

Qu'ils trouvent dans ce modeste travail l'expression de ma reconnaissance et mon estime.

Khermaza Elyes

English Abstract

With the development of computerized medical records and the generalization of imaging techniques, it becomes possible, for a given pathology, to have a large number of heterogeneous, complementary and sometimes ambiguous data. The clinician, analyzing this multiple information, aggregates it, based on subjective and approximate judgments based on his own experience. The purpose of this reasoning is to synthesize a state of the pathology as complete as possible, for example to offer a diagnosis, establish a prognosis or even develop an aid for surgical intervention.

In recent years, formal models of this attitude have been constructed, mostly based on approaches that take into account the redundancies, complementarities and ambiguities inherent in medical data.

Solid brain tumors are pathologies that are characterized by an abnormal proliferation of cells.

Establishing a diagnosis of a brain tumor proved to be a complex process which consists of different stages including the observation of external signs, the performance of clinical examinations and even the removal and analysis of tissues in the case of proven presence of an abnormal mass of cells.

However, observing and analyzing images with tumors has proven to be a delicate exercise, most often requiring the use of several acquisitions under different protocols. Each protocol, defined by specific acquisition parameters, makes it possible to highlight on the images distinct anatomical elements, dependent on their tissue composition. A brain tumor, frequently made up of different types of tissue, therefore cannot always be fully observed with a single acquisition.

To fully locate and visualize the tumor and make a diagnosis, doctors must multiply the number of acquisitions under different protocols and then mentally synthesize the different information provided by each of them.

Medical imagery, which has been constantly evolving in recent years, provides an increasing amount of data. Intelligent / automatic methods of image processing and

analysis have recently increased to assist the expert in the qualitative and quantitative analysis of these images and to facilitate their interpretation.

Automatic segmentation of brain tissue has become a fundamental step in quantitative analyzes of brain MRI images. Many Artificial Intelligence approaches have been proposed to segment these images, classified into supervised and other unsupervised methods.

Artificial intelligence makes it possible to use data more efficiently; its algorithms for medical imaging have developed rapidly around the world.

It has been found that the use of "Deep CNN" neural networks is capable of extracting relevant characteristics from MRI images with the same level of performance as humans.

The objective of this project is to implement a medical diagnostic assistance system, based on different architectures of convolutional neural networks in a Cloud environment, for the detection of brain tumors in magnetic resonance images (MRI) using "Deep Learning" and "Transfer Learning". We used a database containing 253 images augmented to 5313 images for training. We then used Transfer Learning by modifying the structures of the VGG 16, RESNET 50, and Inception V3 networks for classification, and Mask R-CNN for detection.

In this project, we create, on the Colab platform and using the Python programming language and its libraries (notably Keras) and many other tools, a deep learning system based on these neural networks by applying the Transfer Learning method which can be considered as the transfer of knowledge that has already been acquired by pre-trained networks to our new networks in order to process and classify brain MRI images. Thus, we will use the R-CNN Mask for the detection and illustration of exact tumor regions.

The images used (for training and for the test) are images of healthy tissue and others that have abnormalities (tumors). These images once passed through our recognition system are classified either in “normal” class or in “tumor” class. Then the area of the tumor is detected and stained using our R-CNN Mask.

Satisfactory learning, test and validation results have been obtained.

ملخص: الهدف من هذا المشروع هو تنفيذ نظام للمساعدة التشخيصية الطبية ، بناءً على الهياكل مختلفة للشبكات العصبية التلافيفية في بيئة سحابية ، للكشف عن أورام الدماغ في صور الرنين المغناطيسي (MRI) باستخدام "التعلم العميق" و "نقل التعلم" . استخدمنا قاعدة بيانات تحتوي على 253 صورة مُضخمة إلى 5313 صورة للتدريب. ثم استخدمنا نقل التعلم من خلال تعديل هياكل شبكات VGG 16 و RESNET 50 و Inception V3 للتصنيف ، و Mask R-CNN للكشف. تم الحصول على نتائج مرضية في التعلم والاختبار والتحقق.

كلمات المفاتيح: اكتشاف أورام الدماغ ؛ التعلم العميق ؛ نقل التعلم ؛ تصنيف صور الرنين المغناطيسي ؛ زيادة البيانات ؛ الذكاء الإصطناعي ؛ التصوير الطبي

Résumé: L'objectif de ce projet est d'implémenter un système d'aide au diagnostic médical, à base de différentes architectures des réseaux de neurones convolutionnels sous environnement Cloud, pour la détection des tumeurs cérébrales dans les images à résonances magnétique (IRM) en utilisant le «Deep Learning» et le «Transfer Learning». Nous avons utilisé une base de données contenant 253 images augmentées à 5313 images pour l'entrainement. Nous avons ensuite utilisé le Transfer Learning en modifiant les structures des réseaux VGG 16, RESNET 50, et Inception V3 pour la classification, et Mask R-CNN pour la détection. Des résultats d'apprentissage, de test, et de validation satisfaisants ont été obtenus.

Mots clés: Détection des tumeurs cérébrales ; Deep Learning ; Transfer Learning ; Classification d'IRM ; Google Colab ; VIA Annotator ; CNN ; Mask R-CNN ; Faster R-CNN ; ResNet ; Data augmentation ; VGG ; Inception ; Intelligence Artificielle ; Imagerie Médicale

Abstract: The objective of this project is to implement a medical diagnostic assistance system, based on different architectures of convolutional neural networks in a Cloud environment, for the detection of brain tumors in magnetic resonance images (MRI) using "Deep Learning" and "Transfer Learning". We used a database containing 253 images augmented to 5313 images for training. We then used Transfer Learning by modifying the structures of the VGG 16, RESNET 50, and Inception V3 networks for classification, and Mask R-CNN for detection. Satisfactory learning, test and validation results have been obtained.

Keywords: Brain Tumor Detection ; Deep Learning ; Transfer learning ; MRI classification ; Google Colab ; VIA Annotator ; CNN ; Mask R-CNN; Faster R-CNN; ResNet ; Data Augmentation ; VGG; Inception ; Artificial intelligence ; Medical imaging

Listes des acronymes et abréviations

LCS: Liquide Cérébro-Spinal
IRM : Imagerie par Résonance Magnétique
TEP : Tomographie par Émission de Positons
TDM : Tomodensitométrie
CT : Computed Tomography
SRM : Spectroscopie par Résonance Magnétique
RF : Radio Frequency
TR : Temps de Répétition
TE : Temps d'Écho
SEP : Sclérose En Plaque
IA : Intelligence Artificielle
FPGA : Field Programmable Gate Arrays
SRH : Stimulated Raman Histology
CNN : Convolutional Neural Network
R-CNN : Region-based Convolutional Neural Network.
RNA : Réseau de Neurones Artificiels
PMC : Perceptron Multi Couche
ReLU : Rectified Linear Unit
Tanh : Tangente Hyperbolique
MSE : Mean Squared Error
FC: Fully Connected
VGG : Visual Geometry Group
Conv : Convolution
ResNet : Residual Network
SGD: Stochastic Gradient Descent
ILSVRC: ImageNet Large Scale Visual Recognition Competition
SVM: Support Vector Machine (Machine à vecteurs de support)
HOG: Histogram of Oriented Gradients (histogramme de gradient orienté)
ROI: Region Of Interest
FPS: Frame Per Second
GPU : Graphics Processing Unit (unité de traitement graphique).
TPU : Tensor Processing Unit
CPU : Central Processing Unit (unité centrale de traitement).
FPN: Feature Pyramid Network
RPN: Region Proposal Network
SSD: Single Shot MultiBox Detector
VIA: VGG Image Annotator
CSV: Comma-Separated Values
JSON: Java Script Object Notification
LR: Learning Rate
ADAM: ADaptive Moment Estimation
VAL: Validation

Table de matières :

Liste des figures :

Chapitre 3 : Outils et Paramètres d'Implémentation

Chapitre 4 Implémentation et résultats

Liste des Tableaux :

Introduction générale

La segmentation automatique des tissus du cerveau est devenue une étape fondamentale pour les analyses quantitatives des images d'IRM cérébrale. De nombreuses approches d'Intelligence Artificielle ont été proposées pour segmenter ces images, classées en méthodes supervisées et d'autres non supervisées.

L'intelligence artificielle permet d'exploiter les données plus efficacement, ses algorithmes pour l'imagerie médicale se sont développés rapidement dans le monde entier.

Il s'est avéré que l'utilisation des réseaux de neurones profonds « Deep CNN » est capable d'extraire des caractéristiques pertinentes des images IRM avec le même niveau de performance que l'humain.

Dans ce projet, nous créons, sur la plateforme Colab et à l'aide du langage de programmation Python et ses bibliothèques (notamment Keras) et bien beaucoup d'autres outils, un système de Deep Learning basé sur ces réseaux de neurones en appliquant la méthode de Transfer Learning qui peut être considérée comme le transfert des connaissances ayant déjà été acquises par des réseaux pré-entrainés vers nos nouveaux réseaux afin de traiter et classer des images d'IRM cérébrale. Ainsi, Nous utiliserons le Mask R-CNN pour la détection et l'illustration des régions tumorales exactes.

Les images utilisées (pour l'entrainement et pour le test) sont des images des tissus sains (ces normal) et d'autres qui présentent des anomalies (tumeurs). Ces images une fois passées par notre système de reconnaissance sont classées soit en classe « normale » ou en classe « tumeur ». Ensuite, la zone de la tumeur est détectée et colorée a l'intermédiaire de notre Masque R-CNN.

Ce mémoire est composé de 4 chapitres:

Le premier chapitre : Cerveau Biologique et Imagerie Médicale,

Le deuxième chapitre: Architectures des Réseaux de Neurones,

Le troisième chapitre: Outils et Paramètres d'Implémentation,

Le quatrième chapitre: Implémentation et Résultats.

Chapitre 1:
Cerveau Biologique et Imagerie Médicale

Chapitre 1 Cerveau Biologique et Imagerie Médicale

I.1. Introduction

Les tumeurs cérébrales solides sont des pathologies qui se caractérisent par une prolifération anormale de cellules.

L'établissement d'un diagnostic de tumeur cérébrale s'avère être un processus complexe qui se compose de différentes étapes dont l'observation des signes extérieurs, la réalisation d'examens cliniques et même le prélèvement et analyse des tissus en cas de présence avérée d'une masse anormale de cellules.

Cependant, l'observation et l'analyse de clichés présentant des tumeurs s'avèrent être un exercice délicat nécessitant le plus souvent l'utilisation de plusieurs acquisitions sous différents protocoles. Chaque protocole, défini par des paramètres d'acquisition particuliers, permet de mettre en évidence sur les images des éléments anatomiques distincts, dépendants de leur composition tissulaire. Une tumeur cérébrale, composée fréquemment de différents types de tissus, ne pourra donc pas toujours être entièrement observée au moyen d'une unique acquisition.

Pour localiser et visualiser complètement la tumeur et poser leur diagnostic, les médecins doivent ainsi multiplier le nombre d'acquisitions sous différents protocoles puis synthétiser mentalement les différentes informations apportées par chacune d'elles.

Dans ce chapitre nous étudions l'aspect anatomique des tumeurs cérébrales dans une première partie, puis nous présenterons les méthodes de détection des tumeurs cérébrales par imagerie médicale incluant les technologies de reconstruction et de traitement des images médicales.

I.2. Eléments d'Anatomie Cérébrale

Selon une classification ontogénique, le cerveau est divisé en quatre parties hiérarchiquement organisées : le télencéphale, le plus élaboré, puis le diencéphale, le mésencéphale et le myélencéphale, auquel on ajoute le système ventriculo-sous-arachnoïdien et un réseau vasculaire. Le cerveau est en outre composé de trois matières principales, la matière blanche (MB, gaines de myéline regroupées en faisceaux), la matière grise (MG, constituée d'une population de cellules neuronale) et le liquide cérébro-spinal (LCS).

Nous présentons dans cette partie une vue d'ensemble du cerveau, en détaillant les structures qui seront utiles pour la compréhension de la suite.

I.2.1. Le Système Vasculaire

La Figure présente une vue de profil du réseau vasculaire superposé au volume cérébral. Sont représentés sur cette figure le polygone de Willis, nœud du réseau, (violet), et les artères cérébrales antérieure (vert), moyenne (rouge) et postérieure (jaune). Les branches terminales irriguent le cortex, la matière blanche et les noya ux gris centraux. Le polygone de Willis est alimenté par les artères carotides internes et l'artère basilaire, elle-même formée des artères vertébrales. C'est l'artère basilaire et ses branches (ici non représentées) qui irriguent le tronc cérébral et le cervelet [1].

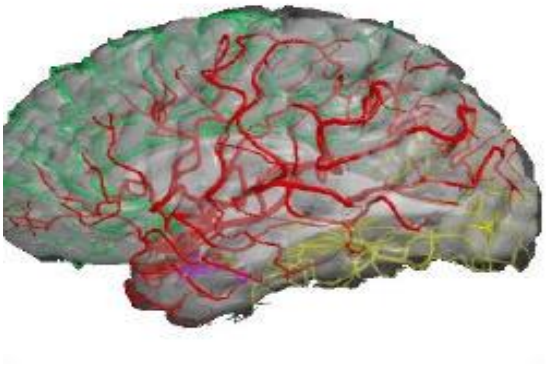


Figure I-1 : Système vasculaire cérébral en violet : polygone de Willis ; en vert : artère cérébrale antérieure ; en rouge : artère cérébrale moyenne ; en jaune : artère cérébrale postérieure

I.2.2. Le Système Ventriculaire

Le système ventriculaire est un ensemble de quatre cavités (deux ventricules latéraux, le troisième (V3) et le quatrième (V4) ventricules) et de communications (l'aqueduc du cerveau reliant le V3 et le V4, les trous de Monro qui font communiquer les ventricules latéraux et le V3), contenant du LCS sécrété essentiellement par les plexus choroïdes ventriculaires.

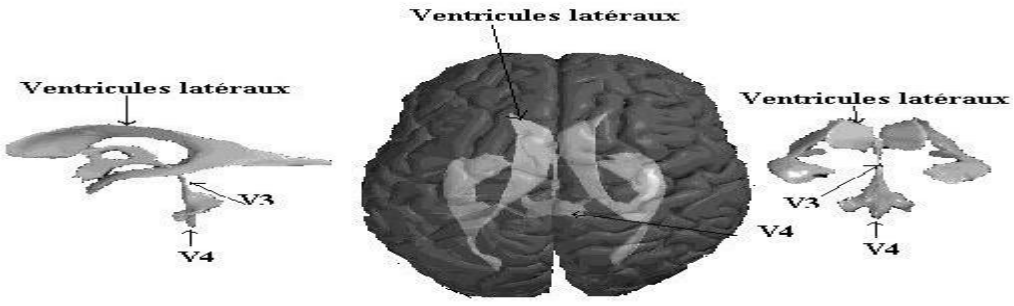


Figure I-2: Représentation du système ventriculaire

I.2.3. Le Télencéphale

Le télencéphale est la région du cerveau la plus développée chez l'homme et est considérée comme le lieu où sont localisées les fonctions supérieures. Il est composé de deux grandes régions, le parenchyme (cortex et substance blanche) et les corps striés.

I.2.4. Le Parenchyme

Le cerveau est divisé en deux hémisphères droit et gauche. Sa surface est parcourue de sillons (sulci), qui délimitent de gros plis de substance grise appelés circonvolutions cérébrales ou gyri. Bien que tous les cerveaux humains aient en commun la présence des sillons et de circonvolutions, ils présentent des variations anatomiques [2]. Deux sillons profonds remarquables sont présents sur chaque hémisphère : le sillon central (ou scissure de Rolando), et le sillon latéral (ou scissure de Sylvius). Chaque hémisphère est divisé en quatre aires principales appelées lobes (frontal, pariétal, temporal et occipital), séparés par des scissures.

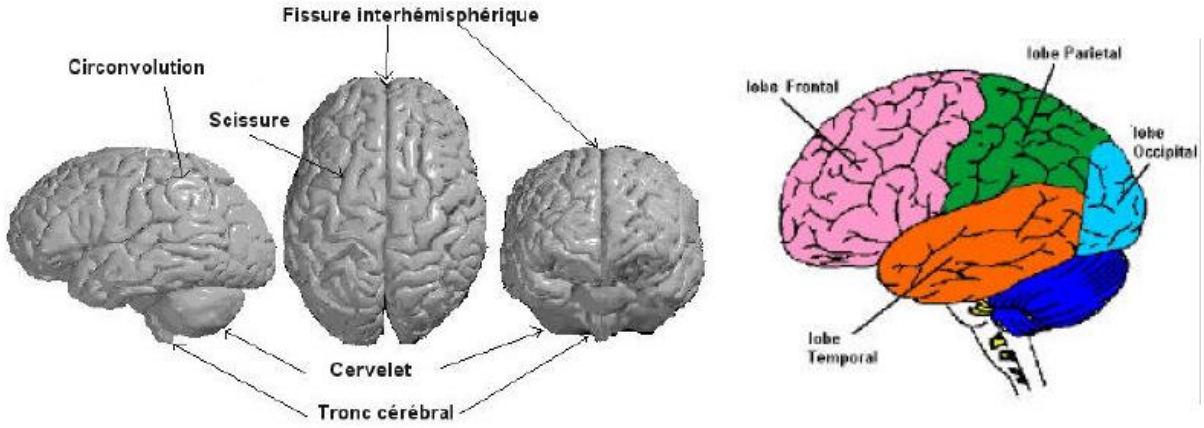


Figure I-3: Vue d'ensemble du cerveau et délimitation des lobes

I.2.5. Corps Striés

Les corps striés sont des regroupements de neurones disposés profondément par rapport à l'écorce corticale. Les corps striés regroupent trois noyaux: le noyau caudé, le noyau lenticulaire et le claustrum. Le noyau lenticulaire est lui-même formé de deux parties : la partie externe s'appelle putamen, qui forme avec le noyau caudé un complexe nommé néostriatum. La partie interne s'appelle pallidum [3].

I.2.6. Le Diencéphale

Le diencéphale est situé au centre du cerveau, entre les deux hémisphères cérébraux. Il est composé du thalamus, de l'hypothalamus et de la région autour du V3. Le premier est une masse de substance grise qui est le grand carrefour auquel aboutissent principalement les sensibilités et les impressions sensorielles [4].

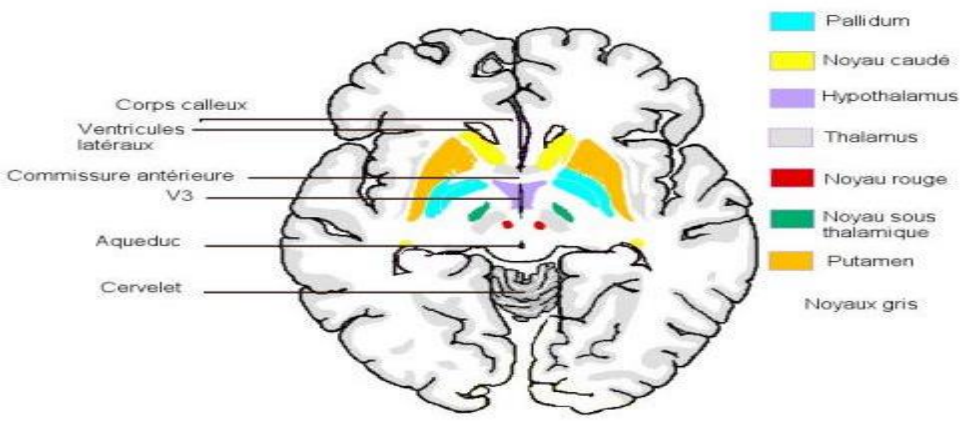


Figure I-4: Coupe axiale de cerveau

I.2.7. Le Mésencéphale

Le mésencéphale est situé entre le diencéphale et le pont. Il est traversé par l'aqueduc du cerveau qui transporte le LCS depuis le V3 vers le V4.

I.2.8. Le Pont et le Cervelet

Le pont et le cervelet constituent la quatrième division du cerveau. Le cervelet est une sorte de petit cerveau, situé à la face postérieure du tronc cérébral
Le pont quant à lui est séparé du cervelet par le V4, et contient de nombreux faisceaux de fibres (dont le faisceau pyramidal) ainsi que les cellules de 3 nerfs crâniens.

I.2.9. Le Bulbe

Le bulbe est la dernière division du cerveau. Il rejoint le canal rachidien par le foramen magnum. Il contient de nombreux faisceaux de fibres, ainsi que les cellules des nerfs cochléovestibulaire (VIIIème), glossopharyngien (IXème), vague (Xème), spinal (XIème) et hypoglosse (XIIème). Les centres des systèmes respiratoire et cardiaque sont également situés dans le bulbe [5].

I.3. La tumeur bénigne et la tumeur maligne

Une tumeur est une masse anormale qui résulte d'une multiplication accrue et non contrôlée de cellules. Une tumeur peut également être appelée lésion, néoplasme ou excroissance.

On distingue en particulier deux types de tumeurs : la tumeur bénigne et la tumeur maligne.

La tumeur bénigne n'est pas un cancer. Elle se caractérise en particulier par son absence de croissance, d'invasion et de métastases. Cependant, elle peut s'aggraver et se transformer en tumeur maligne. Cette dernière se caractérise alors par son caractère envahissant, avec des proliférations de cellules vers les tissus environnants.

I.4. Le diagnostic de la tumeur

Le diagnostic d'une tumeur s'inscrit dans une démarche clinique précise et complexe conduisant, par la suite, à une décision thérapeutique adaptée. Il se découpe en plusieurs étapes : suspicion, détection, observation, détermination de sa nature histologique. . .

I.4.1. Examen clinique

Les signes cliniques accompagnant une tumeur cérébrale sont aussi nombreux que divers. Les symptômes varient considérablement d'un patient à l'autre en fonction du siège de la tumeur. Les plus fréquents sont les maux de tête avec nausées, un ralentissement physique et psychique, une faiblesse musculaire, une difficulté à parler ou une gêne visuelle. Des crises d'épilepsie peuvent être aussi la première manifestation d'une tumeur. L'observation de ces signes conduit le médecin à effectuer un certain nombre d'examens parmi lesquels des examens radiologiques. On recherche alors des masses opaques, claires ou de densité anormale, signes d'un processus tumoral [6]. Ce processus détecté, on détermine le nombre de lésions et leur topographie, puis, pour chacune d'elles, on recherche différentes caractéristiques telles que :

– la taille

– la forme

– le caractère infiltrant ou circonscrit

– la localisation dans le cerveau et par rapport aux structures cérébrales environnantes,

– la composition ; on cherche en particulier si celle-ci est homogène ou au contraire si les lésions présentent des kystes, du liquide ou encore des phénomènes nécrotiques.

Ces éléments ne sont pas les seuls recherchés sur les clichés. Différents signes indirects peuvent en effet être observés : présence d'œdème(s) à proximité de la tumeur, modifications topologiques et volumiques des structures anatomiques adjacentes.

D'autres examens peuvent se révéler nécessaires en fonction de l'état du patient:
- Prélever un peu de liquide céphalo-rachidien (une ponction lombaire).
- Injection du produit iodé dans les artères du cerveau pour mieux préciser leur trajet et effectuer une artériographie.

I.4.2. La gradation des tumeurs

Le tableau I.1 présente la gradation des tumeurs [7]:

grade **I** *Tumeurs bénignes*	Tumeurs à croissance lente et normalement bien circoncites bien qu'elles puissent envahir de grandes régions du cerveau. Selon la localisation, une ablation chirurgicale ou une biopsie peut être recommandée.

grade **II** *Tumeurs bénignes*	Tumeurs à croissance lente, mais, contrairement aux tumeurs de grade I, leurs limites sont imprécises. Les entités tumorales appartenant à ce groupe sont moins nombreuses que celles du grade I.
grade **III** *Tumeurs malignes*	Tumeurs anaplasiques Dans les tumeurs de bas grade (I et II), des foyers de cellules anaplasiques (cellules ayant perdues une partie de leurs caractère propre, donc anormales) se développent activement. Leur évolution est plus rapide que celle des tumeurs de bas grade.
grade **IV** *Tumeurs malignes*	Ces tumeurs peuvent contenir divers types de cellules qui se multiplient rapidement et ayant une forte tendance nécrosante spontanée. Elles ne sont pas bien définies et s'infiltrent dans le cerveau.

Tableau I.1: gradation des tumeurs selon l'OMS (Organisation Mondiale de la Santé)

Pour diagnostiquer ces tumeurs on a d'abord besoin de l'acquisition des images par les méthodes d'imagerie médicale (La tomodensitométrie à rayons X, IRM, TEP…) puis le traitement de ces images par les radiologues, ou par des algorithmes de traitement d'image que nous allons aborder dans la partie I.5 .

I.5. Imagerie Médicale

Dans cette partie, Nous allons présenter de différentes méthodes de reconstruction et de traitement d'images en Imagerie Médicale:

Les examens d'imagerie médicale dépendent essentiellement de la méthode de reconstruction utilisée, On obtient des informations sur l'anatomie des organes (la taille, la forme, le volume et localisation de l'organe), ou sur leur fonctionnement (physiologie, métabolisme).

La figure I-5 illustre les méthodes les plus fréquemment utilisées :

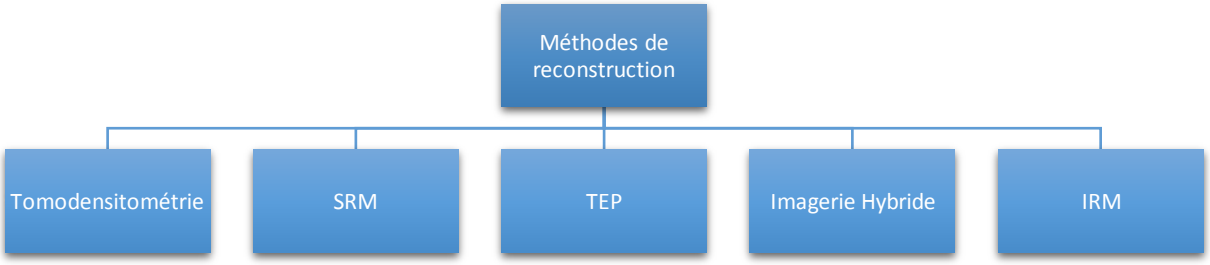


Figure I-5: Imagerie médicale: méthodes de reconstruction

I.5.1. La Tomodensitométrie à rayons X

La tomodensitométrie (TDM) à rayons X (autres noms : scanner X, scanographe, computed tomography (CT)) a constitué l'investigation clinique principale par imagerie jusqu'à l'arrivée de la résonance magnétique nucléaire [8]. Elle permet de visualiser l'objet par tranches successives de quelques millimètres d'épaisseur chacune.

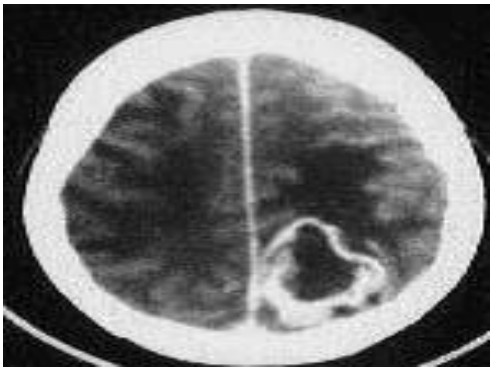


Figure I-6 : Image TDM d'un glioblastome avec injection de produit de contraste

I.5.2. La Spectroscopie par Résonance Magnétique (SRM)

La SRM permet d'accéder à des molécules biologiques autres que celles détectées par les autres méthodes. Le résultat est un spectre composé de plusieurs pics de résonance à des fréquentes différentes. La décomposition du spectre permet de déterminer la fréquence de résonance et de l'aire de chacun des pics correspondant au nombre de noyaux étudiés.
Il existe deux types de SRM : la SRM in vivo qui permet de déterminer la nature histologique d'une tumeur et la SRM in vitro principalement utilisée pour prédire la réponse tumorale à un traitement [9].

I.5.3. La Tomographie par Émission de Positons (TEP)

La TEP est une technique d'imagerie médicale consistant à injecter une substance radioactive, émettrice de positons, à recueillir les rayonnements par un capteur externe et à reconstruire par ordinateur une image en coupe de l'organe [10].
Les deux radio-isotopes les plus utilisés en imagerie TEP sont les isotopes $_{18}F$ et $_{11}C$.

I.5.4. Imagerie Hybride :

Les premières expériences cliniques avec la TEP/IRM suggèrent que cette méthode d'imagerie très sophistiquée possède un grand potentiel pour améliorer le diagnostic non invasif de différentes maladies. Toutefois, de nombreux problèmes méthodologiques restent encore à résoudre avant que la TEP/IRM puisse être utilisée dans la pratique clinique de routine. Les nouvelles connaissances dans la recherche fondamentale moléculaire et dans la radio-pharmacie permettront de déterminer les possibilités d'application futures de cette méthode. L'objectif de la TEP/IRM est clairement de poser des diagnostics de manière non invasive et fiable et d'évaluer précocement la réponse thérapeutique [11].

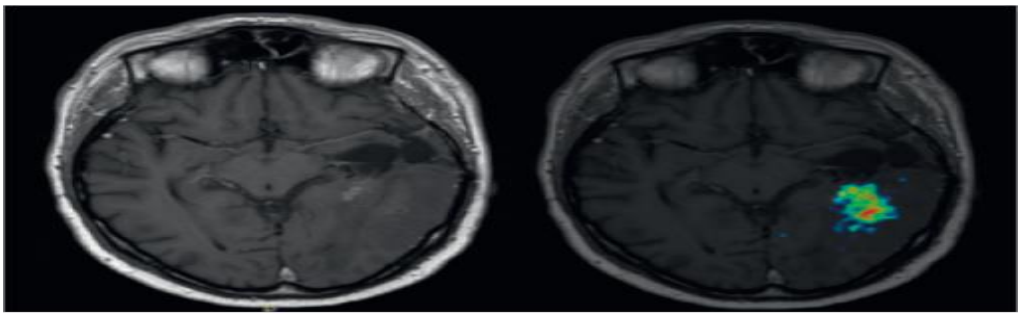


Figure I-7: Gauche: IRM avec produit de contraste (gadolinium). Droite: Fusion avec la TEP à l'éthyle

I.5.5. L'Imagerie par Résonance Magnétique IRM

L'IRM est basée sur le phénomène de résonance magnétique du proton. En imagerie clinique, ce phénomène de résonance s'obtient en plaçant le sujet dans un champ magnétique homogène intense dont l'amplitude varie de 0.2T à 3T, ce qui correspond à des fréquences RF d'excitation de 8 à 120 MHZ. Après excitation, les protons composant les tissus retournent à leur état d'équilibre à des vitesses variables caractéristiques de leur composition physico-chimique en réémettant des ondes RF [12].

Des gradients de champ dispersent ces fréquences de résonance en permettant ainsi le codage spatial. Les vitesses de retour à l'équilibre (les temps de relaxation) constituent la source principale de contraste dans l'image. Comme le scanner ou l'échographie, l'IRM est une modalité d'imagerie médicale tomographique. Par opposition à l'imagerie de projection, représentation 2D d'un objet 3D comme la radiographie, l'imagerie tomographique conserve l'information tridimensionnelle et les distances.

a. Formation de l'IRM

Lors du passage dans l'appareil IRM, le patient est soumis à un champ magnétique initial B_0, puis à une onde électromagnétique d'excitation B_1. Les atomes d'hydrogène entrent alors en résonance avec l'onde de cette dernière. Au terme de la phase d'excitation, les atomes d'hydrogène commencent leur relaxation. Lors de ce retour à l'équilibre ils continuent à émettre un champ appelé « décroissance d'induction libre ».
Ce signal est réceptionné par les capteurs de la machine et transformé en signal électrique qui sera analysé et codé pour constituer une image numérique [13].

La problématique liée au codage de l'image est alors de savoir différencier les signaux issus des différentes zones examinées. Ce codage est réalisé grâce au couplage de trois gradients différents :

– le gradient de coupe sélective sélectionne le plan de coupe, celui-ci pouvant être quelconque dans l'espace,

– le gradient de codage de phase sélectionne les lignes dans le plan de coupe sélectionné,

– le gradient de fréquence sélectionne les colonnes dans le plan de coupe sélectionné.

L'orientation des coupes, qui dépend en particulier du gradient de coupe sélective, varie en fonction de la pathologie recherchée. On retrouve cependant trois orientations principales : axiale, coronale et sagittale. Nous effectuons notre travail sur des images de type axial.

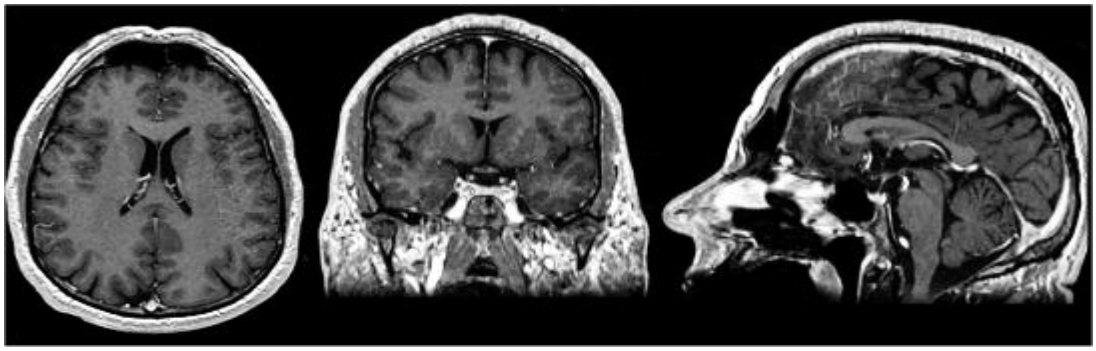


Figure I-8 : plans axial, coronal et sagittal sur une acquisition en T1

b. Caractéristiques des Images IRM

Différents types de contrastes comparés aux autres méthodes d'imagerie médicale, l'IRM est sans doute la plus polyvalente. Elle peut être utilisée pour imager des contrastes induits par différentes caractéristiques des tissus. Ces contrastes peuvent se séparer en deux groupes :

• **Les contrastes statiques**, sensibles au type d'atomes étudié, ainsi qu'à leur nombre et leurs propriétés de relaxation.

• **Les contrastes dynamiques**, sensibles au mouvement des atomes. Typiquement, les contrastes dynamiques peuvent être rendus sensibles au flux sanguin (en angiographie par IRM), à la diffusion de l'eau (en IRM de diffusion) ou à l'irrigation capillaire (en IRM de perfusion).

c. Séquence d'acquisition de l'IRM

Dans l'IRM classique, on peut pondérer l'image en T1, T2 ou en densité de protons, suivant certains paramètres d'acquisition [14]. Trois types de contrastes peuvent donc être obtenus :

– En utilisant un temps de répétition court (TR) et un temps d'écho (TE) court (Neutralise les différences de temps T2), on obtient un contraste d'image pondérée en T1, pondération dite « Anatomique ».

– 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 ».

– Pour les images pondérées en densité de protons, le contraste est obtenu lorsque l'intensité de l'image dépend essentiellement de la densité locale de protons et beaucoup moins des constantes de relaxation T1 et T2. Il est obtenu en utilisant un TR long et un TE court.

Chaque modalité contient des informations spécifiques ne se retrouvant pas dans les autres.

À partir des images pondérées en T1, on peut par exemple distinguer les différents tissus cérébraux tels que la matière blanche, la matière grise et le liquide céphalo-rachidien, tandis que les images pondérées en T2 mettent plus facilement en évidence certaines anomalies comme les lésions dues à la sclérose en plaque (SEP).

d. Avantages de l'IRM

L'étude du cerveau a connu un développement impressionnant ces dernières années grâce à l'IRM. Son développement permettra d'observer le cerveau et ses pathologies avec une précision encore plus fine, à une échelle plus représentative des phénomènes qui l'animent. Elle est l'une des méthodes les plus courantes et les plus importantes pour diagnostiquer et évaluer le cerveau du patient et la croissance de nombre d'appareils IRM reste très élevée. Plusieurs atouts sont à la base de ce succès [15]:

- Un contraste excellent pour les tissus mous grâce aux temps de relaxation T_1 et T_2 et la densité des spins;
- Versatilité dans la modification du signal avec des agents de contraste.
- Bonne résolution spatiale (de l'ordre de 1 mm) et temporelle (de l'ordre de 50 ms pour une seule coupe à quelques minutes pour les images 3D).
- L'examen IRM peut durer jusqu'à une heure, mais La précision est plus élevée que celle du scanner.

I.6. Méthodes de traitement d'images

Pour effectuer des traitements à des images numériques dans le but d'améliorer leur qualité ou d'en extraire de l'information, on peut passer par de différentes méthodes et algorithmes qui se regroupent sous 2 catégories, Les Méthodes Classiques et les Méthodes Intelligentes.

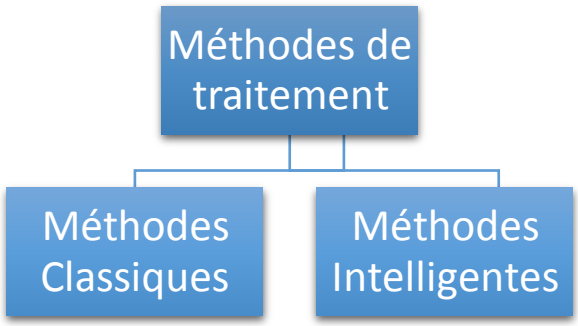


Figure I-9 : Imagerie médicale: méthodes de traitement

I.7. Méthodes classiques

Les méthodes classiques se basent principalement sur la segmentation d'images, ces méthodes sont solides avec leurs bases mathématiques mais sont maintenant dépassées avec l'apparition d'autres méthodes plus sophistiquées et qui donnent de meilleurs résultats, et dans le domaine médical le traitement d'images avec précision joue un très grand rôle à déterminer l'exactitude du diagnostic.

I.7.1. La segmentation de l'image

Cette opération de traitement d'images a pour but de rassembler des pixels entre eux suivant des critères prédéfinis. Plusieurs méthodes sont utilisées et qui appartiennent à une de ces 2 approches :

a. Approches régions

Les approches régions visent à créer une partition de l'image en un ensemble de régions homogènes au sens d'un ou plusieurs critères. On recherche donc à rassembler les pixels par critères de similarité et à leur attribuer une même étiquette. On peut citer les méthodes de seuillages, de croissance de régions et les K-means.

b. Approches contours

Contrairement aux approches régions, qui cherchent à former des zones homogènes, les approches contours correspondent à une discontinuité de l'image afin de déterminer les contours des régions. Les contours obtenus doivent être fermés de façon à obtenir une partition de l'ensemble des pixels de l'image. On peut citer les méthodes dérivatives et markoviennes.

I.8. Méthodes intelligentes

Les nouvelles méthodes ou les méthodes intelligentes sont basées sur la vision par ordinateur et l'Intelligence Artificielle en générale, ces algorithmes sont puissants et varient selon les applications. Dans ce projet nous nous intéressons bien évidement des applications médicales de l'IA et plus précisément sur la relation de l'IA avec l'Imagerie Médicale.

I.8.1. La Médecine et l'Intelligence Artificielle

L'IA augmente la capacité d'analyse et de prise de décision, elle améliore la façon de diagnostiquer, de détecter, de traiter, de pronostiquer une maladie et d'accompagner un patient.

Les systèmes d'IA ont la capacité d'apprendre, conduisant à la découverte de nouveaux phénomènes et à la création de connaissances médicales. Un système informatique doté avec l'intelligence artificielle peut être utilisé pour analyser de grandes quantités de données, Et quand le cas d'un patient est complexe, rare ou que la personne qui pose le diagnostic est

simplement inexpérimentée, un système expert peut aider à établir des diagnostics probables basés sur les données du patient. Maintenant, les systèmes informatiques intelligents peuvent même prédire les étapes chirurgicales, automatiser de différentes taches pendant la chirurgie, identifier les complications et avertir les chirurgiens des défis à venir [16].

L'intelligence artificielle, c'est aussi l'opportunité de rapprocher les territoires. L'IA permet d'avoir l'avis d'un expert éloigné et peut participer à améliorer la prise en charge médicale et non médicale via la télémédecine. De même, Les chirurgiens peuvent envoyer des flux en direct de point de vue du site opératoire à des experts du monde entier pour des conseils en temps réel.

I.8.2. L'Imagerie Médicale et l'Intelligence Artificielle

L'utilisation raisonnée de la technologie et des algorithmes d'intelligence artificielle est le futur de la santé et est déjà en train de la révolutionner. Ainsi, dans le domaine médical, l'IA excelle dans les tâches de perception: elle est en train de changer la radiologie diagnostique, et l'anatomopathologie. Dans ce domaine, ses performances ont été comparées à celles de médecins. Il a ainsi été démontré que dans certains cas l'IA était capable d'être plus rapide, plus précise et plus fiable que l'humain.

Certains voient dans les applications médicales de l'IA la possibilité de remplacer le médecin, Mais c'est sans doute que ses résultats doivent être validés par un humain car les performances des radiologues sont très bonnes et celles des radiologues associés à l'IA sont excellentes, Ce qui renforce l'idée de complémentarité.

Dans la figure I-10, une comparaison du taux d'erreur entre les 3 cas :
le Pathologiste seul, Le modèle d'Intelligence Artificielle seul, et les deux ensembles [17].

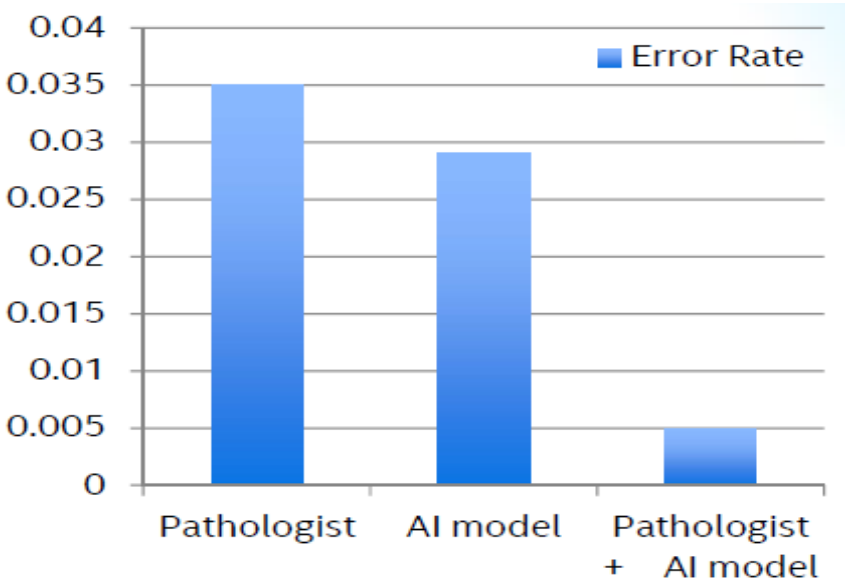


Figure I-10 : statistiques du taux d'erreur du diagnostic des pathologistes et de l'IA et leur association

I.8.3. Solutions d'Intelligence Artificielle pour le diagnostic des tumeurs

Plusieurs études d'imagerie médicale qui utilisent l'intelligence artificielle ont démontré en effet que les techniques d'apprentissage automatique comme les réseaux de neurones à convolution avaient une capacité de décrire de manière non invasive des phénotypes tumoraux avec un pouvoir prédictif supérieur aux mesures cliniques de routine.

Les algorithmes de l'IA sont capables de localiser automatiquement les régions dont les caractéristiques divergent de celles des tissus en bonne santé et d'en extraire les particularités.

I.8.4. L'Électronique Embarquée et L'Imagerie Médicale Intelligente

Le médical en numérique n'est plus en rade, il est en marche et capte l'attention de nombreux secteurs de la recherche scientifiques et de l'industrie.
Dans le but d'obtenir des diagnostiques en temps réel, actuellement, et depuis le début des années, on embarque sur les systèmes de reconstruction d'images médicales, les algorithmes de traitement d'images soit sous forme de programmes exécutés par des processeurs embarqués (software), soit sous forme de circuits (hardware: beaucoup plus puissants) implémentés sur des circuits FPGA (Field Programmable Gate Arrays), ou des circuits intégrés spécifiques.

La nouvelle technique SRH (Stimulated Raman Histology) est parmi les meilleurs exemples de ce domaine, C'est une combinaison d'une technique d'imagerie optique avec un algorithme d'intelligence artificielle [18].

Ce dispositif permet de créer un outil pouvant poser des diagnostics extrêmement précis en temps réel, Les images microscopiques sont traitées et analysées avec les réseaux de neurones CNN implémentés dans le système, et aux environs de 2 minutes et 30 secondes, les chirurgiens peuvent voir un diagnostic prévu de tumeur cérébrale. Utilisant la même technologie, après la résection, ils peuvent exactement trouver et retirer la tumeur autrement indétectable.

D'autres applications de « l'Intelligence Embarquée » en imagerie médicale visent surtout d'assister les médecins même au cours de l'opération pour éviter d'endommager les tissus sains.

Dans ce même contexte, de futures applications de la robotique sont en cours de développement pour fournir des soins de plus en plus élaborés.

I.9. Conclusion

Comme nous avons vu dans les dernières parties de ce chapitre, la médecine ces dernières années se dirige vers l'intelligence artificielle, et l'Imagerie Médicale n'est pas une exception avec tous les algorithmes d'Intelligence Artificielle qui sont en train de la révolutionner comme les réseaux de neurones convolutifs. Dans le chapitre 2, nous allons voir ces derniers et bien d'autres types de réseaux desquels consiste le Deep Learning.

Chapitre 2:
Architectures des Réseaux de Neurones

Chapitre 2 Architectures des Réseaux de Neurones

II.1. Introduction

Afin de détecter les tumeurs cérébrales, Nous allons utiliser l'apprentissage profond (le Deep Learning) qui est une branche du machine Learning, qui lui-même est une branche de l'intelligence artificielle.

Dans ce chapitre nous allons présenter les réseaux neurones et leurs topologies, les CNN et leurs différentes couches et les algorithmes de détection, notamment les R-CNN, Fast RCNN, Faster R-CNN et Mask R-CNN, ainsi quelques bases mathématiques sur lesquelles est fondé le Deep Learning.

II.2. Les neurones artificiels

II.2.1 Le Neurone Formel

Le neurone formel, l'unité élémentaire d'un Réseau de neurones artificiels (RNA), se compose de deux parties [19]:

–évaluation de la stimulation reçue (fonction E)

–évaluation de son activation (fonction f)

Il est caractérisé par :

–son état X (binaire, discret, continu)

–le niveau d'activation reçu en entrée U

–le poids des connections en entrée W

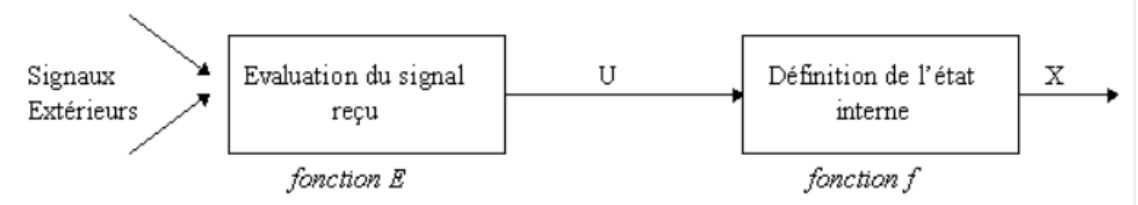


Figure II-1: les éléments d'un neurone formel

II.2.2. La fonction d'entrée

La somme pondérée des signaux d'entrée :

$U_i = E$ (X1 ……X_j .....Xn) = $\sum_{i=1}^{N} Xi \,.\, Wi$

II.2.3. Le biais d'entrée

Unité fictive dont sa valeur permet de régler le seuil de déclenchement du neurone en ajustant les poids avec son initialisation à 1.

II.3. Le perceptron

C'est un modèle utilisé pour la classification binaire avec une simple relation entrée-sortie.

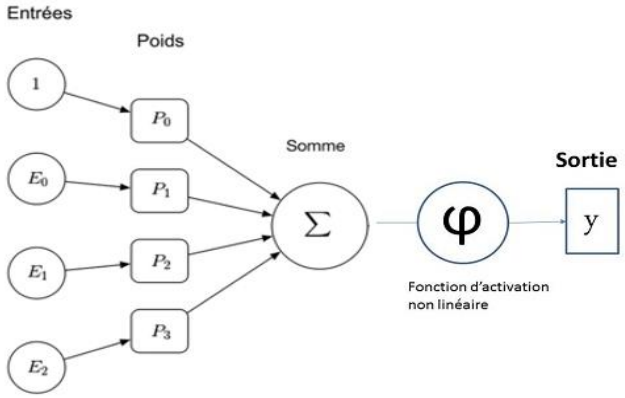


Figure II-2: Représentation du perceptron

• Chaque entrée possède un poids
• La sortie est une fonction du poids et des entrées
• Il est doté d'une couche dédiée à la perception et une couche dédiée à la prise de décision.

II.4. Perceptron multicouche

Le PMC est un réseau de couches successives capables d'approximer toute fonction continue pourvu que l'on fixe convenablement le nombre de neurones dans la couche cachée.

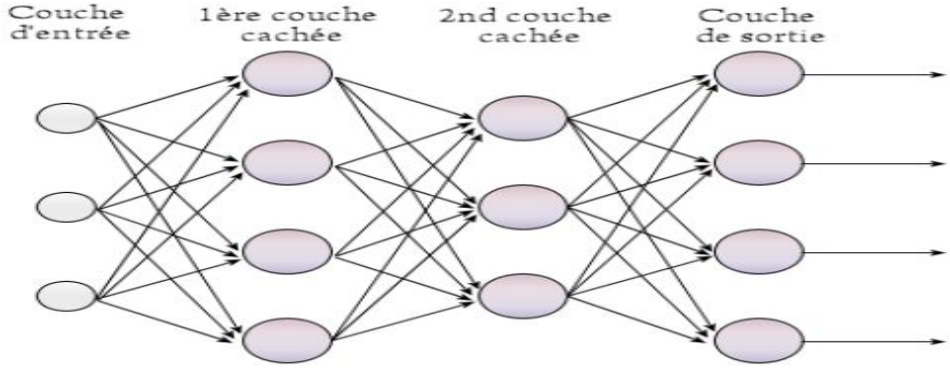


Figure II-3: Perceptron multicouche

II.5. L’apprentissage

L’apprentissage est une phase du développement d’un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu’à l’obtention du comportement désiré. On distingue parmi les types d’apprentissage [20]:

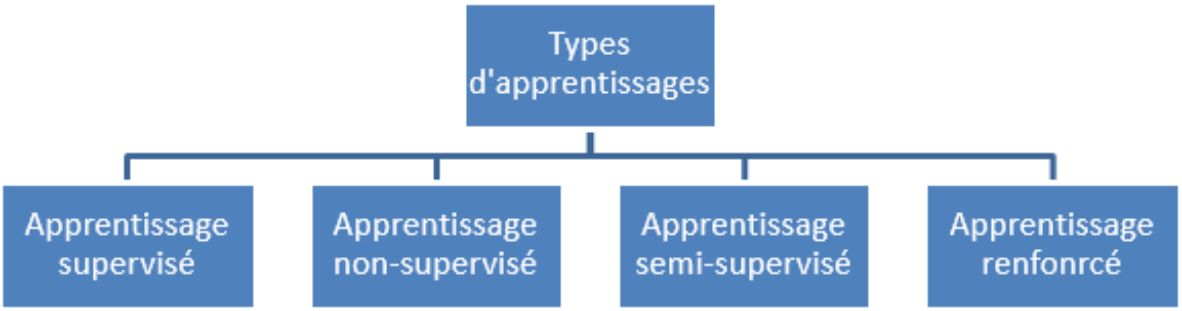


Figure II-4 : Schéma synoptique des types d’apprentissage

II.5.1. L’apprentissage supervisé

On leur fournit des données dites annotées pour entraîner le modèle, c'est-à-dire qu’on leur avait déjà associé un **label** ou une classe **cible** et on veut que l'algorithme devienne capable, une fois entraîné, de prédire cette cible sur de nouvelles données non annotées.

II.5.2. L’apprentissage non supervisé

Les données d'entrées ne sont pas annotées. L'algorithme d'entraînement s'applique dans ce cas à trouver seul les similarités et distinctions au sein de ces données, et à regrouper ensemble celles qui partagent des caractéristiques communes.

II.5.3. L’apprentissage semi supervisé

Il prend en entrée certaines données annotées et d'autres non. Ce sont des méthodes très intéressantes qui tirent parti des deux mondes (supervisé et non-supervisé), mais apportent leur lot de difficultés.

II.5.4. L’apprentissage renforcé

Il se base sur un cycle d'expérience / récompense et améliore les performances à chaque itération. Une analogie souvent citée est celle du cycle de dopamine : une "bonne" expérience augmente la dopamine et donc augmente la probabilité que l'agent répète l'expérience.

II.6. Les fonctions d'activation

II.6.1. La fonction ReLu (Rectified Linear Unit)

Cette fonction, appelée aussi "fonction d'activation non saturante", augmente les propriétés non linéaires de la fonction de décision et de l'ensemble du réseau sans affecter les champs reçus de la couche de convolution [21].

La fonction d'activation est appliquée après la convolution et laisse la taille volume inchangé qui sert comme une couche de correction

La correction ReLU: f (x) = max (0, x)

La ReLU a plusieurs propriétés intéressantes:

- Biologiquement plausible: comparé unilatéral à l'antisymétrique **tanh**
- Activation creuses: 50% des neurones sur une activation nulle après l'initialisation aléatoire des poids
- Pas de vanishing gradient: La dérivée vaut 1 partout dans la portion positive
- Complexité faible: comparaison

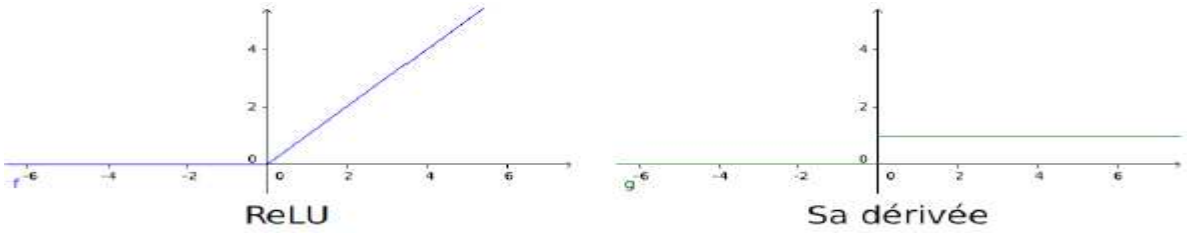


Figure II-5 : la fonction Relu et sa dérivée

II.6.2 La fonction sigmoïde

Elle renvoie une valeur comprise entre 0 et 1. Il s'agit d'une bonne imitation d'un neurone qui s'active ou se désactive car il renvoie le plus souvent des valeurs proches de 0 ou 1. Le problème principal est que le gradient sur les queues est en train de disparaître. Ainsi, l'algorithme de rétro propagation modifie à peine les paramètres et l'apprentissage est ralenti[22].

$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure II-6 : Fonction d'activation Sigmoïde

II.7. Fonctions de perte

Les fonctions de perte quantifient à quel point un réseau neuronal donné est proche de la vérité.

II.7.1. L'erreur quadratique moyenne (MSE)

Tout comme les « moindres carrés ordinaires » dans la régression linéaire, il faut additionner toutes les pertes quadratiques de chaque exemple, puis diviser cette somme par le nombre d'exemples [23].

$$MSE = \frac{1}{N}\sum_{i=0}^{N}|y - x|^2$$

II.7.2. L'algorithme de rétro propagation

C'est une méthode pour calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones, de la dernière couche vers la première [24].

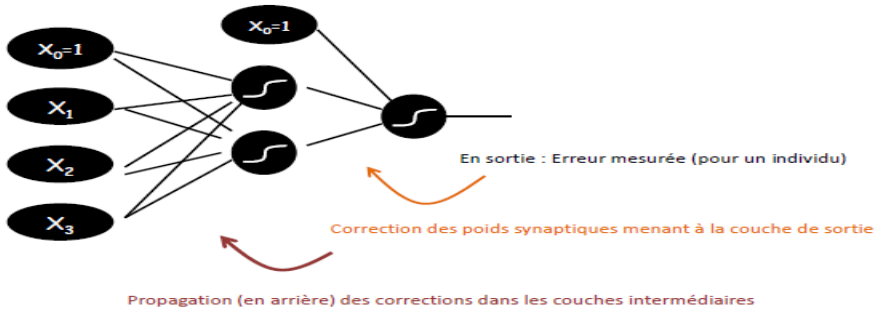


Figure II-7 : L'algorithme de rétro propagation

II.8. Les réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutifs peuvent apprendre des fonctions de cartographie extrêmement complexes lorsqu'ils sont formés sur suffisamment de données. Au niveau de base, les poids d'un CNN sont constitués de filtres. Considérez un filtre comme une matrice (n * n) composée de certains nombres. Maintenant, ce filtre est alambiqué (fait glisser et multipliez) à travers l'image fournie.

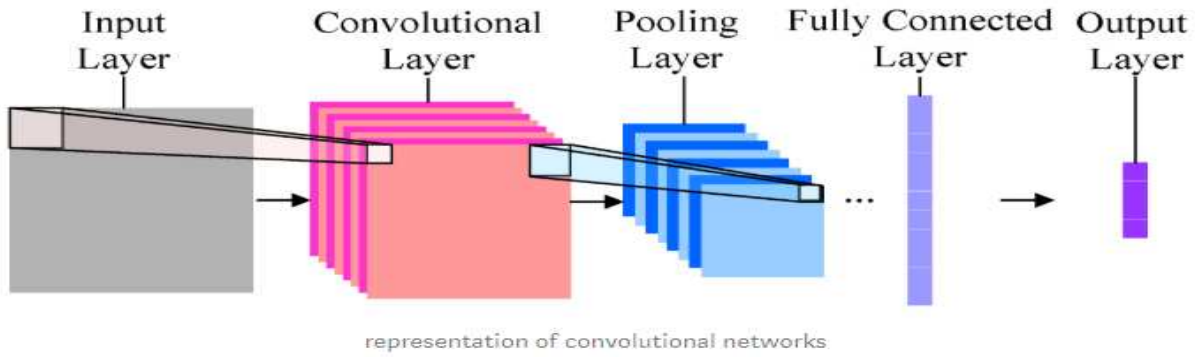


Figure II-8: Représentation d'un réseau de neurones convolutif

Fondamentalement, la formation d'un CNN implique de trouver les bonnes valeurs sur chacun des filtres afin qu'une image d'entrée, lorsqu'elle passe à travers les multiples couches, active certains neurones de la dernière couche afin de prédire la classe correcte.

II.8.1. La convolution

L'objectif de cette couche est de détecter les caractéristiques telles que les bords, les taches de couleur et d'autres éléments visuels, la convolution d'image avec le filtre crée des images appelées cartes de caractéristiques de sortie, plus de filtres dans les couches de convolution plus des caractéristiques ont d´détecté. L'opération de convolution commence dans le coin supérieur gauche de la sous matrice [25].

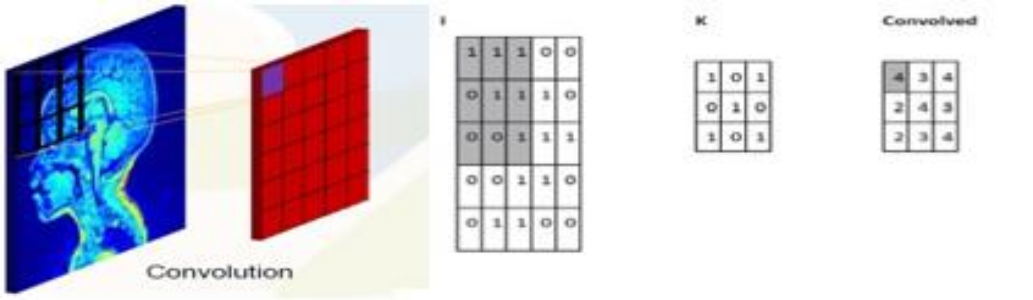


Figure II-9: Exemple de produit de convolution

II.8.2. Le pooling

Le pooling est un sous-échantillon de l'image, permettant à la couche suivante des régions spatiales plus grandes de réduire la dimension (nombre de variables à traiter) mais aussi de rendre le réseau moins sensible aux traductions éventuelles de l'image sans affecter la profondeur [26].

Ainsi, il regarde les réponses des filtres à différents emplacements, permet de pointer les emplacements des filtres qui donnent les mêmes réponses.

a. Max Pooling

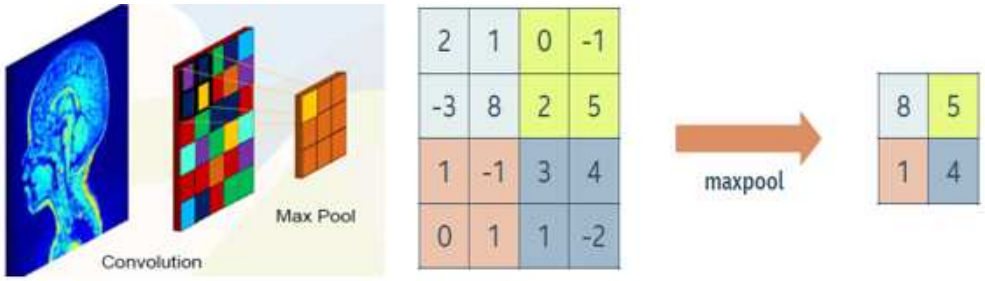


Figure II-10 : Exemple du Max Pooling

b. Moyenne Pooling

Si on avait plutôt choisi l'opération moyenne pooling pour le réseau, la sortie associée à l'opération pooling aurait été donnée par

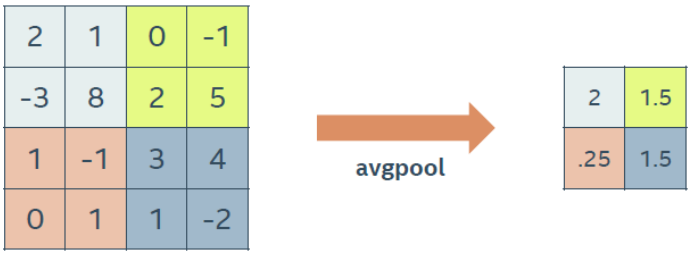


Figure II-11: Exemple du Moyenne pooling

c. Global pooling

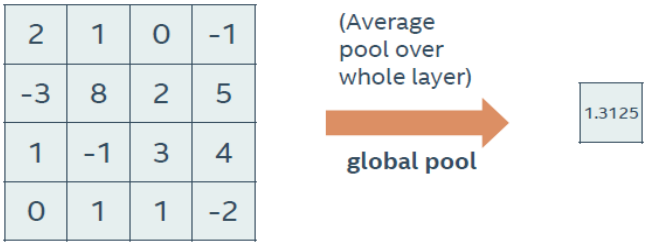


Figure II-12: Exemple du global pooling

II.8.3. Le noyau

Un noyau décrit un filtre qui passe sur une image d'entrée et se déplacer sur toute l'image, de gauche à droite, de haut en bas en appliquant un produit de convolution. La sortie de cette opération est appelée une image filtrée.

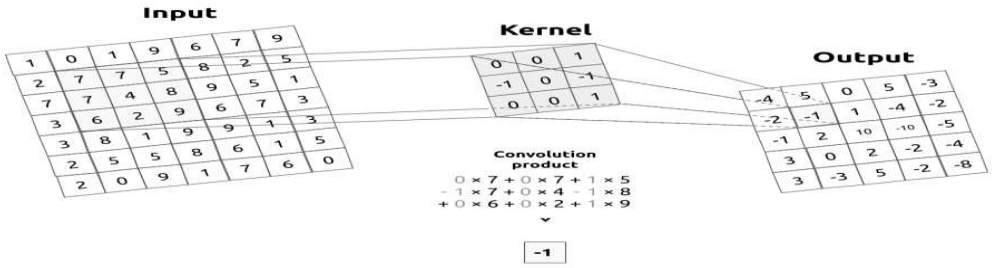


Figure II-13 : Convolution entre une image d'entrée et un noyau

II.8.4. Carte des caractéristiques « Features Map »

Chaque couche de cellules simples est composée d'un certain nombre de filtres de convolution. Pour appliquer le même motif à toute l'entrée, une opération de convolution du filtre fait le travail. On obtient, pour chaque filtre, une carte de caractéristiques « Features Map » de sortie donnant la réponse, à chaque position de l'entrée, de la somme des poids du filtre multipliés un à un aux pixels correspondants dans le voisinage de cette position.

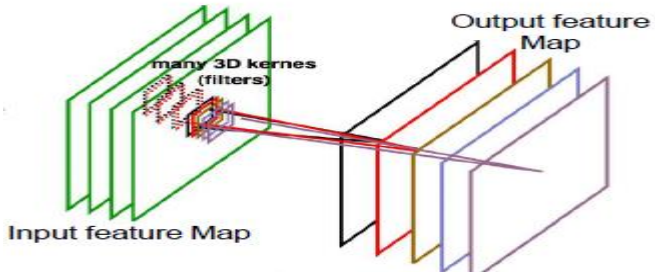


Figure II-14: Exemple d'une carte de caractéristiques « Feature Map »

Le nombre de caractéristiques « Features Map » de sortie est habituellement plus grand que le nombre d'entrée.

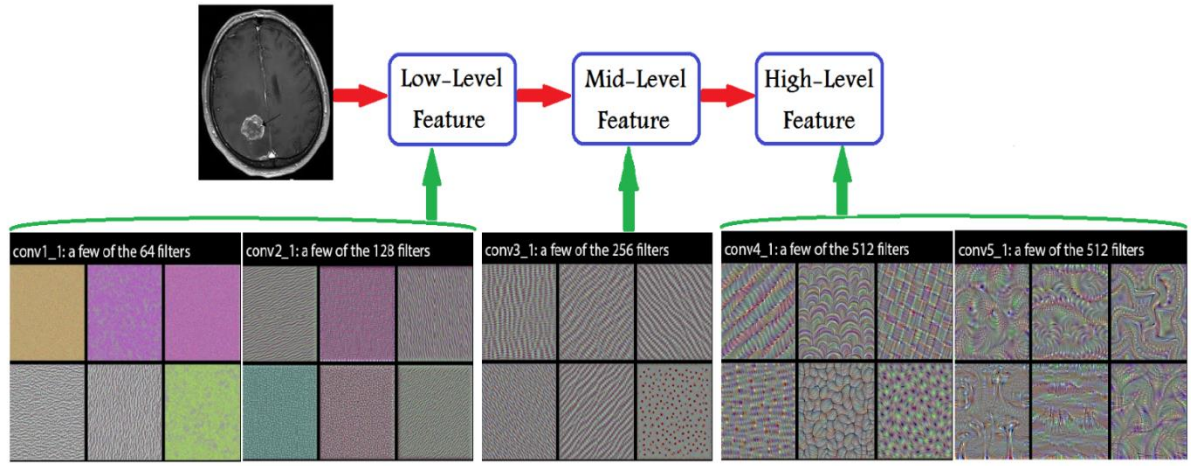


Figure II-15: Visualisation de la carte des caractéristiques dans les réseaux de neurones.

II.8.5. Le pas « Stride »

On choisit le pas auquel on déplace le noyau à travers l'entrant. Le pas horizontal représente le pas auquel on se déplace horizontalement le noyau à travers l'image alors que le pas vertical représente le pas sur les déplacements verticalement le noyau à travers l'image.

Les noyaux, par défaut, se déplacent de gauche à droite, de bas en haut de pixel en pixel. Mais ce mouvement peut aussi être changé [27].

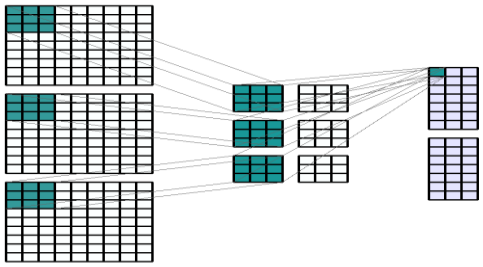


Figure II-16: Exemple du déplacement du noyau

II.8.6. Le remplissage (Padding)

Le remplissage définit le nombre de pixels ajoutés aux côtés des canaux d'entrée avant leur filtrage par convolution. Habituellement, les pixels de remplissage sont définis sur zéro. Le canal d'entrée est étendu.
Ceci est très utile lorsque la taille des canaux de sortie soit égale à la taille des canaux d'entrée. Pour faire simple, le noyau fait 3 * 3, la taille du canal de sortie diminue d'une unité de chaque côté. Pour surmonter ce problème, nous pouvons utiliser un remplissage de 1 [28].

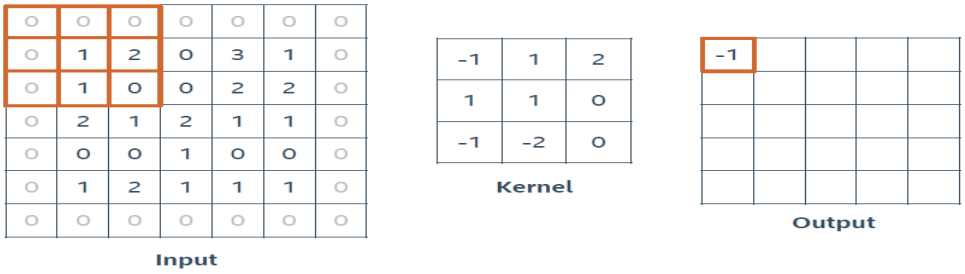


Figure II-17: Exemple de remplissage (Padding)

Le remplissage n'a donc aucun impact sur le nombre de paramètres mais génère un temps de calcul supplémentaire proportionnel à la taille du rembourrage. Mais d'une manière générale, il est souvent suffisamment petit par rapport à la taille du canal d'entrée pour considérer qu'il n'y a pas d'impact sur le temps de calcul.

II.8.7. Dilatation

La dilatation est, en quelque sorte, la largeur du noyau. Par défaut égal à 1, il correspond au décalage entre chaque pixel du noyau sur le canal d'entrée lors de la convolution. Tout comme le Padding, la dilatation n'a aucun impact sur le nombre de paramètres et un impact très limité sur le temps de calcul [29].

Le paramètre de la classe Conv2D est un double de 2 entiers, contrôlant le taux de dilatation pour la convolution dilatée. La convolution dilatée est une convolution de base appliquée uniquement au volume d'entrée avec des espaces définis.

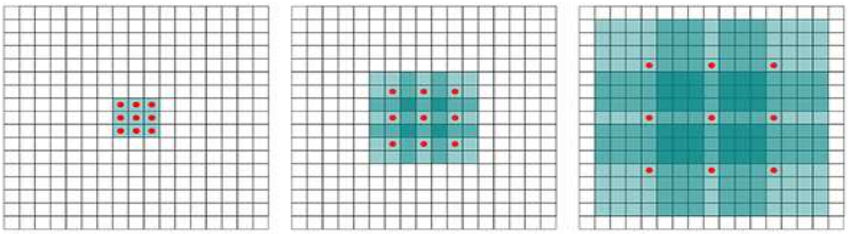


Figure II-18: taux de dilatation, accepte un tuple de 2 entiers pour contrôler la dilatation.

La convolution dilatée est utilisée lorsque: taux de dilatation

1. On travaille avec des images de plus haute résolution mais les détails fins sont toujours importants.
2. On construit un réseau avec moins de paramètres.

II.8.8. La couche entièrement connectée « Fully Connected »

Les neurones d'une couche entièrement connectée sont connectés avec chacun des neurones du volume correspondant à son entrant, c'est-à-dire avec chacun des neurones de la couche précédente. Si on décompose notre volume d'entrants sous forme vectorielle, on a exactement un réseau de neurones PMC [30].

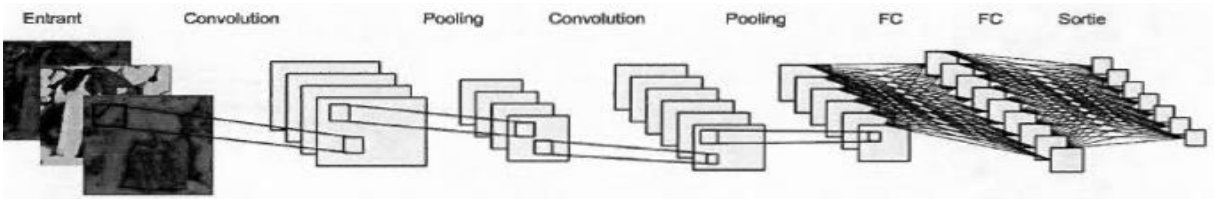


Figure II-19 : Exemple de réseau convolutif deux étapes CONV-POOL suivies par deux couches FC

II.9. Le modèle VGG

Ce réseau se caractérise par sa simplicité, n'utilisant que 3 × 3 couches convolutives empilées les unes sur les autres en profondeur croissante. La réduction de la taille du volume est gérée par la mise en commun maximale. Deux couches entièrement connectées, chacune avec 4096 nœuds, sont ensuite suivies d'un classificateur softmax. Le Dropout fonctionne en déconnectant de manière aléatoire les nœuds de la couche actuelle à la couche suivante. Ce processus de déconnexions aléatoires pendant les lots d'apprentissage aide à introduire naturellement la redondance dans le modèle [31].

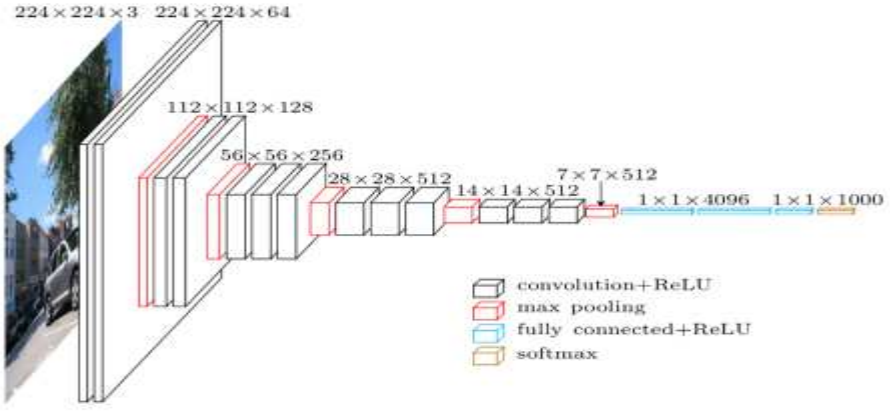


Figure II-20 : Visualisation de l'architecture VGG

II.9.1. VGG16

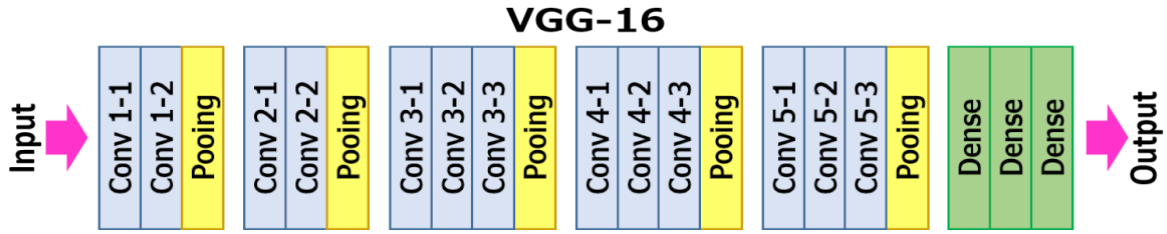


Figure II-21: Le réseau de neurones VGG16

II.9.2. VGG19

- VGG-19 se compose de 16 couches de convolution et de non-linéarité ReLU, séparées par 5 couches de mise en commun et se terminant par 3 couches entièrement connectées.

Le VGG16 et VGG19 représentent le nombre de couches de poids dans le réseau (colonnes D et E dans le tableau II-1):

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Tableau II-1 : Les ***Réseaux convolutionnels très profonds pour la reconnaissance d'images à grande échelle***

II.10. Le Modèle ResNet

Contrairement aux architectures de réseau séquentielles traditionnelles telles que AlexNet, OverFeat et VGG, ResNet est plutôt une forme d '« architecture exotique» qui s'appuie sur des modules de micro-architecture (également appelés «architectures réseau dans réseau»).Une collection de blocs de construction de micro-architecture (avec des couches CONV, POOL, etc.) mène à la macro-architecture (le réseau final lui-même) [32].

L'architecture ResNet est devenue un travail séminal, démontrant que les réseaux extrêmement profonds peuvent être formés à l'aide de SGD standard (et d'une fonction d'initialisation raisonnable) grâce à l'utilisation de modules résiduels:

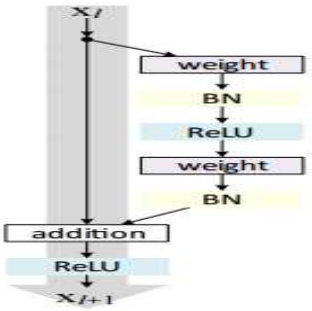


Figure II-22: Le module résiduel dans ResNet tel que proposé initialement

Une plus grande précision peut être obtenue en mettant à jour le module résiduel pour utiliser les mappages d'identité

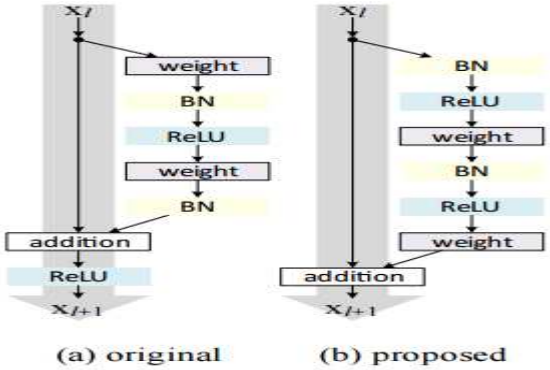


Figure II-23: (Gauche) Le module résiduel d'origine. (Droite) Le module résiduel mis à jour à l'aide de la pré-activation.

Le module résiduel utilise des filtres 1 × 1 et 3 × 3 comme une forme de réduction de dimensionnalité qui aide à maintenir le nombre de paramètres dans le réseau bas. Cela permet de garder le réseau global plus petit avec moins de paramètres [33].

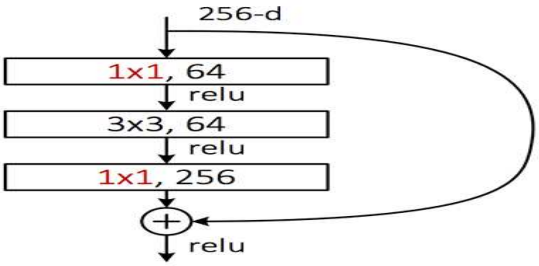


Figure II-24: Réduction de la dimensionnalité avec le « module résiduel »

II.11. Le modèle Inception V3

GoogLeNet, également appelé Inception, a remporté le ILSVRC 2014. Inspiré de VggNet [34], il utilisait des filtres 3 par 3 plus petits dans chaque couche convolutif au lieu des filtres 5 par 5 ou 7 par 7. Cette idée ouvre la possibilité d'augmenter le nombre de couches et de construire des réseaux plus profonds. Inception v3 est utilisé pour aider à l'analyse d'images et à la détection d'objets, et a fait ses débuts en tant que module pour GoogLeNet

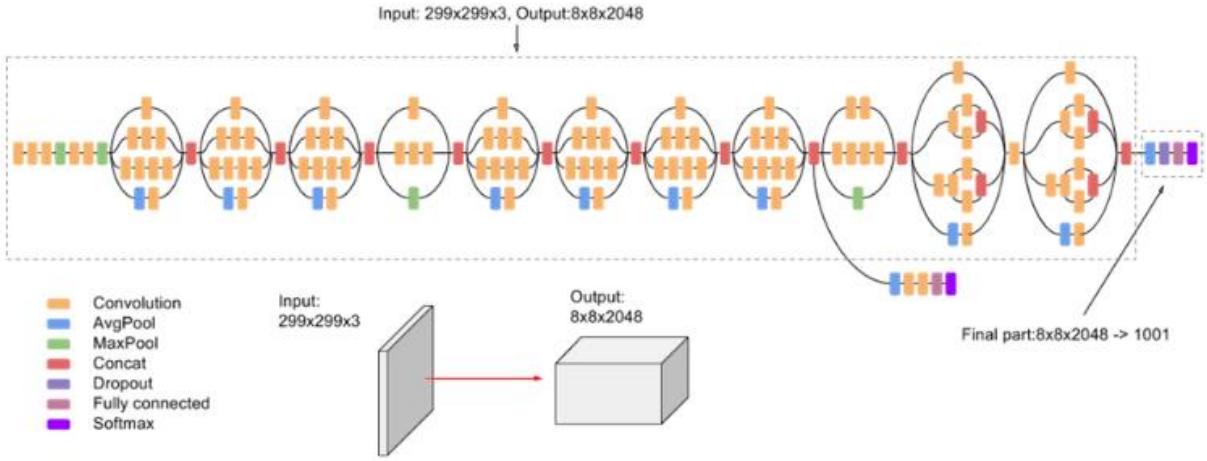


Figure II-25: Architecture du réseau Inception V3

II.12. Réseaux de Neurones Convolutifs Régionaux (R-CNN)

L'algorithme R-CNN d'origine est un processus en quatre étapes [35]:

- **Étape 1:** Saisir une image sur le réseau.
- **Étape 2:**Extraire des propositions de région (Des régions d'une image pouvant contenir des objets) à l'aide d'un algorithme tel que Recherche sélective.
- **Étape 3:**Utilisez l'apprentissage par transfert, en particulier l'extraction de fonctionnalités, pour calculer les fonctionnalités de chaque proposition à l'aide du CNN.
- **Étape 4:**Classifiez chaque proposition à l'aide des fonctionnalités extraites avec une machine à vecteur de support (SVM).

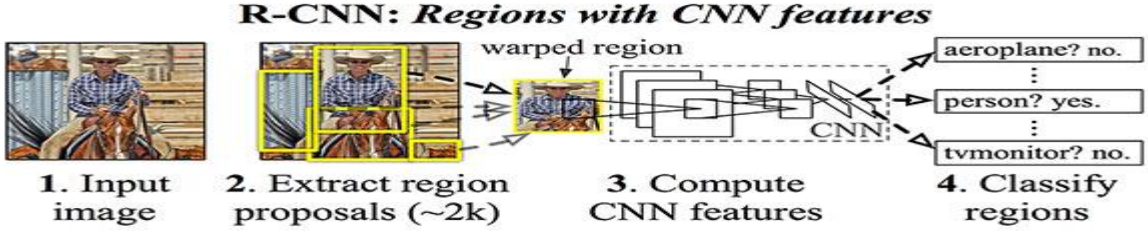


Figure II-26: L'architecture R-CNN d'origine

Cependant, le problème avec la méthode R-CNN est qu'elle est incroyablement lente. Et de plus, nous n'apprenons pas réellement à localiser via un réseau de neurones profond, on construit simplement un système plus avancé Détecteur linéaire HOG + SVM [36].

II.13. Fast R-CNN

Pour améliorer le R-CNN d'origine, l'algorithme Fast R-CNN a été publié [37]:

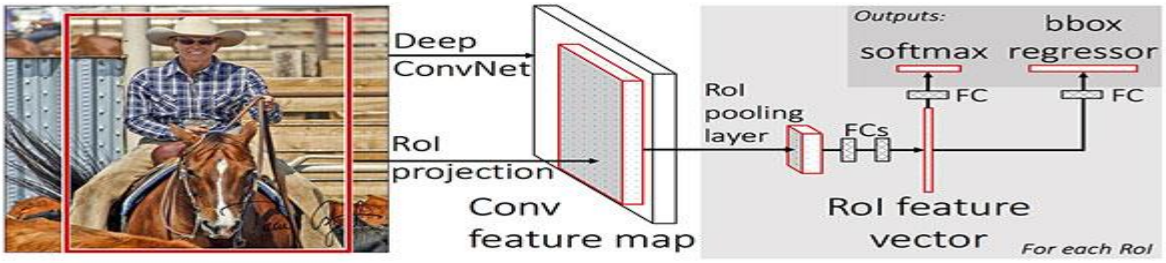


Figure II-27: L'architecture Fast R-CNN

Le ROI Pooling fonctionne en extrayant une fenêtre de taille fixe de « Features Map » et en utilisant ces entités pour obtenir l'étiquette de classe finale et le cadre de délimitation. Le principal avantage ici est que le réseau peut désormais être formé de bout en bout:

1. Entrer une image et les boîtes de délimitation de la vérité au sol associées
2. Extraire la carte des caractéristiques
3. Appliquer le pool de ROI et obtenir le vecteur de fonctionnalité ROI
4. Et enfin, utiliser les deux ensembles de couches entièrement connectées pour obtenir les prédictions d'étiquettes de classe et les emplacements du cadre de délimitation pour chaque proposition.

II.14. Faster R-CNN

Dans l'ensemble, l'architecture Faster R-CNN est capable de fonctionner à environ 7-10 FPS, une étape énorme vers la réalisation de la détection d'objets en temps réel avec un apprentissage en profondeur.

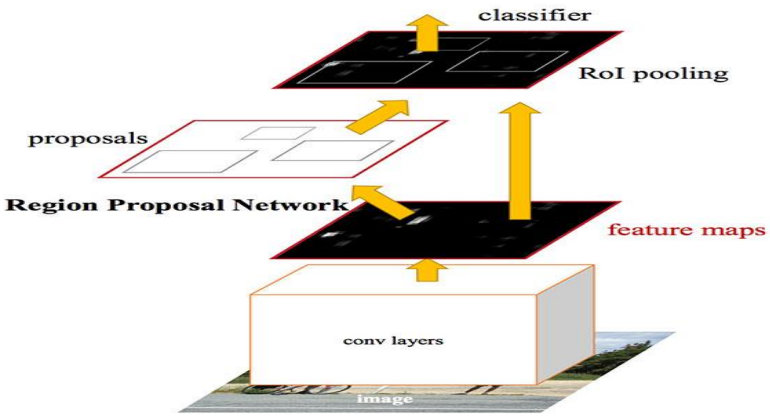


Figure II-28: L'architecture plus rapide R-CNN « Faster R-CNN »

II.14.1. Problèmes avec Faster RCNN

Tous les algorithmes de détection d'objets utilisent des régions pour identifier les objets. Le réseau ne regarde pas l'image complète d'un seul coup, mais se concentre sur des parties de l'image de manière séquentielle. Cela crée deux complications:

- L'algorithme nécessite de nombreux passages à travers une seule image pour extraire tous les objets
- Comme il existe différents systèmes fonctionnant les uns après les autres, les performances des systèmes plus avancés dépendent de la performance des systèmes précédents.

II.15. Mask R-CNN

Le masque R-CNN est une extension du populaire Faster R-CNN introduit deux contributions majeures [38]:

1. Remplacement du module ROI Pooling par un module ROI Align plus précis
2. Insertion d'une branche supplémentaire hors du module ROI Align

Les chercheurs ont lancé une architecture d'apprentissage en profondeur, appelée Mask R-CNN, qui peut créer un masque pixel par pixel pour chaque objet d'une image. La branche supplémentaire accepte la sortie du ROI Align, puis l'introduit dans deux couches CONV.La sortie des couches CONV est le masque lui-même.

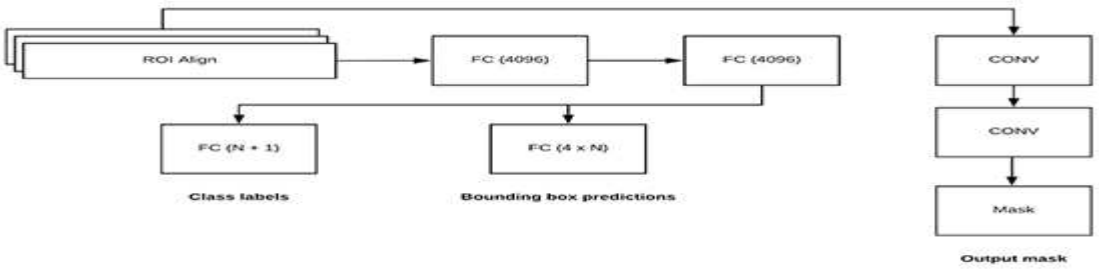


Figure II-29: Mask R-CNN. La sortie des couches CONV est le masque lui-même.

II.15.1. Mask R-CNN en temps réel

Les R-CNN plus rapides sont incroyablement coûteux en calcul, et lorsqu'on ajoute une segmentation d'instance au-dessus de la détection d'objets, le modèle ne devient que plus coûteux en termes de calcul, par conséquent:

- Sur un CPU, un masque R-CNN ne peut pas fonctionner en temps réel.
- Mais sur un GPU, Le masque R-CNN peut obtenir jusqu'à 5-8 FPS.

Pour exécuter le Mask R-CNN en semi-temps réel, on aura besoin d'un GPU.

II.15.2. L'évolution du Mask R-CNN

Le modèle Mask R-CNN pour la segmentation d'instance a évolué à partir de trois architectures précédentes pour la détection d'objets:

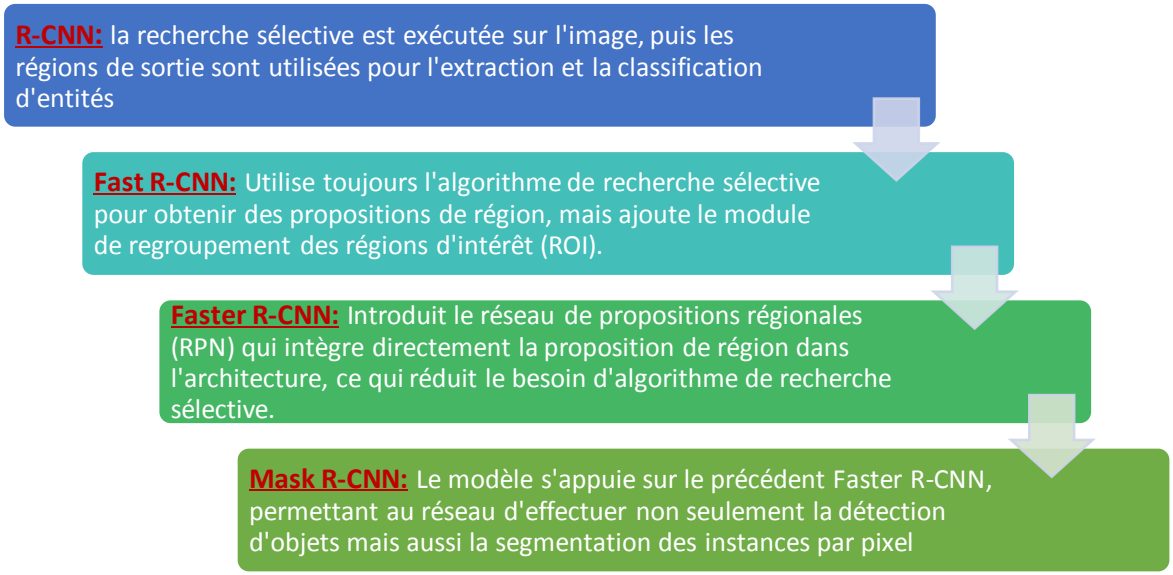


Figure II-30 : Schéma synoptique du Mask R-CNN

Le masque R-CNN y ajoute une troisième branche qui génère également le masque d'objet :

1. En transmettant l'image en entrée au ConvNet, qui renvoie « features Map » de cette image
2. Le réseau de proposition de région (RPN) est appliqué sur ces cartes d'entités « Features Map ». Cela renvoie les propositions d'objets avec leur score d'objectivité
3. Une couche de regroupement RoI est appliquée sur ces propositions pour ramener toutes les propositions à la même taille

4. Enfin, les propositions sont transmises à une couche entièrement connectée pour classer et sortir les cadres de délimitation des objets. Il renvoie également le masque pour chaque proposition.

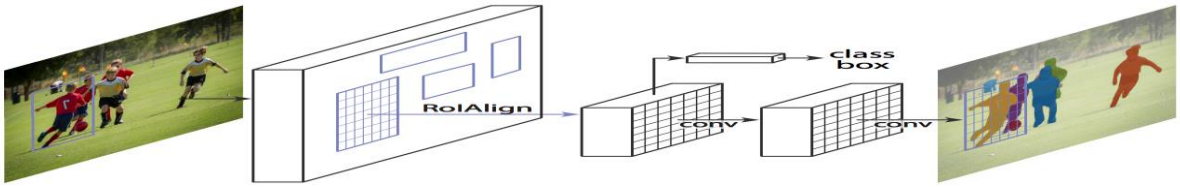


Figure II-31: Exemple de segmentation avec le Mask R-CNN

II.15.3. Région d'intérêt (ROI)

Les régions obtenues du RPN peuvent avoir des formes différentes, nous appliquons une couche de regroupement et convertissons toutes les régions à la même forme. Ensuite, ces régions sont passées à travers un réseau entièrement connecté « fully connected » afin que l'étiquette de classe et les boîtes englobantes soient prédites.

Pour cela, il faut calculer d'abord la région d'intérêt afin que le temps de calcul puisse être réduit. La relation est comme ceci:

ROI = Aire de l'intersection / Aire de l'union

II.15.4. Segmentation d'instance vs segmentation sémantique

On distingue 2 types de segmentation [39]:

a. Segmentation sémantique

Les algorithmes de segmentation sémantique nous obligent à associer chaque pixel d'une image d'entrée à une étiquette de classe (y compris une étiquette de classe pour l'arrière-plan).

Chaque objet est en effet segmenté mais chaque objet a la même couleur. Bien que les algorithmes de segmentation sémantique soient capables d'étiqueter chaque objet d'une image, ils ne peuvent pas différencier deux objets de la même classe.

Ce comportement est particulièrement problématique si deux objets de la même classe s'occluent partiellement - nous n'avons aucune idée où les limites d'un objet se terminent et le suivant commence.

b. Segmentation d'instance

Ces algorithmes, en revanche, calculent un masque au niveau des pixels pour chaque objet dans l'image, même si les objets sont de la même étiquette de classe.

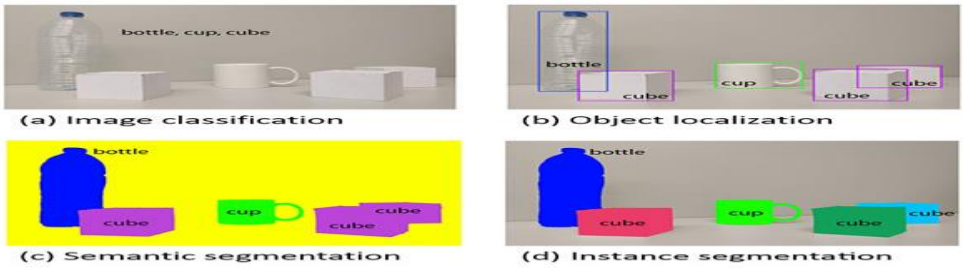


Figure II-32: Classification d'image (en haut à gauche), détection d'objets (en haut à droite), segmentation sémantique (en bas à gauche) et segmentation d'instance (en bas à droite)

II.15.5. Fonctionnement du Mask R-CNN

Le réseau se divise en deux étages : le premier étage est constitué d'un réseau FPN « Feature Pyramid Network » et un autre réseau plus léger RPN « Region Proposal Network ». Ensuite un autre réseau va envoyer les caractéristiques « features » vers le ROI Align qui va les donner aux couches du deuxième étage qui va générer la prédiction et l'instance du masque.

a. Réseau pyramidal de caractéristiques « FPN »

Le FPN est un réseau de neurones profonds de style[40]. Il se compose d'un chemin ascendant, d'un chemin haut en bas et de connexions latérales. La voie ascendante peut être n'importe quel ConvNet, généralement ResNet ou VGG, qui extrait des fonctionnalités d'images brutes. La voie de haut en bas génère une carte pyramidale d'entités de taille similaire à la voie de bas en haut. Les connexions latérales sont la convolution et l'ajout d'opérations entre deux niveaux correspondants des deux voies. Le FPN surpasse les autres ConvNets simples principalement pour la raison qu'il maintient de fortes fonctionnalités sémantiques à différentes échelles de résolution.

b. Réseau de propositions de région « RPN »

Un réseau neuronal léger appelé RPN analyse toutes les voies FPN de haut en bas et propose des régions pouvant contenir des objets. Les classes de vérité au sol (seuls les objets ou les binaires d'arrière-plan sont classés à ce stade) et les boîtes englobantes sont attribuées à des ancres individuelles IoU valeur. RPN utilise ces ancres pour déterminer où la carte d'entités devrait obtenir un objet et la taille de son cadre de sélection [41].

À la deuxième étape, un autre réseau de neurones prend les régions proposées par la première étape et les affecte à plusieurs zones spécifiques d'un niveau de carte d'entités, analyse ces zones et génère des classes d'objets (classées multi-catégorielles), des boîtes englobantes et des masques. La procédure ressemble à RPN. Les différences sont que sans l'aide d'ancres, la deuxième étape a utilisé une astuce appelée ROIAlign pour localiser les zones pertinentes de la carte des caractéristiques, et il existe une branche générant des masques pour chaque objet au niveau des pixels.

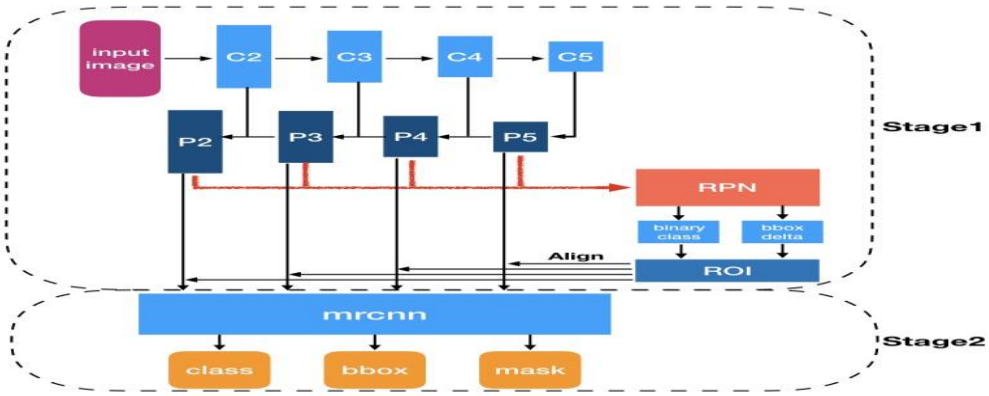


Figure II-33 : Illustration de la structure du masque RCNN

Lors de la prédiction, les ROI passent par suppression non maximale et les 100 premières boîtes de détection sont conservées. Enfin, le masque redimensionné peut être superposé sur l'image d'entrée d'origine.

Les détecteurs d'objets tels que les YOLO, les SSD et les R-CNN ne sont capables de produire que les coordonnées du cadre de délimitation d'un objet dans une image - ils ne nous disent rien sur la forme réelle de l'objet lui-même.

En utilisant le masque R-CNN, nous pouvons générer des masques au niveau des pixels pour chaque objet d'une image, ce qui nous permet de segmenter l'objet au premier plan de l'arrière-plan.

II.16. Comparaison entres les réseaux de neurones

Algorithmes	**Caractéristiques**	**Temps / image de prédiction**	**Limites**
CNN	Divise l'image en plusieurs régions, puis classe chaque région en différentes classes.	-	Nécessite beaucoup de régions pour prédire avec précision et donc un temps de calcul élevé.
R-CNN	Utilise la recherche sélective pour générer des régions. Extrait environ 2000 régions de chaque image.	40-50 secondes	Le temps de calcul élevé étant donné que chaque région est transmise au CNN séparément, il utilise également trois modèles différents pour faire des prédictions.

Fast R-CNN	Chaque image n'est transmise qu'une seule fois au CNN et les cartes d'entités sont extraites. La recherche sélective est utilisée sur ces cartes pour générer des prédictions. Combine les trois modèles utilisés dans RCNN ensemble.	2 secondes	Le temps de calcul élevé étant donné que chaque région est transmise au CNN séparément, il utilise également trois modèles différents pour faire des prédictions.
Faster R-CNN	Remplace la méthode de recherche sélective par un réseau de propositions de régions, ce qui a rendu l'algorithme beaucoup plus rapide.	0.2 secondes	La proposition d'objet prend du temps et comme différents systèmes fonctionnent l'un après l'autre, les performances des systèmes dépendent de la performance du système précédent.
Mask R-CNN	Le modèle s'appuie sur le précédent Faster R-CNN, permettant au réseau d'effectuer non seulement la détection d'objets mais aussi la segmentation des instances par pixel	T < 0.2 secondes	Les régions de la carte d'entités sélectionnées par RoIPool étaient légèrement décalées par rapport aux régions de l'image d'origine. Pour régler ce problème on doit utiliser RoIAlign au lieu du RoIPool.

Tableau II-2 : Comparaison entre les réseaux de neurones

II.17. Conclusion

Dans ce second chapitre, nous avons présenté les différents types de réseaux de neurones et avons décrit leur fonctionnement théorique et leurs caractéristiques.

Dans le prochain chapitre, Nous allons présenter les outils utilisés pour l'implémentation de nos réseaux ainsi les étapes d'installations et de manipulation, tout en introduisant la méthode de Transfer Learning.

Chapitre 3:
Outils et Paramètres d'Implémentation

Chapitre 3 Outils et Paramètres d'Implémentation

III.1. Introduction

Ce chapitre présente les différentes plateformes et outils utilisés pour l'implémentation de nos réseaux (Google Colab, VIA Annotator, le language de programmation Python et ces Bibliothèques), les méthodes et les techniques appliquées (Transfer Learning, Data Augmentation), l'architecture modifiée de chacun de nos réseaux (VGG, ResNet, Inception et Mask R-CNN) et les paramètres nécessaires pour la configuration de leur entrainement et apprentissage.

III.2. Base de données

Afin de réaliser ce projet de détection des tumeurs cérébrales il fallait construire une base de données contenant des images IRM. Les hôpitaux et les centres d'imagerie médicale n'étaient pas utiles car ils classent la nature de notre base de données en tant que « sensible » et parmi « les informations confidentielles des patients » et il fallait passer par une infinité de procédures pour pouvoir - peut-être - avoir une autorisation spéciale.

Il fallait donc trouver l'alternatif en parcourant des sites et plateformes de recherche médicale spécialisée dont on en mentionne: La plateforme « OASIS-Brains ». Malheureusement, ce genre de plateformes scientifiques impose autant des formalités que les hôpitaux comme devoir remplir quelques conditions et signer un contrat.

La dernière solution était les plateformes « Open source », qui avait peu de données mais qui n'imposaient aucune condition ou obligation de la façon d'utiliser les données sachant que ces données étaient complètement anonymes et ne contenait aucune information autre que les images elles-mêmes.

Notre base de données a été obtenue à partir de la plateforme Open Source « Github ». Elle est composée d'environ 250 images IRM au format JPEG présentant les 2 classes en 2 dossiers:

- Classe 1 : présence de la tumeur nommée « Yes » avec 155 images
- Classe 2 : absence de la tumeur nommée « No » avec 98 images

Il est évident que Ce nombre d'images est très peu pour entrainer nos réseaux de neurones, donc nous avons recouru à une solution très pratique qui a résolu ce problème de manque des images IRM. On a utilisé une technique de traitement d'image permettant de générer plus de données/d'images à partir des données originales. Cette technique est appelée « Data augmentation », elle va être présentée et son fonctionnement sera expliqué dans ce qui suit.

III.3. Augmentation d'images (Data Augmentation)

L'augmentation des données est une technique pour créer artificiellement de nouvelles données d'entraînement à partir des données d'entraînement existantes. Cela se fait en créant des versions transformées d'images dans l'ensemble de données d'apprentissage qui appartiennent à la même classe que l'image d'origine. Les techniques d'augmentation peuvent créer des variations des images qui peuvent améliorer la capacité des modèles à généraliser ce qu'ils ont appris à de nouvelles images [42].

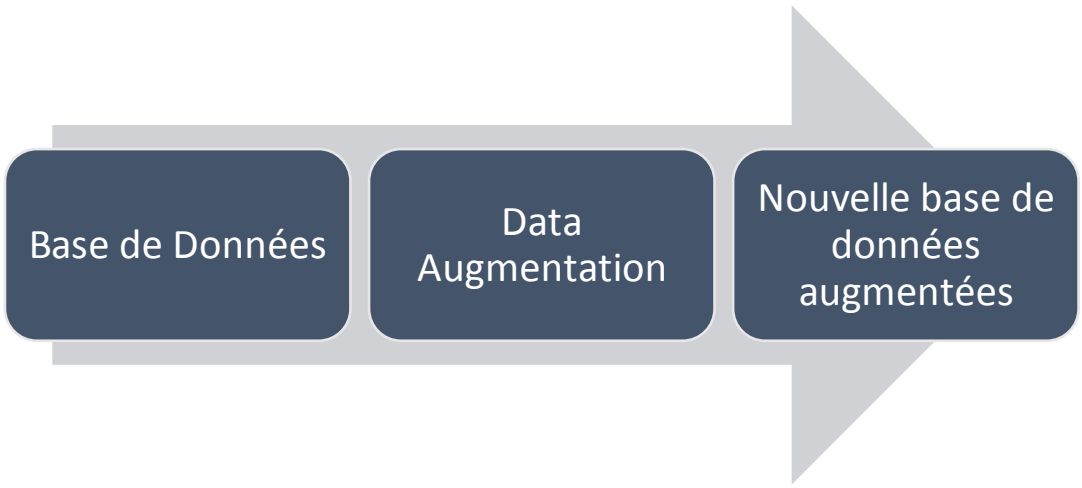


Figure III-1:étapes d'obtention d'une nouvelle base de données à partir de la base de données originale

La bibliothèque de Deep Learning Keras offre la possibilité d'utiliser l'augmentation des données automatiquement lors de l'entrainement d'un modèle, ce qui est possible en utilisant la classe ImageDataGenerator. Tout d'abord, la classe peut être instanciée et la configuration des types d'augmentation de données est spécifiée par des arguments au constructeur de classe.

III.3.1 Types de Data Augmentation

Il existe de nombreuses techniques d'augmentation des données, on a choisi quelques-unes, elles sont reprises dans la figure III-10:

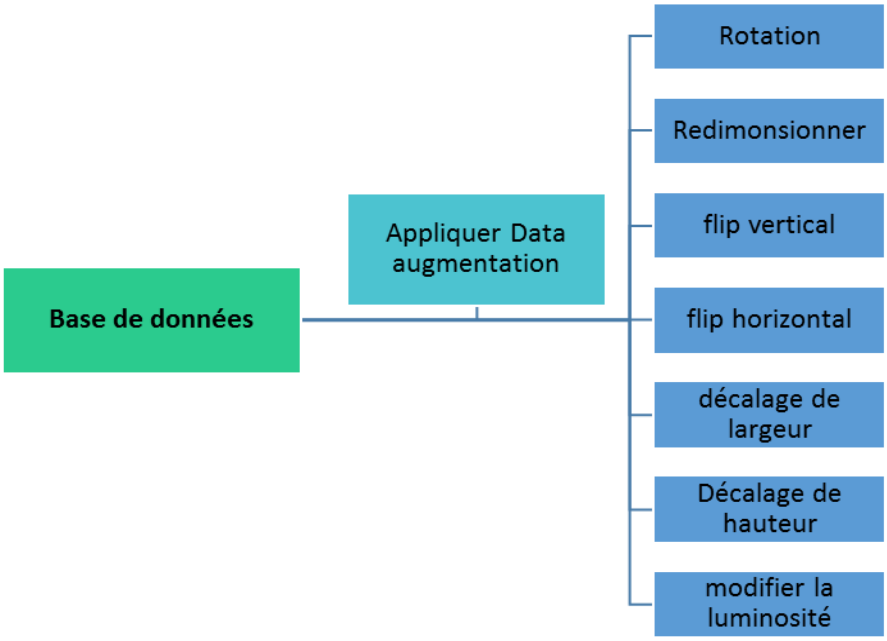


Figure III-2: Schéma synoptique des types de la technique d'augmentation des données

Pour chacune de ces techniques, nous spécifions également le facteur d'augmentation de la taille de l'ensemble de données, appelé (Data Augmentation Factor).

Le facteur définit pour notre travail : Facteur d'augmentation de données = 21, cela donne au total : 21 * 253 images = 5313 images, ce qui veut dire que notre nouvelle base de données générée contient 5313 images qui seront par la suite utilisées pour l'entrainement des réseaux.

Notre choix de facteur était arbitraire mais il peut varier selon le cas et en tenant compte des conditions de chaque application sur laquelle on travaille.

III.4. Le VGG Image Annotator (VIA)

Le logiciel VIA est un projet open source, Léger, autonome et hors ligne qui ne nécessite aucune installation ou configuration et fonctionne uniquement dans un navigateur Web. En raison de sa légèreté et de sa flexibilité, il est rapidement devenu un outil essentiel et inestimable de soutien à la recherche dans de nombreuses disciplines académiques [43].

Ce choix de plateforme nous a permis de construire un outil d'annotation d'images manuelles flexible avec les capacités suivantes:

- formes de régions prises en charge: rectangle, cercle, ellipse, polygone, point et polyligne (ligne polygonale).
- Importation des données de région (region data) au format de fichier csv et json et Exportation au format de csv.
- prend en charge la mise à jour en masse des annotations dans la vue de la grille d'images.
- mise à jour rapide des annotations à l'aide de l'éditeur d'annotations sur image

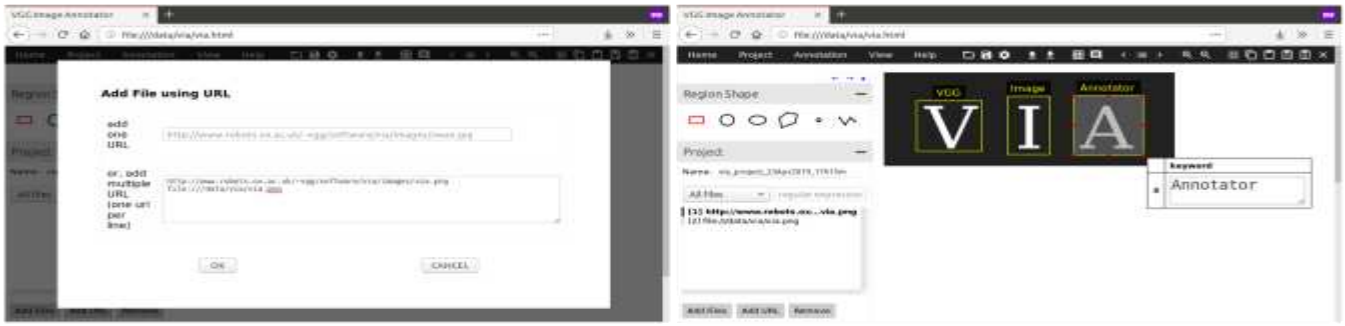


Figure III-3: Le logiciel VIA fonctionne comme une application hors ligne sur un navigateur Web.

Il peut fonctionner avec des images stockées sur un ordinateur local ainsi que sur des images stockées sur des serveurs distants (des sites).

Les annotateurs humains peuvent définir des régions d'image (par exemple une région en forme de polygone) et décrire ces régions à l'aide de l'éditeur d'annotation sur image. Dans la figure III-14 un exemple d'annotations :

Figure III-4: Options d'annotation avec le logiciel VIA

Les Images annotées seront utilisées dans la deuxième partie d'implémentation (Détection de région de la tumeur) et non pas dans la première partie (Classification des images).

III.5. Transfer learning avec le Deep Learning

Le Transfer Learning [44] est le processus de:

1. Prendre un réseau pré-entrainé sur un ensemble de données
2. Et l'utiliser pour reconnaître les catégories d'images / d'objets sur lesquelles il n'a pas été entrainé.

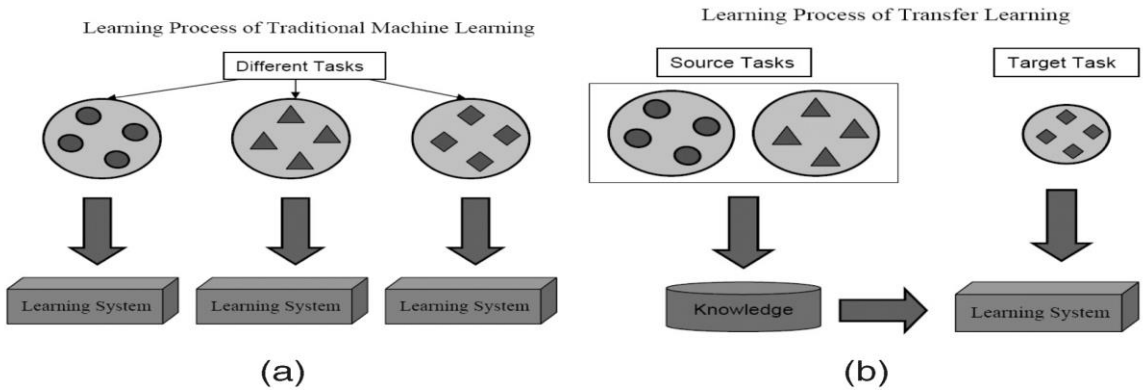


Figure III-5: la différence entre le transfer learning et le machine learning

Avec le Transfer Learning, nous pouvons prendre un modèle pré-entrainé, qui a été entrainé sur un grand ensemble de données (entrainé sur une tâche complètement différente, avec la même entrée mais une sortie différente). Ensuite, nous essayons de trouver des couches qui produisent des caractéristiques réutilisables. Nous utilisons la sortie de cette couche comme caractéristiques d'entrée pour former un réseau beaucoup plus petit qui nécessite un plus petit nombre de paramètres. Ce réseau plus petit n'a besoin que d'apprendre les relations pour notre problème spécifique, ayant déjà appris les modèles dans les données du modèle préformé.

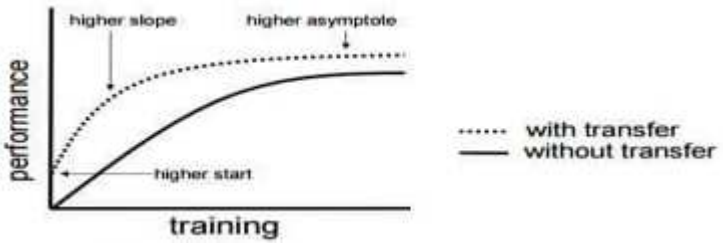


Figure III-6: différence de performance avec et sans le Transfer Learning

En général, il existe deux types de Transfer Learning dans le contexte de Deep Learning:

1. Transfer Learning via l'extraction de caractéristiques (**feature extraction**)
2. Transfer Learning via un réglage fin (**fine-tuning**)

Lors de l'extraction de caractéristiques, nous traitons le réseau préformé comme un extracteur de caractéristiques arbitraire, permettant à l'image d'entrée de se propager, s'arrêtant à une couche prédéfinie et prenant les sorties de cette couche comme nos caractéristiques.

Le réglage fin (fine-tuning), d'autre part, nécessite que nous mettions à jour l'architecture du modèle lui-même en supprimant les têtes de couche entièrement connectées précédentes, en fournissant de nouvelles têtes fraîchement initialisées, puis en entraînant les nouvelles couches entièrement connectées (FC) pour prédire nos classes d'entrée.

III.5.1. Difficultés à mettre en œuvre le Transfer Learning

1. Décider quel modèle à utiliser pour le pré-entrainement

2. Difficile de déboguer lequel des deux modèles ne fonctionne pas

3. Ne pas savoir combien de données supplémentaires suffisent pour entrainer le modèle

4. Difficulté à décider de continuer où cesser d'utiliser le modèle pré-entrainé

5. Décider le nombre de couches et le nombre de paramètres dans le modèle utilisé par-dessus du modèle pré- entrainé

III.5.2. Les avantages de Transfer Learning

1: Il n'est pas nécessaire d'avoir un ensemble de données de formation extrêmement volumineux.

2: Peu de puissance de calcul est requise, car nous utilisons des poids pré-entrainés et n'avons qu'à apprendre les poids des dernières couches.

III.6. Structures modifiées des réseaux utilisées

III.6.1. Le modèle VGG 16

Les dernières couches du modèle VGG16 sont constituées de deux couches « fully connected » avec la fonction « ReLu » et une troisième couche « fully connected » suivie par le « Softmax ».

Pour approprier ce réseau à notre travail, on va appliquer la méthode de Transfer learning, On garde les mêmes poids des premières couches du réseau original mais on enlève les trois dernières couches « fully connected ».

Ensuite, on ajoute notre propre couche « fully connected » avec « Sigmoïde » comme fonction d'activation et aussi le classifieur « Softmax ». Les poids au niveau de ces couches modifiées seront entrainés sur notre base de données. Tout cela le montre la figure III-15 :

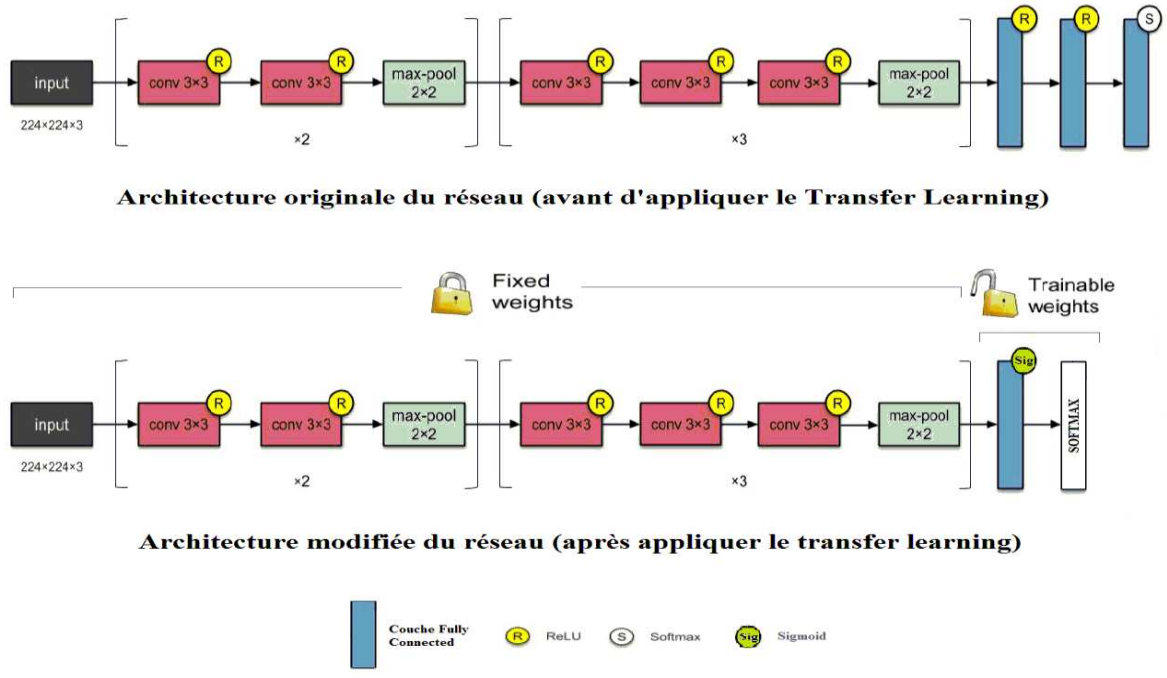


Figure III-7: l'architecture de notre réseau VGG16 après appliquer le Transfer learning

III.6.2. Le modele ResNet 50

Le modèle ResNet 50 original utilise le « Global average pooling » au niveau de ses dernières couches il et il se termine par une couche « fully connected » avec le « Softmax ».

On va modifier sa structure et appliquer le Transfer learning en gardant les mêmes poids dans les premières couches du réseau, puis on change le type du pooling vers le « Max pooling » et on enlève la dernière couche « fully connected », On ajoute a ca place notre propre couche « fully connected » avec la fonction d'activation « Sigmoïde ». On ajoutera aussi le classifieur « Softmax ». Les poids au niveau de ces couches modifiées seront entrainés sur notre base de données. Tout cela le montre la figure III-16:

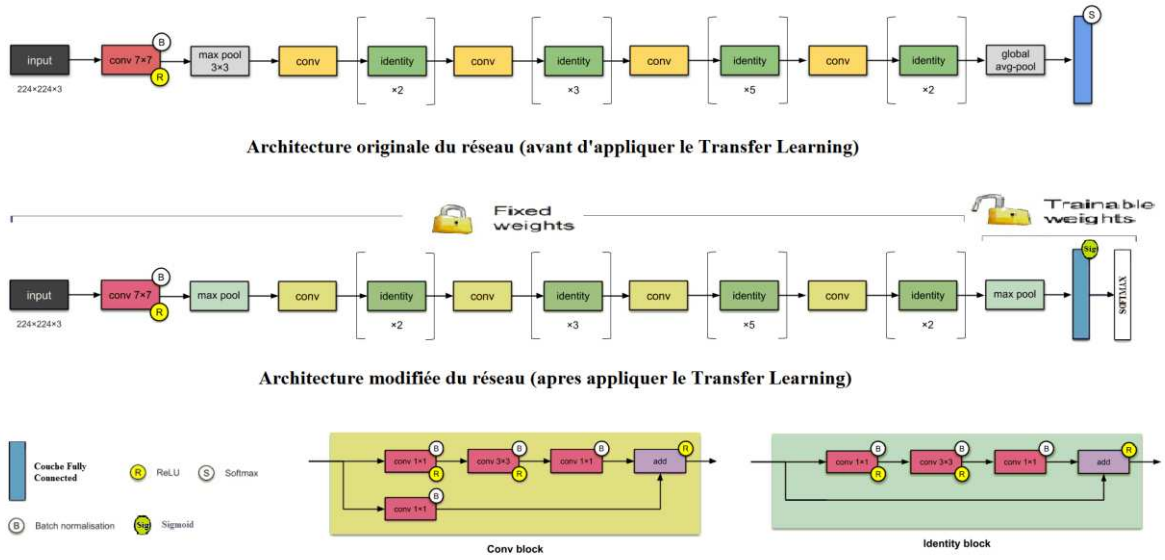


Figure III-8: l'architecture de notre réseau ResNet après appliquer le Transfer learning

III.6.3. Le modèle Inception

Comme le montre la figure III-17, le réseau Inception original est constitué au niveau de ses dernières couches de trois couches « fully connected » et se termine par le « Softmax ».

Pour modifier son architecture, on va appliquer la méthode de Transfer Learning en gardant les mêmes poids des premières couches du réseau et en enlèvant les trois dernières couches « fully connected » et le « Softmax ». Ensuite, comme fait avec les réseaux précédents, on ajoute notre propre couche « fully connected » avec la fonction d'activation « Sigmoïde » et le classifieur « Softmax » à la fin. Les poids de ces derniers seront entrainés.

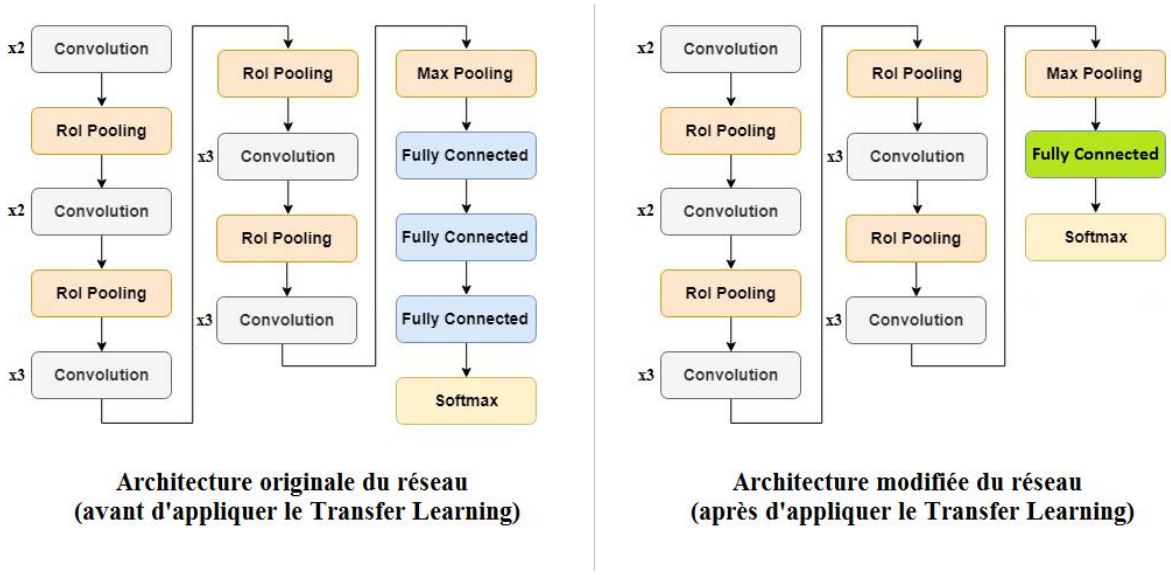


Figure III-9: l'architecture de notre réseau Inception après appliquer le Transfer learning

III.6.4. Le modèle Mask R-CNN

Notre réseau est constitué d'un réseau de ResNet qui est la base du bloc FPN « Feature Pyramid Network » et un bloc RPN « Region Proposal Network » qui est un réseau plus léger que celui du premier Bloc. Ces 2 blocs sont suivis d'un bloc « ROI ALIGN » qui localise les zones pertinentes de la carte des caractéristiques, il y a deux branches sortants de ce dernier, une première branche qui allant vers des couches « fully connected » suivies par un classifieur « softmax » pour générer les classes et un « Boundary Box regressor » en sortie, Et une deuxième branche qui sort de la même branche vers deux convolutions qui va générer des masques pour chaque objet au niveau des pixels comme le montre la figure III-18:

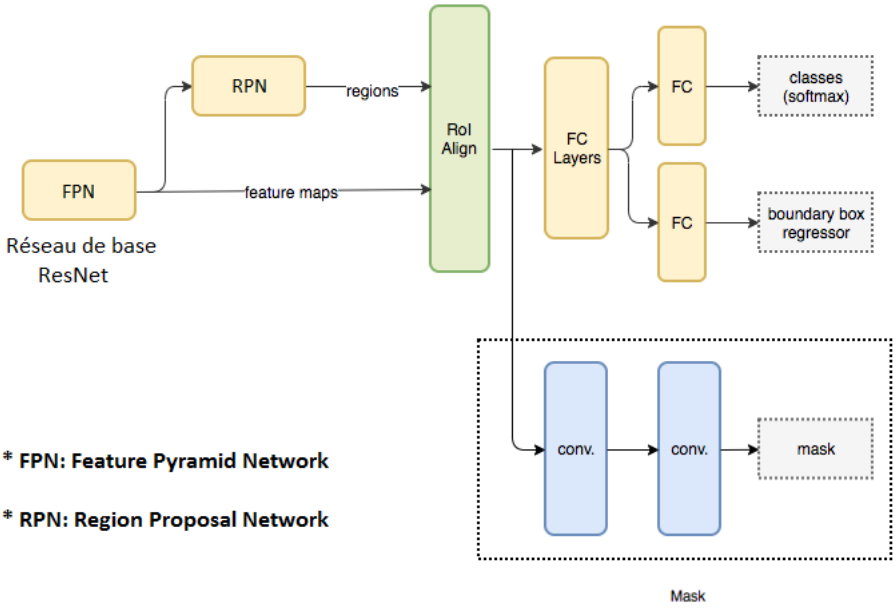


Figure III-10: l'architecture de notre réseau Mask R-CNN utilisé pour la détection

III.7. Paramètres d'entraînement

L'entrainement d'un réseau nécessite la configuration minutieuse de plusieurs propriétés de modèle et d'hyper paramètres de ce dernier et de faire de nombreux choix apparemment arbitraires tels que le nombre et les types de couches, les taux d'apprentissage, les ensembles de formation et de test, etc.

III.7.1. Le classifieur « Softmax »

Une fonction Softmax est un type de fonction d'écrasement. Les fonctions d'écrasement limitent la sortie de la fonction dans la plage de 0 à 1. Cela permet à la sortie d'être interprétée directement comme une probabilité.
La fonction Softmax génère un vecteur qui représente les distributions de probabilité d'une liste de résultats potentiels [45].

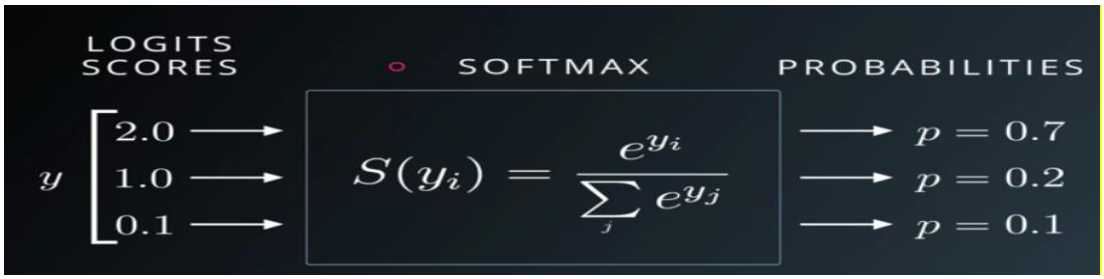


Figure III-11: exemple du classifieur « Softmax »

Une couche Softmax est généralement la couche finale utilisée dans les fonctions du réseau neuronal. Il est important de noter que cette couche doit avoir le même nombre de nœuds que la sortie ultérieure.

Softmax transforme les logits (sortie numérique de la dernière couche linéaire d'un réseau neuronal de classification à classes multiples) en probabilités en prenant les exposants de chaque sortie, puis en normalisant chaque nombre par la somme de ces exposants afin que le vecteur de sortie entier soit égal à un.

Softmax est fréquemment ajouté à la dernière couche d'un réseau de classification d'images comme VGG16.

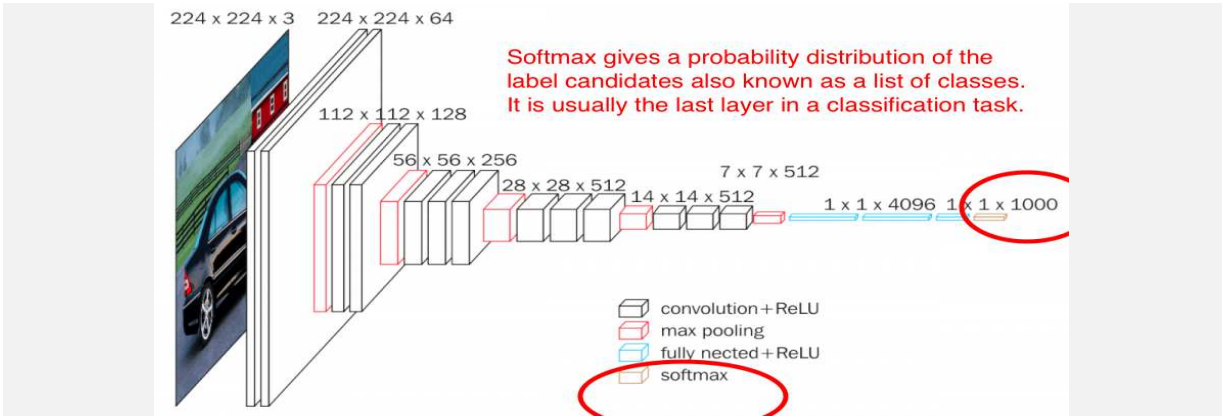


Figure III-12: Exemple d'utilisation du classifieur « Softmax » dans un réseau de neurones

L'entrée de Softmax est la sortie de la couche entièrement connectée qui la précède immédiatement, et elle produit la sortie finale de l'ensemble du réseau neuronal. Cette sortie est une distribution de probabilité de toute la classe d'étiquette.

III.7.2. Fonction de perte (Loss Function)

Dans le contexte d'un algorithme d'optimisation, la fonction utilisée pour évaluer une solution candidate (c'est-à-dire un ensemble de poids) est appelée fonction objective. Nous pouvons chercher à maximiser ou à minimiser la fonction objective, ce qui signifie que nous recherchons une solution candidate qui a respectivement le score le plus élevé ou le plus bas.

Avec les réseaux de neurones, nous cherchons à minimiser l'erreur. La fonction objective est souvent appelée fonction de coût ou fonction de perte et la valeur calculée par la fonction de perte est simplement appelée «perte» [46].

III.7.3. L'entropie croisée (Cross-Entropy)

L'entropie croisée (Cross-entropy) mesure l'entropie relative entre deux distributions de probabilité sur le même ensemble d'événements [47].

On peut penser que l'entropie croisée calcule l'entropie totale entre les distributions.

La perte d'entropie croisée (Cross-entropy loss) est souvent simplement appelée «entropie croisée» (cross-entropy) ou «perte logarithmique» (logarithmic loss).

Chaque probabilité prédite est comparée à la valeur de sortie de classe réelle (0 ou 1) et un score est calculé qui pénalise la probabilité en fonction de la distance de la valeur attendue. La pénalité est logarithmique, offrant un petit score pour les petites différences (0,1 ou 0,2) et un score énorme pour une grande différence (0,9 ou 1,0).

La perte d'entropie croisée est minimisée, où des valeurs plus petites représentent un meilleur modèle que des valeurs plus grandes. Un modèle qui prédit des probabilités parfaites a une entropie croisée ou une perte de log de 0,0.

L'entropie croisée pour un problème de prédiction binaire ou à deux classes est en fait calculée comme l'entropie croisée moyenne dans tous les exemples.

III.7.4. Epoques (Epochs)

Une époque d'apprentissage signifie que l'algorithme d'apprentissage a effectué un passage dans l'ensemble de données d'apprentissage, où les exemples ont été séparés en groupes de «taille de lot» (batch size) sélectionnés aléatoirement [48].

III.7.5. Optimisation des fonctions

L'optimisation des fonctions est un élément fondamental de Deep Learning. La plupart des algorithmes de Deep Learning impliquent l'optimisation des paramètres (poids, coefficients, etc.) en réponse aux données d'apprentissage. L'optimisation fait également référence au processus de recherche du meilleur ensemble d'hyperparamètres qui configure l'entrainement d'un algorithme de Deep Learning.

a. Taux d'apprentissage (Learning Rate)

Le taux d'apprentissage peut être l'hyper paramètre le plus important lors de la configuration des réseaux neuronaux. Il contrôle le degré de changement du modèle en réponse à l'erreur estimée chaque fois que les poids du modèle sont mis à jour [49].

Le choix du taux d'apprentissage est difficile car une valeur trop petite peut entraîner un long processus de formation qui pourrait rester bloqué, tandis qu'une valeur trop grande peut entraîner l'apprentissage d'un ensemble de poids sous-optimal trop rapidement ou un processus instable d'entrainement.

b. Optimisation d'Adam

Le nom Adam est dérivé de l'estimation adaptative du moment (adaptive moment estimation). C'est un algorithme pour l'optimisation du gradient de premier ordre des fonctions objectives, basé sur des estimations adaptatives des moments d'ordre inférieur [50].

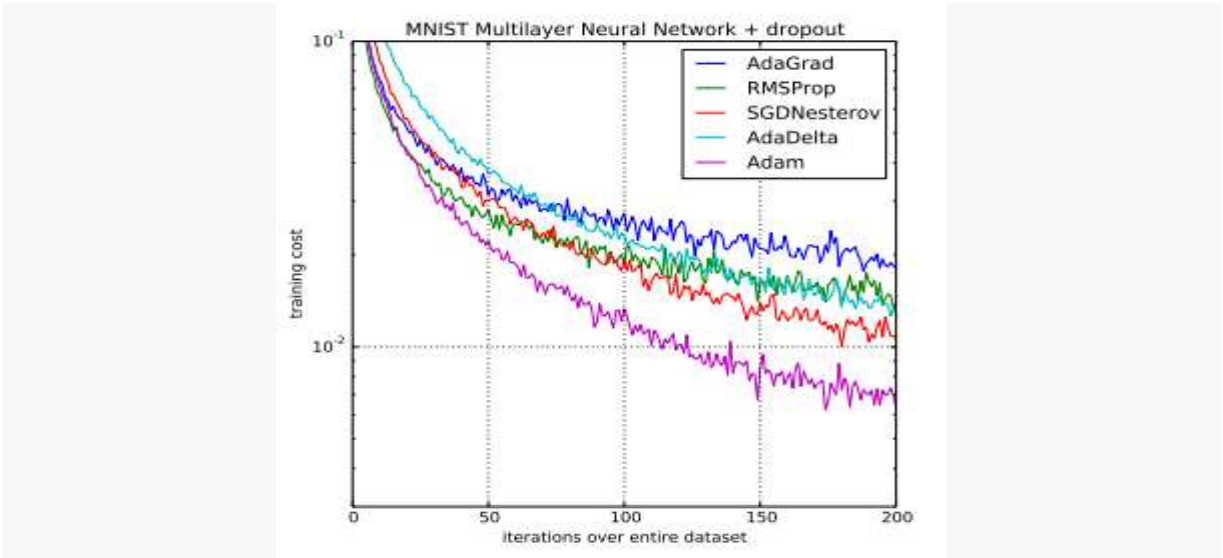


Figure III-13: Exemple de comparaison d'Adam à d'autres algorithmes d'optimisation

Les différents paramètres de configuration d'Adam sont :

Alpha: taux d'apprentissage ou taille de pas. La proportion de pondérations mises à jour. Des valeurs plus élevées entraînent un apprentissage initial plus rapide avant la mise à jour du taux. Des valeurs plus petites ralentissent l'apprentissage tout au long de l'entrainement.

beta1 : Le taux de décroissance exponentielle pour les premières estimations de moment.

beta2 : Le taux de décroissance exponentielle pour les estimations du deuxième moment. Cette valeur doit être proche de 1,0 pour les problèmes avec un gradient clairsemé.

epsilon: Est un très petit nombre pour empêcher toute division par zéro dans la mise en œuvre.

Les bons paramètres par défaut pour les problèmes de Deep Learning sont alpha = 0,001, beta1 = 0,9, beta2 = 0,999 et epsilon = 10−8, mais ce n'est pas une obligation de les utiliser.

III.7.6. La Précision (Accuracy)

La moyenne des performances d'un modèle sur plusieurs exécutions donne une idée des performances moyennes du modèle spécifique sur l'ensemble de données spécifique. L'écart ou l'écart type des scores donne une idée de la variance introduite par l'algorithme d'apprentissage [51].

La performance moyenne du modèle est rapportée pour chaque taille d'ensemble d'entraînement (training set), montrant une amélioration constante de la précision du test à mesure que l'ensemble d'entraînement augmente, comme nous nous y attendons.

III.8. Le langage de programmation utilisé (Python)

Python est le langage de programmation le plus utilisé dans le domaine de l'intelligence artificielle et surtout en Deep Learning vu qu'il contient un nombre important de bibliothèques performantes et utiles pour la vision artificielle et l'utilisation des réseaux de neurones. Notre projet est fait avec le langage de Python dans le cloud et on n'aura pas besoin d'installer un logiciel ou un IDE dans notre ordinateur.

III.9. Les bibliothèques Python

III.9.1. NumPy

C'est l'une des bibliothèques les plus essentielles lorsqu'on travaille sur des projets en Python. Elle permet des manipulations de tableaux de grande dimension comme les opérations arithmétiques, les conversions, etc.

III.9.2. Matplotlib

nous utilisons cette bibliothèque pour une représentation visuelle des résultats. Elle peut faire des tonnes de choses avec les données, comme les fonctions de tracé, les graphiques à barres, etc. Nous l'utilisons quand nous devons tracer quelques images pour faire des comparaisons.

III.9.3. Pandas

Cette bibliothèque est utilisée lorsqu'on a besoin d'organiser nos données et de les rendre plus faciles à utiliser et à comprendre. Elle les organise dans des structures appelées « trames de données » et celles-ci aideront avec une représentation claire des données ainsi que leur personnalisation.

III.9.4. SciPy

C'est une bibliothèque Python open source pour le calcul scientifique et technique. Les principaux packages incluent des modules pour: l'optimisation, l'algèbre linéaire, l'intégration, la FFT, le traitement du signal et de l'image.

III.9.5. Scikit-learn

Une bibliothèque Python open source qui propose de nombreuses classifications, des algorithmes de régression et de regroupement comme les support vector machines (SVM), le random forests, le gradient boosting, les k-means et DBSCAN. Elle est utilisée lorsque la création et la formons des modèles de Machine Learning.

III.9.6. OpenCV

C'est l'une des meilleures bibliothèques de vision par ordinateur et de traitement d'image utilisée pour faire des tonnes de choses sur des images et des vidéos. Elle met à disposition de nombreuses fonctionnalités très diversifiées pour le traitement.

III.9.7. Tensorflow

TensorFlow est une bibliothèque de Deep Learning développée par l'équipe Google Brain de Google, il s'agit d'une boîte à outils permettant de résoudre des problèmes mathématiques extrêmement complexes avec aisance. Elle permet de développer des architectures d'apprentissage expérimentales.

III.9.8. Keras

Keras est une API de réseaux neuronaux de haut niveau, écrite en Python et capable de s'exécuter sur TensorFlow, CNTK ou Theano. Il a été développé dans le but de permettre une expérimentation rapide.

Elle prend en charge les réseaux convolutionnels et les réseaux récurrents, ainsi que les combinaisons des deux.

Keras est compatible avec les versions de Python 2.7 / 3.6 et autres.

Keras a été initialement développé dans le cadre de l'effort de recherche du projet ONEIROS (système d'exploitation de robot intelligent neuro-électronique à composition open-ended).

III.10. Google Colab

Google Colab ou « Colaboratory» est un service cloud gratuit hébergé par Google pour encourager la recherche sur le Deep Learning et l'Intelligence Artificielle, où souvent l'obstacle à l'apprentissage et au succès est l'exigence d'une puissance de calcul considérable. Cette plateforme permet alors d'entraîner des modèles de Deep Learning directement dans le cloud. Sans donc avoir besoin d'installer quoi que ce soit sur l'ordinateur à l'exception d'un navigateur [52].

Colab permet :

- De développer des applications en Deep Learning en utilisant des bibliothèques Python populaires telles que Keras, TensorFlow, PyTorch et OpenCV.
- D'utiliser un environnement de développement (Jupyter Notebook) qui ne nécessite aucune configuration.
- L'accès à un processeur graphique GPU et au TPU (Tensor Processor Unit) gratuitement.

Google Colaboratory fonctionne sur les serveurs Google donc nous n'avons rien installé.
De plus, les documents Colab (Jupyter Notebook) sont enregistrés directement sur le compte Google Drive.

III.10.1. Avantages de Colab

En plus d'être facile à utiliser, le Colab est assez flexible dans sa configuration et fait beaucoup de travail pour nous.

• Prise en charge de Python 2.7 et Python 3.6

• Accélération GPU gratuite

• Bibliothèques préinstallées: les principales bibliothèques de Python comme TensorFlow, Scikit-learn, entre autres, sont préinstallées et prêtes à être importées.

• Prend en charge les commandes Bash (langage de commande).

III.10.2. Utiliser la plateforme Colab

Suivre le lien **https://colab.research.google.com/** et se connecter avec le compte gmail

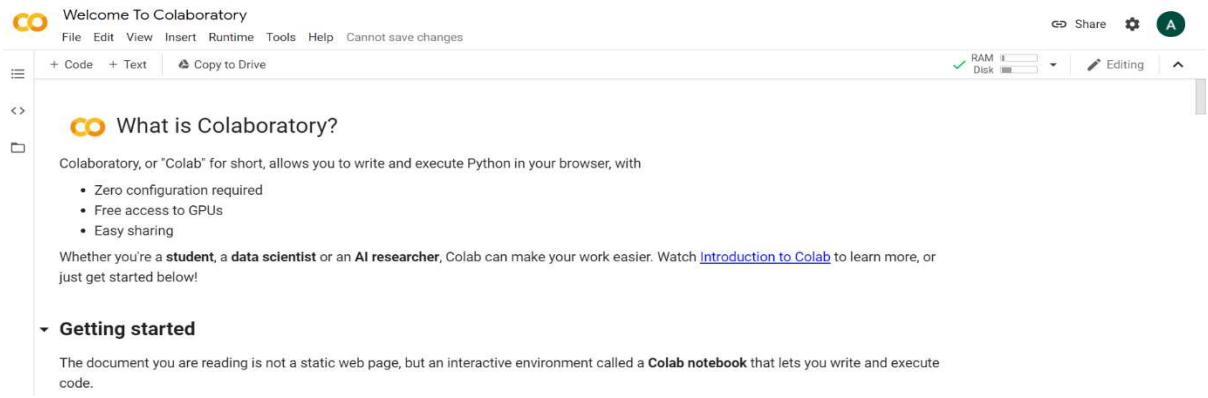


Figure III-14 : la page d'accueil de la plateforme Colab

Sélection de : nouveau bloc-notes Python3 (New Notebook)

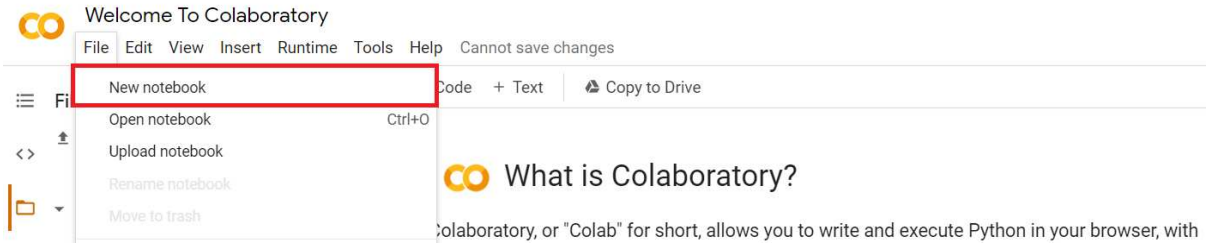


Figure III-15: Illustration de comment ouvrir un nouveau Notebook dans la plateforme Colab

Maintenant on est prêt pour commencer à taper du code dans les cellules de code

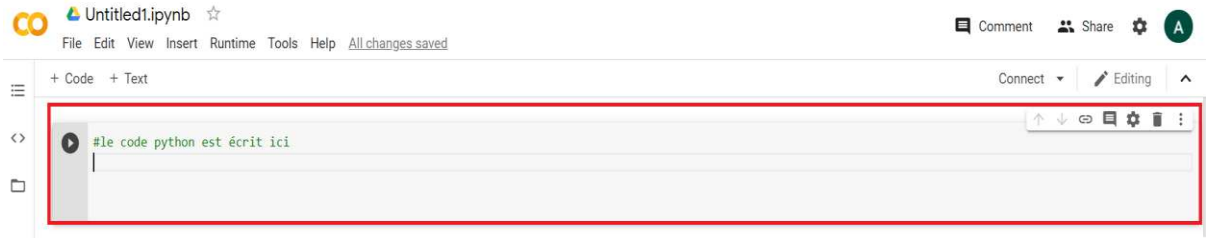


Figure III-16: illustration du nouveau Notebook ouvert dans la plateforme Colab

Pour ajouter une nouvelle cellule: Insert-> Code Cell ou bien cliquer sur l'icône +Code

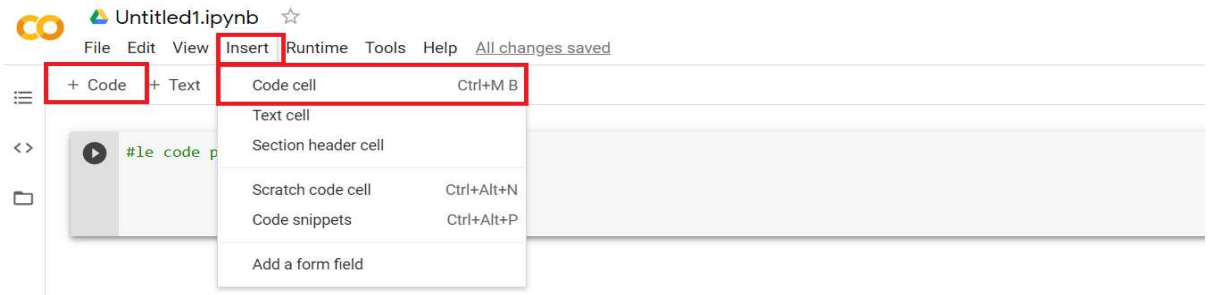


Figure III-17: illustration pour montrer comment insérer une cellule de code dans Colab

Pour exécuter une cellule particulière, on la sélectionne et on appuie sur les touches Ctrl + Entrer ou bien l'icône correspondante, pour exécuter toutes les cellules Runtime -> Run all

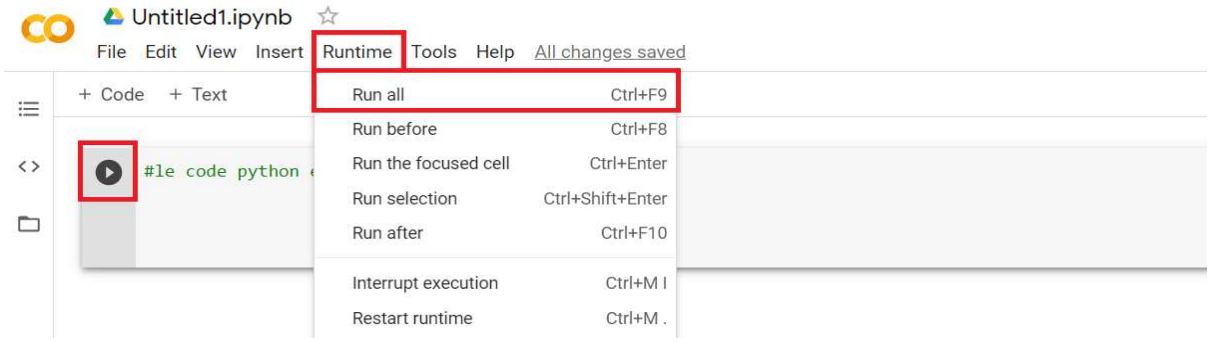


Figure III-18: illustration pour montrer comment exécuter toutes les cellules de codes

III.10.3. Installer des bibliothèques/packages Python

Bien que la plupart des bibliothèques Python couramment utilisées soient préinstallées, de nouvelles bibliothèques peuvent être installées à l'aide des syntaxes suivantes:

!pip install [nom du package]
OU
!apt-get install [nom du package]

Exemples :
!pip install -q matplotlib-venn

!apt-get -qq install -y libfluidsynth1

a. Importer les bibliothèques Python (Python Libraries) :

L'import des bibliothèques Python se fait de la même manière que sur les notebook jupyter:

import pandas as pd

III.10.4. Manipuler les fichiers

Importer (upload) des fichiers locaux (depuis l'ordinateur)

from google.colab import files

uploaded = files.upload()

On sélectionne « Choisir un fichier » (Choose file) et on importe le fichier souhaité.

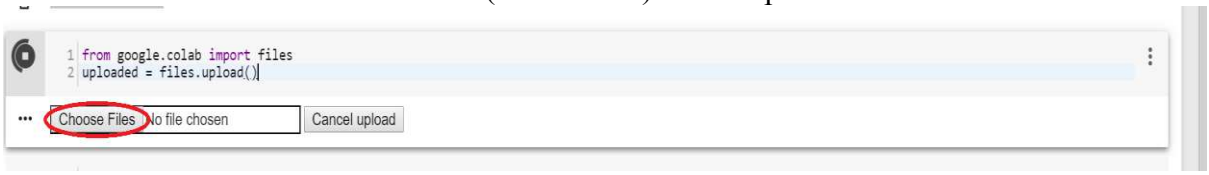


Figure III-19: Illustration : comment importer les fichiers dans colab

III.10.5. Utiliser les fichiers depuis google drive

Pour monter le Drive dans le dossier «gdrive», on exécute la commande:

from google.colab import drive

drive.mount('/content/gdrive')

Ensuite, un lien sera généré, il faut cliquer sur le lien, autoriser l'accès puis copier le code qui apparaît et le coller dans "Entrez votre code d'autorisation".

Maintenant, pour voir toutes les données de Google Drive, on devra exécuter ce qui suit:

!ls "/content/gdrive/My Drive/"

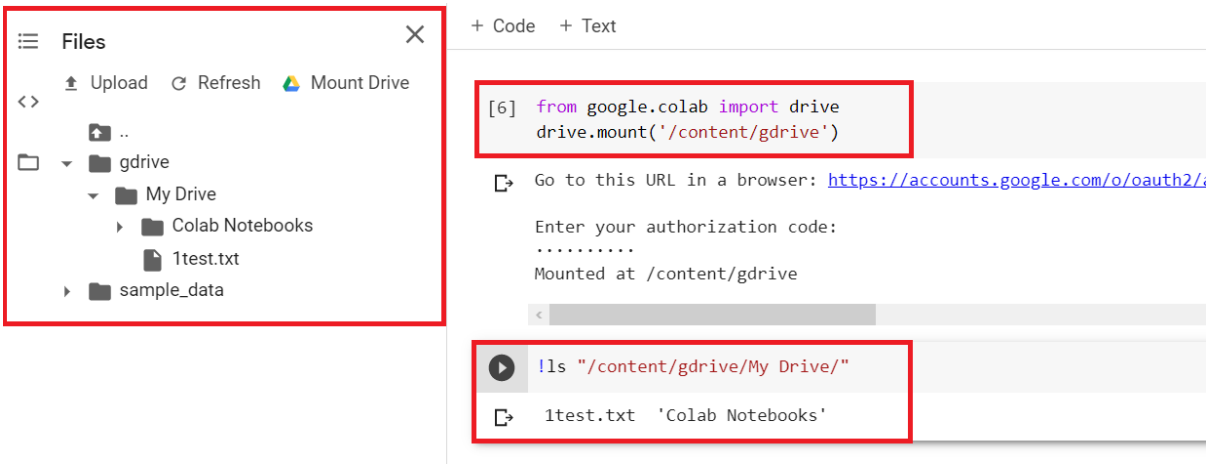


Figure III-20: visualisation du contenu de notre google drive

III.10.6. Entrainer sur GPU

Pour passer en mode GPU, dans la barre des options choisir “exécution”(Runtime) puis “modifier le type d’exécution” (Change Runtime Type) et mettre l’option accélérateur matériel (Hardware Accelerator) en mode GPU.

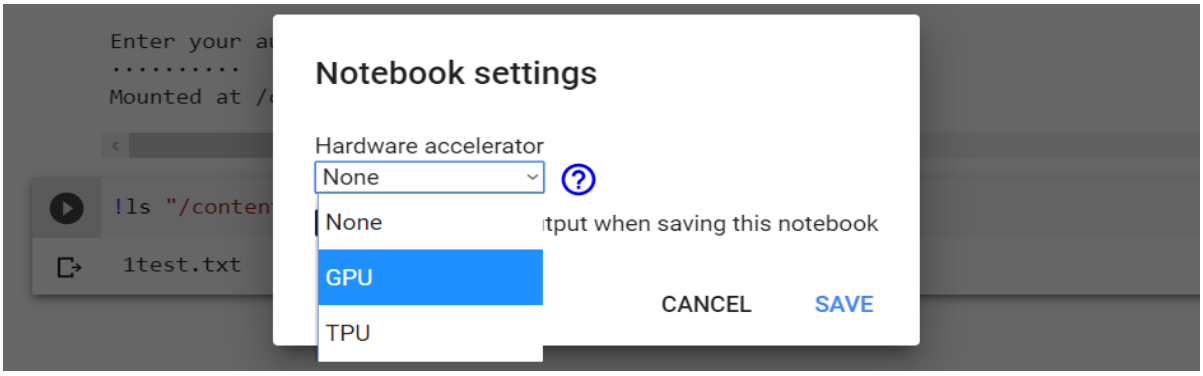


Figure III-21: illustration pour montrer comment configurer le type d’exécution

III.10.7. Entrainer sur TPU

Le TPU (Tensor Processing Unit) est un circuit intégré, développé par Google spécifiquement pour accélérer les systèmes d’intelligence artificielle. Cette architecture est faite pour optimiser les calculs d’entrainement d’un réseau de neurones où d’autres applications puissantes en terme de traitement.

Pour utiliser le “TPU” sur Google Colab il suffit de suivre les instructions précédentes pour l’activation du GPU mais choisir TPU comme Accélérateur matériel.

III.10.8. Vérifier les caractéristiques du CPU et de la RAM

!cat /proc/cpuinfo

!cat /proc/meminfo

III.10.9. Vérifier les caractéristiques du GPU

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

III.11. Conclusion

Dans ce troisième chapitre, nous avons présenté l'environnement du Cloud sous lequel on va travailler, ainsi les différentes bibliothèques et d'autres plateformes exploitées pour réussir à préparer notre base de données. On a mis en valeur une partie importante sur laquelle notre projet se base « le Transfer Learning » dont on a montré son principe et les modifications nécessaires qu'on a apporté aux réseaux de neurones utilisés et on a défini les différents paramètres qu'on va configurer afin de réussir à implémenter nos modèles.

Dans le prochain chapitre, nous présentons la partie pratique, les étapes d'installations et utilisation des différents outils nécessaires en arrivant aux résultats de notre implémentation et leur discussion.

Chapitre 4: Implémentation et Résultats

Chapitre 4 Implémentation et Résultats

IV.1. Introduction

Ce dernier chapitre met en application les méthodes (ou concepts) présentées dans les chapitres précédents. Dans un premier temps, on prépare l'environnement du cloud et on installe les bibliothèques nécessaires, puis, on importe la base de données et les réseaux de neurones utilisés. On passe ensuite à l'implémentation des différents réseaux créés comme indiqué dans le chapitre trois (Transfer Learning), On parle ici des réseaux modifiés pour faire la classification des IRM (VGG 16, ResNet 50 et Inception V3)et aussi du réseau de détection des régions de tumeurs (Mask R-CNN). Nous décrivons les étapes d'implémentation, puis nous comparons et discutons les résultats obtenus.

IV.2. Implémentation des réseaux de classification

Afin d'implémenter nos réseaux de neurones pour pouvoir classer les images IRM on va suivre les étapes montrées dans la figure IV-1:

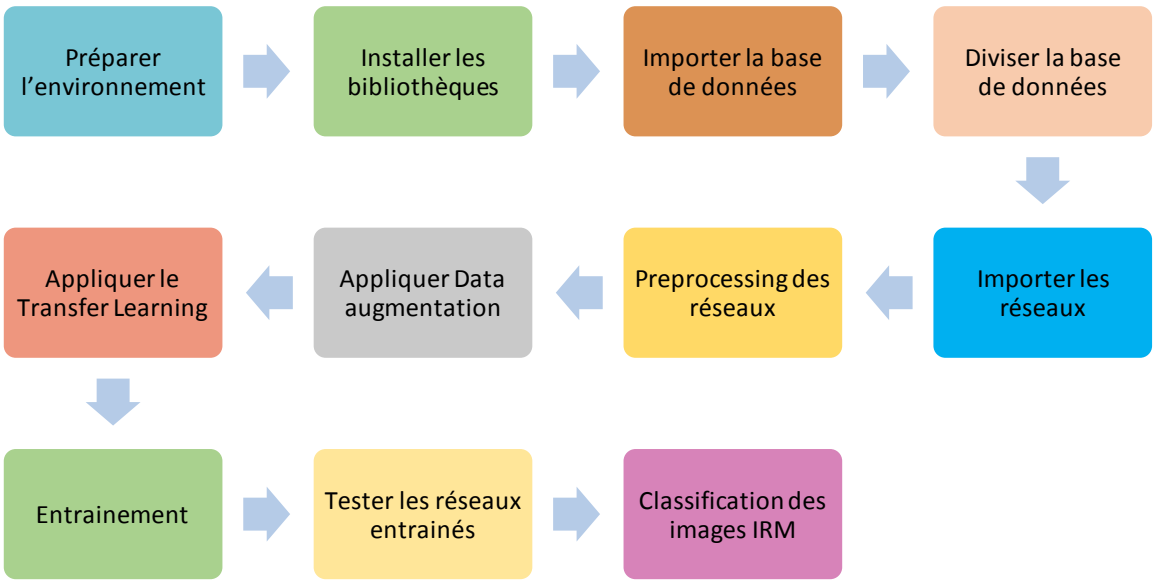


Figure IV-1 :Les étapes d'implementation des réseaux de classification

IV.2.1. Préparer l'environnement

D'abord on prépare notre environnement. On va utiliser les modèles VGG 16, Inception V3 et Resnet50 pour la classification.

```
from IPython.display import clear_output
!pip install imutils
clear_output()

from keras.applications.vgg16 import VGG16,preprocess_input
from keras.applications.inception_v3 import InceptionV3,inception_v3
from keras.applications.resnet50 import ResNet50,resnet50
```

```
Using TensorFlow backend.
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depre
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depre
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depre
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depre
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
```

Figure IV- 2 : code de préparation des réseaux et de l'environnement

IV.2.2. Installer les bibliothèques

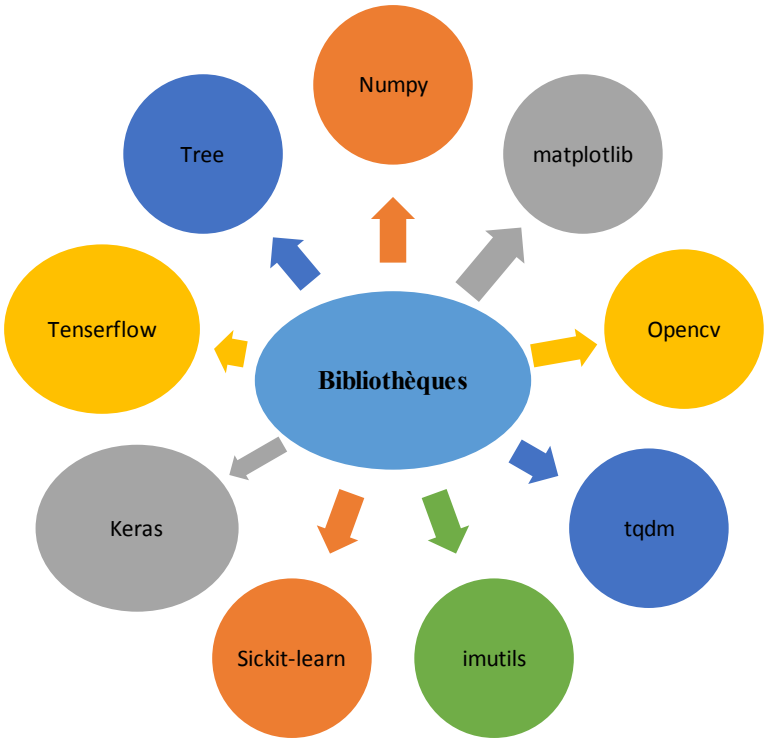


Figure IV-3 : Les bibliothèques utilisées

```
!pip install numpy==1.18.4
!pip install tqdm==4.41.1
!apt-get install matplotlib
!pip install matplotlib
!pip install scipy
!pip install sklearn
import shutil
```

```
Requirement already satisfied: numpy==1.18.4 in /usr/local/lib/python3.6/dist-packages (1.18.4)
Requirement already satisfied: tqdm==4.41.1 in /usr/local/lib/python3.6/dist-packages (4.41.1)
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package matplotlib
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (3.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.6/dist-packages (from matplotlib) (1.18.4)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from cycler>=0.10->matplotlib) (1.12.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.6/dist-packages (from scipy) (1.18.4)
Requirement already satisfied: sklearn in /usr/local/lib/python3.6/dist-packages (0.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (from sklearn) (0.22.2.post1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->sklearn) (0.15.1)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->sklearn) (1.4.1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->sklearn) (1.18.4)
```

Figure IV- 4 : code d'installation des bibliothèques

Vérifier la version préinstallée d'Opencv sur Colab :

```
import cv2
cv2.__version__
```

```
'4.1.2'
```

Figure IV - 5 : exemple de vérification de la version d'une bibliothèque pré installée

Dans cette partie on importe les bibliothèques qu'on a installé et aussi celles qui sont préinstallées :

```
import numpy as np
from tqdm import tqdm
import cv2
import os
import shutil
import itertools
import imutils
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras import layers
from keras.models import Model, Sequential
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping
```

```
init_notebook_mode(connected=True)
RANDOM_SEED = 123
```

Figure IV – 6 : code d'importation des bibliothèques et les paramètres d'entrainement

IV.2.3. Créer les dossiers de la base de données

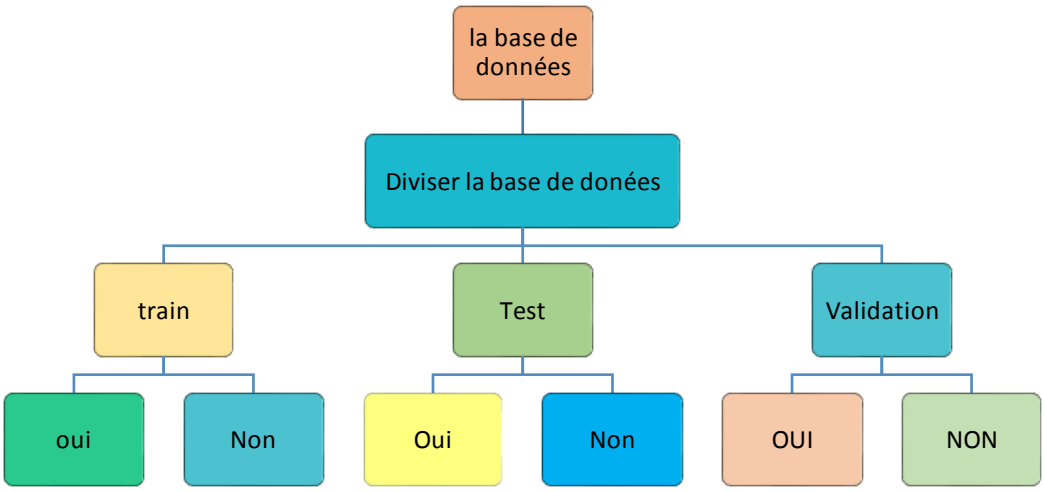


Figure IV- 7 : Division de la base de données

On installe le package de « tree == 1.7.0.5 » pour nous permettre de créer les dossiers qui vont contenir les images de la base de données :

```
!apt-get install tree
!mkdir TRAIN TEST VAL TRAIN/YES TRAIN/NO TEST/YES TEST/NO VAL/YES VAL/NO
!tree -d
```

```
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 12 not upgraded.
Need to get 46.1 kB of archives.
After this operation, 106 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian stretch/main amd64 tree amd64 1.7.0-5 [46.1 kB]
Fetched 46.1 kB in 0s (1382 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package tree.
(Reading database ... 59251 files and directories currently installed.)
Preparing to unpack .../tree_1.7.0-5_amd64.deb ...
Unpacking tree (1.7.0-5) ...
Setting up tree (1.7.0-5) ...
.
├── TEST
│   ├── NO
│   └── YES
├── TRAIN
│   ├── NO
│   └── YES
└── VAL
    ├── NO
    └── YES

9 directories
```

Figure IV – 8 : code d'installation du package tree et créer ses chemins de dossiers

La prochaine étape c'est d'importer la base de données a Colab, Cela peut se faire de manières différentes, on en cite 2:

- manuellement (Files => upload => et on sélectionne nos données) puis on mets les données dans chaque dossier précis (TEST, TRAIN, OUI...) comme le montre la figure précédente.

-par des lignes de code pour importer la base de données à partir PC local ou d'une URL.

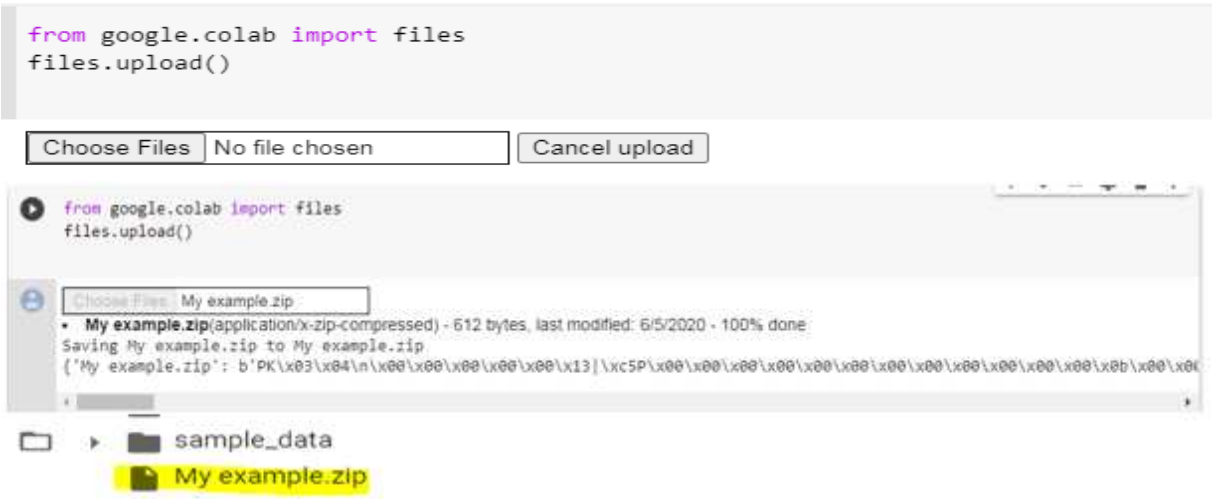


Figure IV-9 : code d'importation de la base de données

Les données sont compressées, maintenant il faut les décompresser et récuperer les dossiers :

```
from zipfile import ZipFile
file_name = "My example.zip"
with ZipFile (file_name,'r') as zip:
  zip.extractall()
  print('done')
```

```
done
```

Figure IV- 10 : code d'extraction du fichier compressé

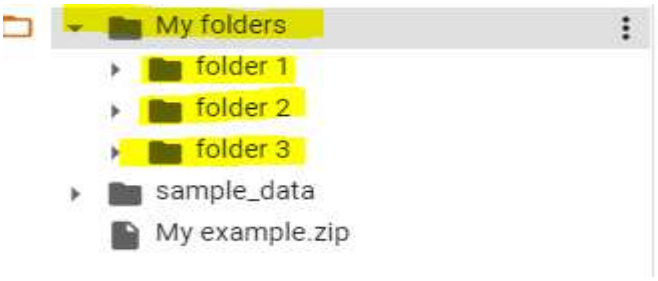


Figure IV- 11 : Récupération des données compressées

Les dossiers ont bien été éxtraits, maintenant avec la meme façon on importe notre base de données au Cloud.

IV.2.4. Diviser la base de données et faire le preprocessing

Dans ce qui suit, les images dans la base de données sont copiées et divisées dans des dossiers differents selon nos besoins pour l'apprentissage du 75% des images dans TRAIN et 25% des images dans les dossier TEST et VAL

```
IMG_PATH = '../input/mri-images/brain_tumor_dataset/'
for CLASS in os.listdir(IMG_PATH):
    if not CLASS.startswith('.'):
        IMG_NUM = len(os.listdir(IMG_PATH + CLASS))
        for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH + CLASS)):
            img = IMG_PATH + CLASS + '/' + FILE_NAME
            if n < 5:
                shutil.copy(img, 'TEST/' + CLASS.upper() + '/' + FILE_NAME)
            elif n < 0.8*IMG_NUM:
                shutil.copy(img, 'TRAIN/'+ CLASS.upper() + '/' + FILE_NAME)
            else:
                shutil.copy(img, 'VAL/'+ CLASS.upper() + '/' + FILE_NAME)
```

Figure IV- 12: code de division de la base de données

Les données (images) passent encore par un autre programme qui lit leurs noms, chemins et leur classe (YES ou NO)

```
def load_data(dir_path, img_size=(100,100)):
    X = []
    y = []
    i = 0
    labels = dict()
    for path in tqdm(sorted(os.listdir(dir_path))):
        if not path.startswith('.'):
            labels[i] = path
            for file in os.listdir(dir_path + path):
                if not file.startswith('.'):
                    img = cv2.imread(dir_path + path + '/' + file)
                    X.append(img)
                    y.append(i)
            i += 1
    X = np.array(X)
    y = np.array(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels
```

Figure IV-13 : Lecture des chemins de la base de données

Définition de la taille des images ([Hauteur * Largeur]) [224 * 224] et chargement de l'ensemble de données.

```
[ ]  TRAIN_DIR = 'TRAIN/'
     TEST_DIR = 'TEST/'
     VAL_DIR = 'VAL/'
     IMG_SIZE = (224,224)

     X_train, y_train, labels = load_data(TRAIN_DIR, IMG_SIZE)
     X_test, y_test, _ = load_data(TEST_DIR, IMG_SIZE)
     X_val, y_val, _ = load_data(VAL_DIR, IMG_SIZE)
```

```
100%|██████████| 2/2 [00:00<00:00,  4.65it/s]
100%|██████████| 2/2 [00:00<00:00, 97.31it/s]
100%|██████████| 2/2 [00:00<00:00, 21.24it/s]193 images loaded from TRAIN/ directory.
10 images loaded from TEST/ directory.
50 images loaded from VAL/ directory.
```

Figure IV-14 : Chargement de la base de données

IV.2.5. Visualisation de la base de données

Maintenant, on va visualiser notre base de données chargée :

```
def plot_samples(X, y, labels_dict, n=50):
    for index in range(len(labels_dict)):
        imgs = X[np.argwhere(y == index)][:n]
        j = 10
        i = int(n/j)

        plt.figure(figsize=(15,6))
        c = 1
        for img in imgs:
            plt.subplot(i,j,c)
            plt.imshow(img[0])

            plt.xticks([])
            plt.yticks([])
            c += 1
        plt.suptitle('Tumor: {}'.format(labels_dict[index]))
        plt.show()
```

Figure IV-15: code de préparation de la visualisation de la base de données

On va afficher quelques échantillons de notre base de données avec leurs classes respectives :

```
plot_samples(X_train, y_train, labels, 10)
```

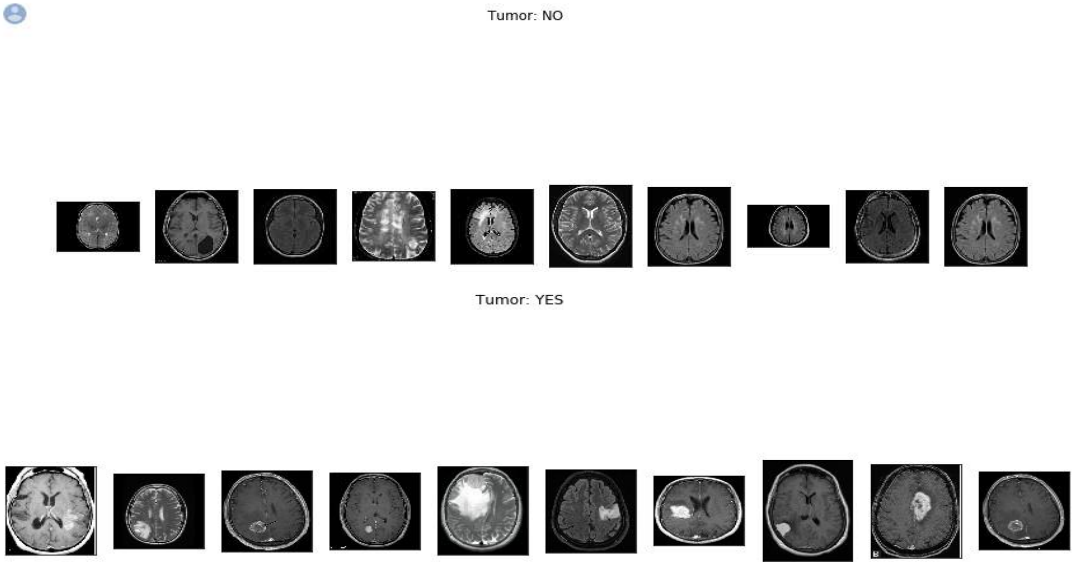


Figure IV-16: Visualisation de quelques échantillons de notre base de données avec leurs classes

IV.2.6. Le prétraitement de la base de données et les réseaux de neurones:

Pour ce qui suit on va appliquer le Preprocessing à la base de données qu'on va l'introduire à nos réseaux de neurones

```
def preprocess_imgs(set_name, img_size):
    set_new = []
    for img in set_name:
        img = cv2.resize(
            img,
            dsize=img_size,
            interpolation=cv2.INTER_CUBIC
        )
        set_new.append(preprocess_input(img))
    return np.array(set_new)
X_train_prep = preprocess_imgs(set_name=X_train_crop, img_size=IMG_SIZE)
X_test_prep = preprocess_imgs(set_name=X_test_crop, img_size=IMG_SIZE)
X_val_prep = preprocess_imgs(set_name=X_val_crop, img_size=IMG_SIZE)
```

```
def crop_imgs(set_name, add_pixels_value=0):
    set_new = []
    for img in set_name:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)

        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)

        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0])

        ADD_PIXELS = add_pixels_value
        new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
        set_new.append(new_img)

    return np.array(set_new)
```

```
img_cnt = cv2.drawContours(img.copy(), [c], -1, (0, 255, 255), 4)

img_pnt = cv2.circle(img_cnt.copy(), extLeft, 8, (0, 0, 255), -1)
img_pnt = cv2.circle(img_pnt, extRight, 8, (0, 255, 0), -1)
img_pnt = cv2.circle(img_pnt, extTop, 8, (255, 0, 0), -1)
img_pnt = cv2.circle(img_pnt, extBot, 8, (255, 255, 0), -1)

ADD_PIXELS = 0
new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
```

Figure IV-17: code de Preprocessing de la base de données

Après que les réseaux appliquent le Preprocessing nécessaire à la classification on va maintenant déplacer notre base de données à d'autres dossiers qu'on va les définir avec leurs chemins virtuels :

```
X_train_crop = crop_imgs(set_name=X_train)
X_val_crop = crop_imgs(set_name=X_val)
X_test_crop = crop_imgs(set_name=X_test)
```

```
!mkdir TRAIN_CROP TEST_CROP VAL_CROP TRAIN_CROP/YES TRAIN_CROP/NO TEST_CROP/YES TEST_CROP/NO VAL_CROP/YES VAL_CROP/NO

save_new_images(X_train_crop, y_train, folder_name='TRAIN_CROP/')
save_new_images(X_val_crop, y_val, folder_name='VAL_CROP/')
save_new_images(X_test_crop, y_test, folder_name='TEST_CROP/')
```

Figure IV-18: Définir les nouveaux dossiers et chemins virtuels de la base de données

Maintenant, On va afficher nos nouvelles images classifiées après avoir effectuer le Preprocessing et le traitement nécessaire :

```
[ ]  y = dict()
     y[0] = []
     y[1] = []
     for set_name in (y_train, y_val, y_test):
         y[0].append(np.sum(set_name == 0))
         y[1].append(np.sum(set_name == 1))

     trace0 = go.Bar(
         x=['Train Set', 'Validation Set', 'Test Set'],
         y=y[0],
         name='No',
         marker=dict(color='#33cc33'),
         opacity=0.7
     )
     trace1 = go.Bar(
         x=['Train Set', 'Validation Set', 'Test Set'],
         y=y[1],
         name='Yes',
         marker=dict(color='#ff3300'),
         opacity=0.7
     )
     data = [trace0, trace1]
     layout = go.Layout(
         title='Count of classes in each set',
         xaxis={'title': 'Set'},
         yaxis={'title': 'Count'}
```

Figure IV-19: code de préparation à visualiser nos novelles images

La figure Figure IV-20 montre nos nouvelles images :

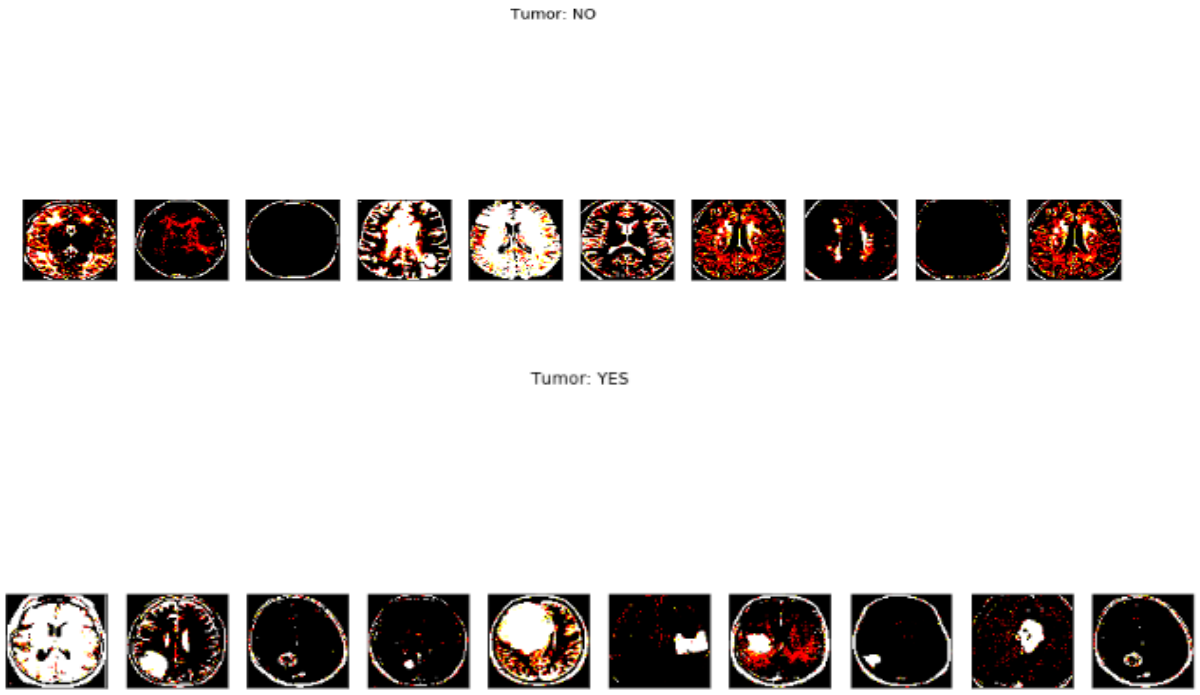


Figure IV-20: Visualisation de nos nouvelles images classifiées

IV.2.7. L'augmentation des données :

On va appliquer la technique d'augmentation des données qui va nous permettre d'ajouter de nouvelles images à notre bases de données à partir des images précédentes après subir une série de transformations aléatoires à chaque image (y compris la rotation aléatoire, le redimensionnement, modification de luminosité, etc.) comme le montre la figure IV-21:

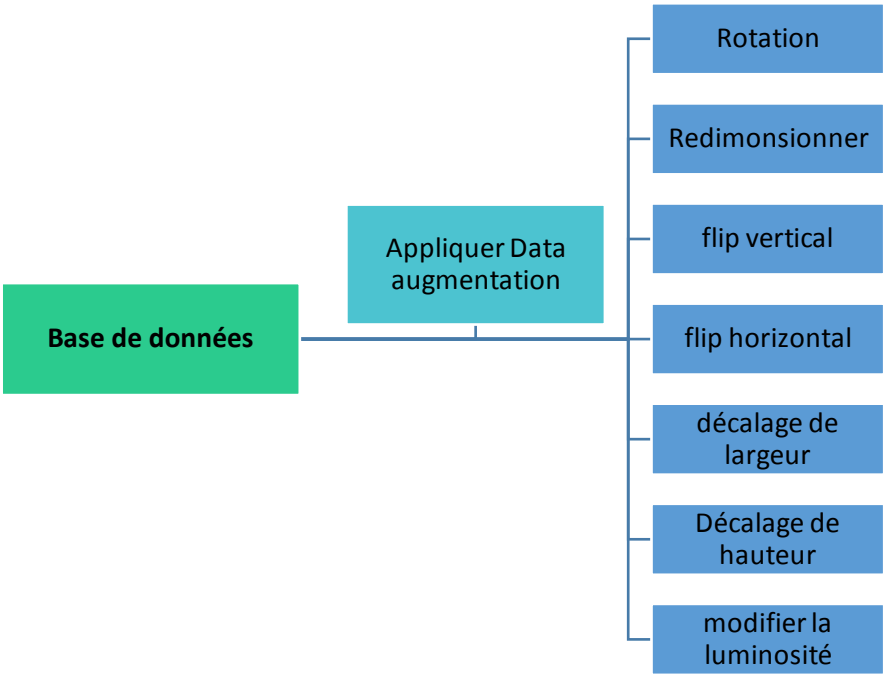


Figure IV-21: Schéma synoptique du principe de la technique d'augmentation des données

Le code dans la figure IV-22 montre comment appliquer l'augmentation de données en apportant des transformation :

```
demo_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.05,
    height_shift_range=0.05,
    rescale=1./255,
    shear_range=0.05,
    brightness_range=[0.1, 1.5],
    horizontal_flip=True,
    vertical_flip=True
)
```

```
plt.imshow(X_train_crop[0])
plt.xticks([])
plt.yticks([])
plt.title('Original Image')
plt.show()

plt.figure(figsize=(15,6))
i = 1
for img in os.listdir('preview/'):
    img = cv2.cv2.imread('preview/' + img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(3,7,i)
    plt.imshow(img)
    plt.xticks([])
    plt.yticks([])
    i += 1
    if i > 3*7:
        break
plt.suptitle('Augemented Images')
plt.show()
```

Figure IV-22: code d'implémentation de l'augmentation de données

Un exemple de resultat d'une image ayant subit l'augmentation des données :

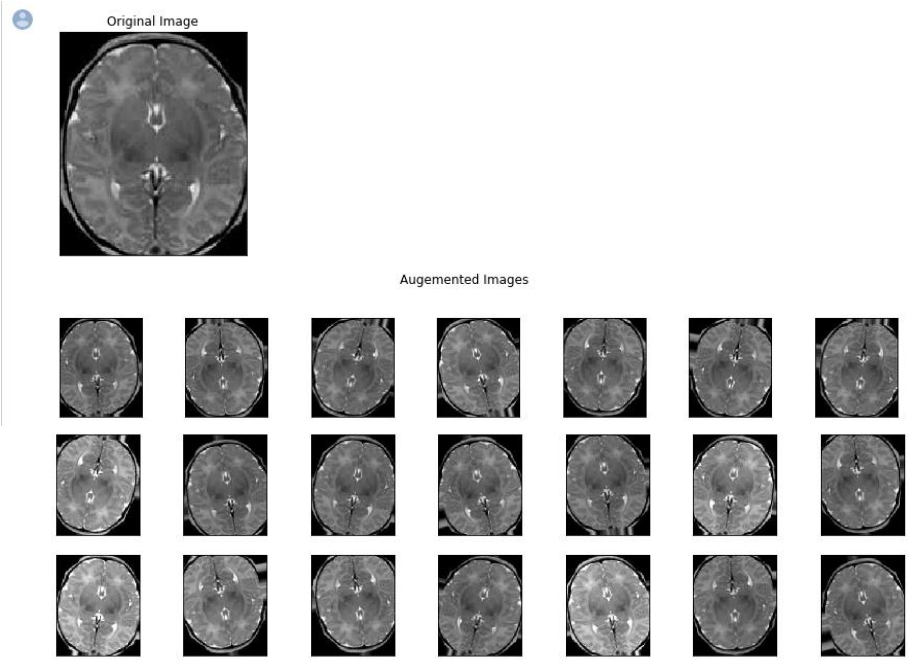


Figure IV-23: Exemple d'une image ayant subit l'augmentation de données

IV.2.8. Appliquer l'augmentation de données pour toute la base de données

En utilisant « ImageDataGenerator » on va augmenter les données d'apprentissage. Elles seront automatiquement générées à l'étape d'entrainement, ce qui va enrichir notre base pour pouvoir booster les performances de nos réseaux et leur permettre de s'entrainer sur plus de données.

```
[ ]  TRAIN_DIR = 'TRAIN_CROP/'
     VAL_DIR = 'VAL_CROP/'

     train_datagen = ImageDataGenerator(
         rotation_range=15,
         width_shift_range=0.1,
         height_shift_range=0.1,
         shear_range=0.1,
         brightness_range=[0.5, 1.5],
         horizontal_flip=True,
         vertical_flip=True,
         preprocessing_function=preprocess_input
     )

     test_datagen = ImageDataGenerator(
         preprocessing_function=preprocess_input
     )
```

Figure IV-24: code d'implémentation de l'augmentation de données pour toute la base

ImageDataGenerator génére les données en utilisant :

* train_datagen.flow_from_directory () * qui prend "le chemin du dossier de données (Data folder path), taille ciblée (target_size), mode de couleurs (color_mode), taille du lot (batch_size), mode de classe (class_mode) " comme paramètres.

```
train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    color_mode='rgb',
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary',
    seed=RANDOM_SEED
)


validation_generator = test_datagen.flow_from_directory(
    VAL_DIR,
    color_mode='rgb',
    target_size=IMG_SIZE,
    batch_size=16,
    class_mode='binary',
    seed=RANDOM_SEED
)
```

Figure IV-25: code de génération de la base de données augmentée

l'augmentation est une étape intermédiaire pour le preprocessing, elle se fait au niveau de image DataGenerator. on n'a pas besoir de visualiser toute la base de données augmentées.

IV.2.9. Appliquer le Transfer Learning

On charge les poids des modèles (en excluant les dernières couche de chacun d'eux car nous voulons que nos propre couches soient les dernières, cela se fait donc avec * include_top = False *), ces modèles seront ajoutés prochainement pour entrainer le réseau.

les poids chargés des premières couches déjà préentrainés et optimisés (donc pas de nécessité d'entrainer toutes les couches, on va entrainer seulement les derniers poids qu'on va ajouter)

Figure IV-26: Chargement des réseaux de neurones qui vont etre utilisés pour le Transfer Learning

Après que les poids des modelés sont chargés (sans inclure leurs dernières couches comme mentionné précédemment), on applique le MaxPooling2D ()

Ensuite on ajoute une couche dense qui est une couche de prédiction qui va nous permettre de prédire entre 2 classes (Classe 1 « Yes » : présence de la tumeur et classe 2 « No » : absence de la tumeur). La fonction d'activation choisie est sigmoïde et on ajoute le classifieur * Softmax * pour la distribution de probabilité.

```
import numpy as np
import pandas as pd
import math
import cv2
import matplotlib.pyplot as plt
import os
import seaborn as sns
import umap
from PIL import Image
from scipy import misc
from os import listdir
from os.path import isfile, join
import numpy as np
from scipy import misc
from random import shuffle
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.utils.np_utils import to_categorical
```

```
from tqdm import tqdm
from itertools import chain
from skimage.io import imread, imshow, imread_collection, concatenate_images
from skimage.transform import resize
from skimage.morphology import label

from keras.models import Model, load_model
from keras.layers import Input
from keras.layers.core import Dropout, Lambda
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import backend as K
import keras
```

Figure IV-27: code de configuration des réseaux de neurones

Étant donné que le classifieur Softmax génère des nombres qui représentent des probabilités des classes, la valeur de chaque nombre se situe entre [0,1]. Les nombres sont nuls ou positifs. L'ensemble du vecteur de sortie est égal à 1. C'est-à-dire lorsque toutes les probabilités sont prises en compte, c'est 100%. On va le configurer dans le code de la figure IV-28 :

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import Model
import matplotlib.pyplot as plt
from numpy import expand_dims

model = VGG16()
ixs = [2, 5, 9, 13, 17]
outputs = [model.layers[i].output for i in ixs]
model = Model(inputs=model.inputs, outputs=outputs)
img = expand_dims(img, axis=0)

img = preprocess_input(img)

feature_maps = model.predict(img)

square = 8
for fmap in feature_maps:
ix = 1
for _ in range(square):
  plt.figure(figsize=(64,64))
  for _ in range(square):

    ax = pyplot.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
```

Figure IV-28: code de configuration du classifieur Softmax

On applique le Preprocessing du VGG 16 et visualiser les caractéristique (features) des premières couches avec les paramètres entrainés précédemment qu'on va les geler comme suit

```
f = plt.figure(figsize=(16,16))
model = VGG16()
model = Model(inputs=model.inputs, outputs=model.layers[1].output)
model.summary()
img = img_to_array(X_val_prep[43])
img = expand_dims(img, axis=0)
img = preprocess_input(img)
feature_maps = model.predict(img)
square = 8
ix = 1
for _ in range(square):
  for _ in range(square):
    ax = pyplot.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    pyplot.imshow(feature_maps[0, :, :, ix-1], cmap='viridis')
    ix += 1
pyplot.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
=================================================================
Total params: 1,792
Trainable params: 1,792
Non-trainable params: 0
_________________________________________________________________
```

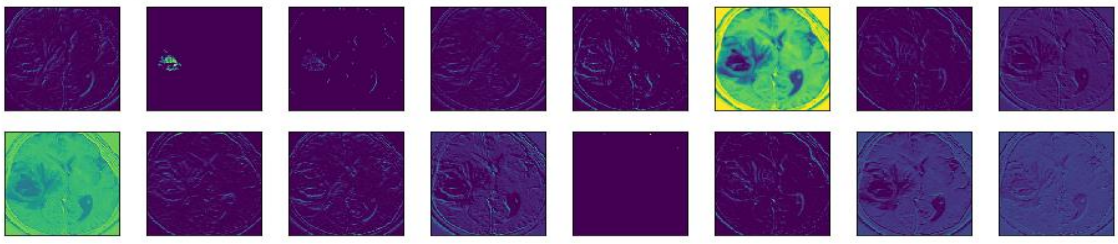


Figure IV-29 : Visualisation des caractéristiques des premières couches et paramètres du réseau

IV.2.10. L'entrainement des réseaux de classification

Pour Procéder à l'entrainement on choisit la configuration de la fonction d'activation sigmoïde et l'optimiseur Adam.

On Compile le modèle en utilisant * model.compile () * qui prend l'optimiseur (Optimizer), la perte (loss) et la métrique (metrics) comme paramètres.

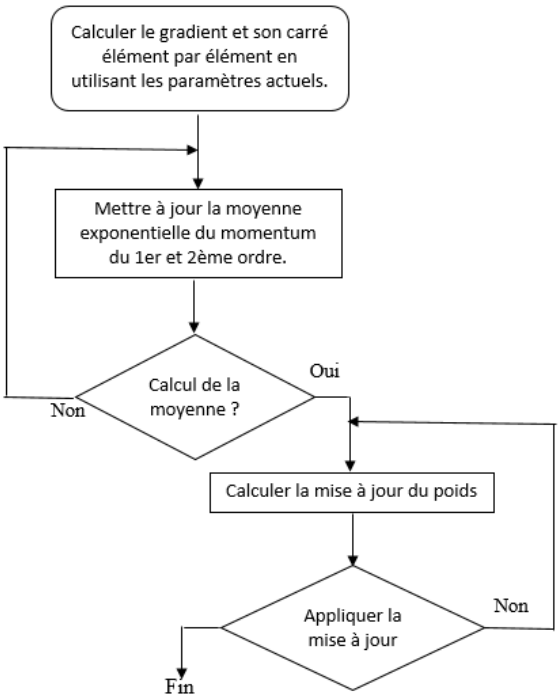


Figure IV-30: Organigramme de l'optimiseur ADAM

L'optimiseur Adam a une étape de correction de biais. Il prend 3 hyper paramètres: le taux d'apprentissage (the learning rate), le taux de décroissance (the decay rate) du moment de 1er ordre et le taux de décroissance du moment de 2e ordre. La configuration pour choisir ces paramètres avec Keras est comme suit :

lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0

```
NUM_CLASSES = 1
vgg16 = Sequential()
vgg16.add(vgg)
vgg16.add(layers.Dropout(0.3))
vgg16.add(layers.Flatten())
vgg16.add(layers.Dropout(0.5))
vgg16.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

vgg16.layers[0].trainable = False

vgg16.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Adam(lr=0.0003, beta_1=0.9, beta_2=0.999,decay=0.0,
                                                                           amsgrad=False), metrics=["accuracy"])
vgg16.summary()
```

Figure IV-31: code de configuration de la fonction d'activation et l'optimiseur ADAM

On visualise le résumé du modèle qu'on a appelé dans le code précédent (**Model Summary**) :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout_2 (Dropout)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25089
=================================================================
Total params: 14,739,777
Trainable params: 25,089
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure IV-32: Visualisation du résumé du modèle

On remarque de la figure IV-32 qu'on va entrainer seulement 25089 paramètres et les autres (14714688 paramètres) c'est ceux qu'on a exclu avec l'instruction *Include_Top= False*, ces derniers seront utilisés directement avec leurs poids précédents et ne vont pas être modifiés durant l'entrainement.

Ensuite, On exécute avec * model.fit_generator () *

On choisit les Epoqus = 120, le pas dans les Epoqus = 50 et on lance l'entrainement :

```
[ ] import time

    start = time.time()

    vgg16_history = vgg16.fit_generator(
        train_generator,
        steps_per_epoch=50,
        epochs=120,
        validation_data=validation_generator,
        validation_steps=30,
    )


    end = time.time()
    print(end - start)
50/50 [==============================] - 20s 395ms/step - loss: 0.2491 - acc: 0.9781 - val_loss: 0.9576 - val_acc: 0.9375
Epoch 111/120
50/50 [==============================] - 20s 396ms/step - loss: 0.1944 - acc: 0.9799 - val_loss: 1.1748 - val_acc: 0.9215
Epoch 112/120
50/50 [==============================] - 19s 387ms/step - loss: 0.2295 - acc: 0.9793 - val_loss: 1.1036 - val_acc: 0.9185
Epoch 113/120
50/50 [==============================] - 20s 397ms/step - loss: 0.2246 - acc: 0.9768 - val_loss: 1.0404 - val_acc: 0.9215
Epoch 114/120
50/50 [==============================] - 20s 396ms/step - loss: 0.2056 - acc: 0.9787 - val_loss: 0.8814 - val_acc: 0.9185
Epoch 115/120
50/50 [==============================] - 20s 401ms/step - loss: 0.8395 - acc: 0.9351 - val_loss: 0.7018 - val_acc: 0.9424
Epoch 116/120
50/50 [==============================] - 20s 396ms/step - loss: 0.3106 - acc: 0.9724 - val_loss: 1.2547 - val_acc: 0.9185
Epoch 117/120
50/50 [==============================] - 20s 399ms/step - loss: 0.2502 - acc: 0.9749 - val_loss: 1.3421 - val_acc: 0.9005
Epoch 118/120
50/50 [==============================] - 20s 401ms/step - loss: 0.1547 - acc: 0.9793 - val_loss: 1.3073 - val_acc: 0.9185
Epoch 119/120
50/50 [==============================] - 19s 389ms/step - loss: 0.2412 - acc: 0.9768 - val_loss: 1.2172 - val_acc: 0.9241
Epoch 120/120
50/50 [==============================] - 20s 395ms/step - loss: 0.3027 - acc: 0.9724 - val_loss: 1.3502 - val_acc: 0.9158
```

Figure IV-33: Visualisation de l'entrainement

Maintenant, on va visualiser quelques caractéristiques (features) complexes des dernières couches du réseau qu'on ajouté pour les voir :

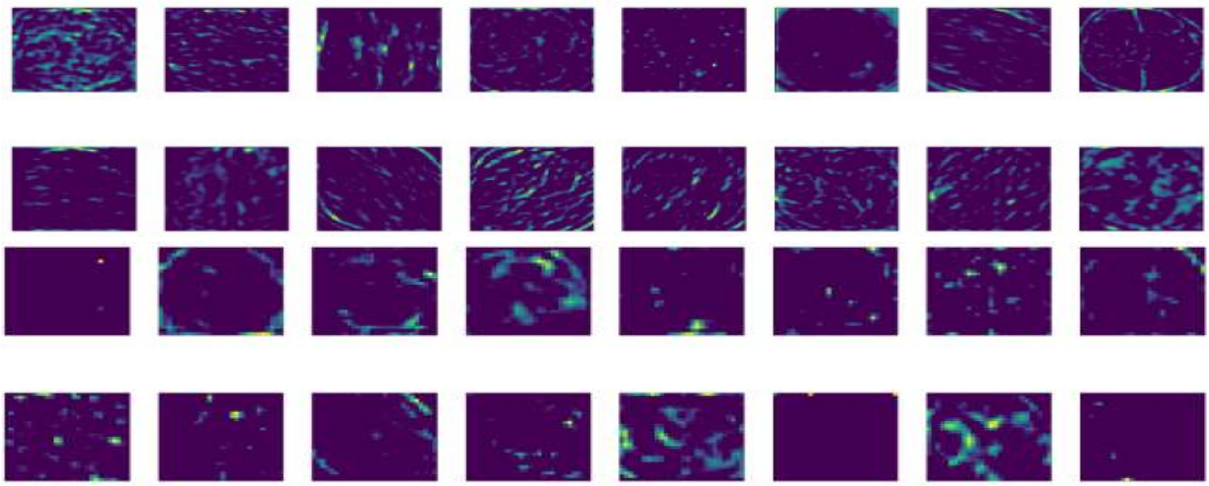


Figure IV-34: les caractéristiques (features) complexes des dernières couches ajoutées

Après qu'on a implémenté le réseau **VGG16** modifié, On a suivi le meme principe et les meme étapes pour les réseaux **Inception V3** et **Resnet50**. on va maintenant tracer leurs courbes et comparer les résultats :

```
[ ]  def ModelGraphTrainngSummaryAcc(history,N,model_name):
        # set the matplotlib backend so figures can be saved in the background
        # plot the training loss and accuracy
        import sys
        import matplotlib
        print("Generating plots...")
        sys.stdout.flush()
        matplotlib.use("Agg")
        matplotlib.pyplot.style.use("ggplot")
        matplotlib.pyplot.figure()
        #matplotlib.pyplot.plot(np.arange(0, N), history.history["loss"], label="train_loss")
        #matplotlib.pyplot.plot(np.arange(0, N), history.history["val_loss"], label="val_loss")
        matplotlib.pyplot.plot(np.arange(0, N), history.history["acc"], label="train_acc")
        matplotlib.pyplot.plot(np.arange(0, N), history.history["val_acc"], label="val_acc")
        matplotlib.pyplot.title("Training Loss and Accuracy on Brain Tumor Classification")
        matplotlib.pyplot.xlabel("Epoch #")
        matplotlib.pyplot.ylabel("Accuracy of "+ model_name)
        matplotlib.pyplot.legend(loc="lower left")
        matplotlib.pyplot.savefig("plot.png")
```

```
[ ]  for x_model in [{'name':'VGG-16','history':history_1,'model':vgg16},
                     {'name':'Inception_v3','history':history_2,'model':inception_v3},
                     {'name':'Resnet','history':history_3,'model':resnet50}]:
         ModelGraphTrainngSummary(x_model['history'],120,x_model['name'])
         ModelGraphTrainngSummaryAcc(x_model['history'],120,x_model['name'])

         # validate on val set
         predictions = x_model['model'].predict(X_val_prep)
         predictions = [1 if x>0.5 else 0 for x in predictions]

         accuracy = accuracy_score(y_val, predictions)
         print('Val Accuracy = %.2f' % accuracy)

         confusion_mtx = confusion_matrix(y_val, predictions)
         cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
```

Figure IV-35: code des tracés des courbes d'entrainement

IV.3. Résultats et discussion

Figure IV-36: La courbe de perte (loss) et la courbe de précision (accuracy) du réseau VGG16

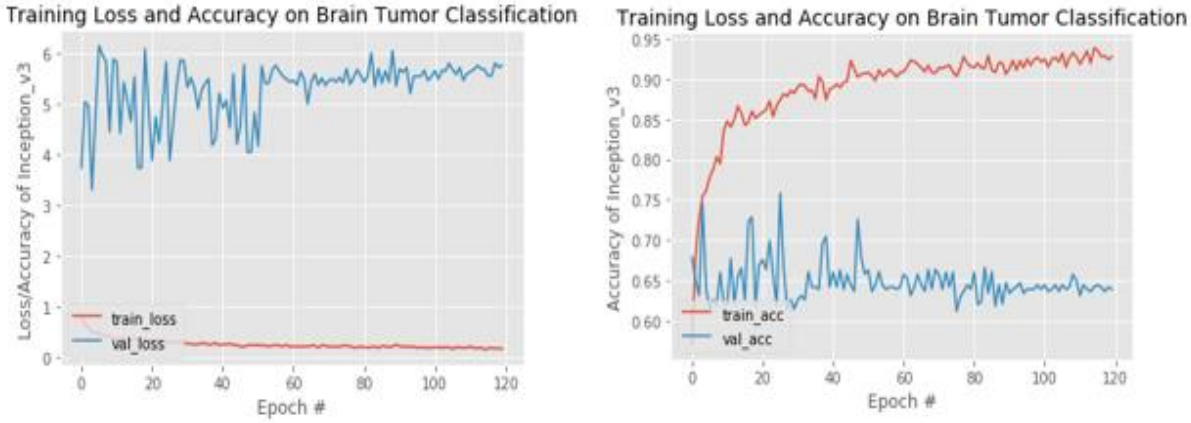


Figure IV-37: La courbe de perte (loss) et la courbe de précision (accuracy) du réseau Inception V3

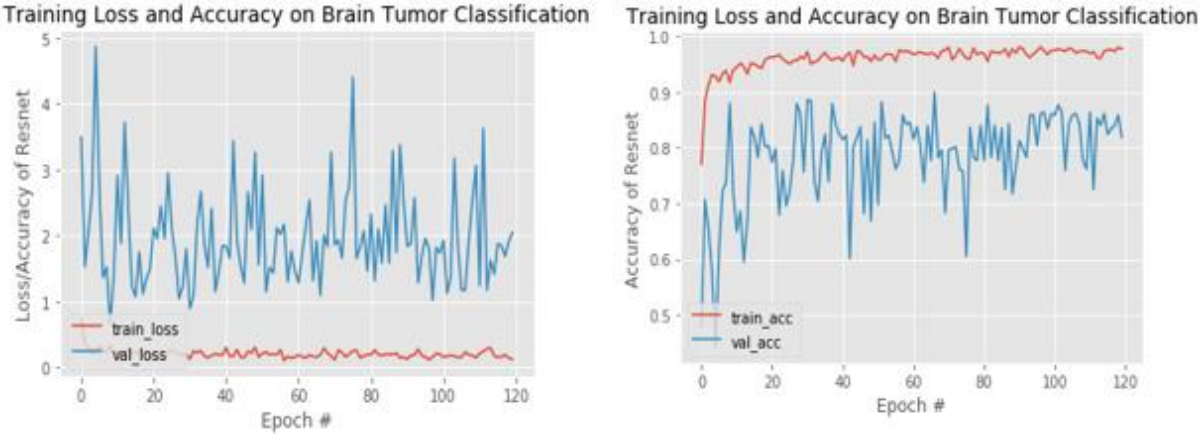


Figure IV-38: La courbe de perte (loss) et la courbe de précision (accuracy) du réseau Resnet 50

D'après les résultats obtenus on peut évidemment constater que la performance diffère d'un réseau à l'autre, la meilleure est celle du VGG-16, son tracé de la perte d'entraînement (Train Loss) diminue jusqu'à un point de stabilité et celui de la perte de validation (Validation Loss) diminue aussi jusqu'à un certain point en présentant un petit écart avec la perte d'entraînement, Cet écart est appelé « l'écart de généralisation ».

Pour les réseaux Resnet 50 et Inception V3, l'écart est un peu grand par rapport à celui du VGG-16, les deux modèles ont bien appris les données d'entrainement (Train Data set) mais n'ont pas eu de bons résultats avec les nouvelles données de validation (Validation Data set).

L'explication peut être que les deux modèles ont appris parfaitement l'ensemble de données d'entrainement (y compris le bruit statistique ou les fluctuations aléatoires dans l'ensemble de données d'apprentissage) au point de ne pas pouvoir se généraliser aux nouvelles données de validation, c'est ce qui est peut être appelé «Overfitting» qui a plusieurs raisons possibles mais on pense que dans notre cas il est dû au fait que l'ensemble de données utilisé pour réaliser notre projet est petit (aux environs de 5313 images après l'augmentation de données) par rapport à la quantité de données généralement utilisée dans d'autres applications de Deep Learning.

IV.4. Implémentation du masque R-CNN pour la détection :

Pour pouvoir implémenter notre réseau qui va générer l'instance du masque de détection des régions exactes des tumeurs on va suivre les étapes montrées dans la figure IV-39:

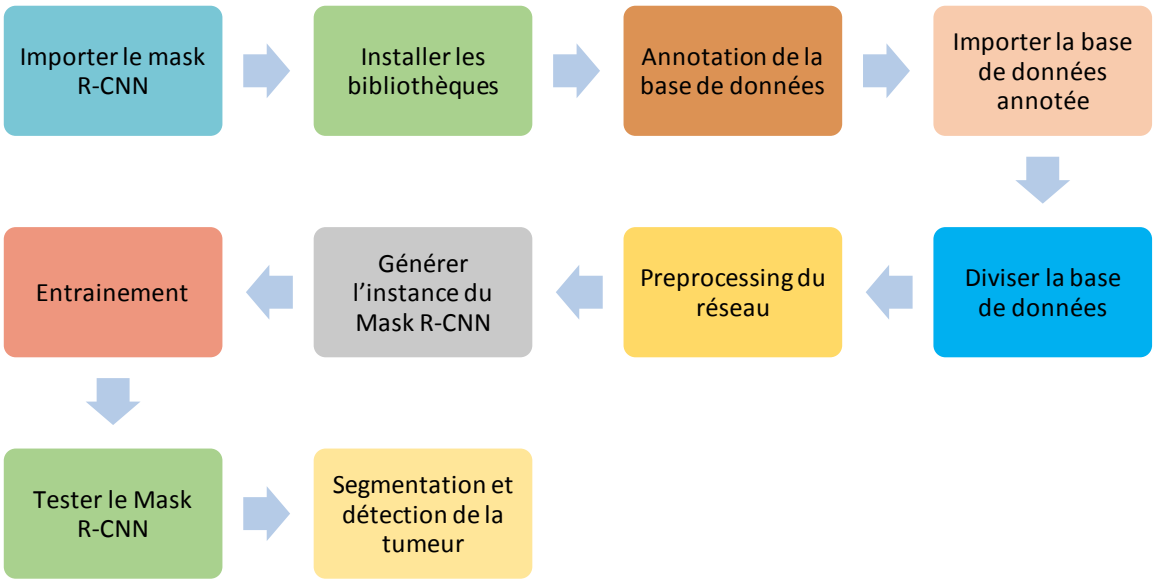


Figure IV-39:Les étapes d'implementation du Mask R-CNN

IV.4.1. Préparer l'environnement

D'abord on prépare notre environnement. On va utiliser le masque R-CNN :

```
from IPython.display import clear_output
os.chdir("/content/drive/My Drive/Colab Notebooks/Mask R-CNN")
os.chdir("/content/drive/My Drive/brain-tumor-master")
!pip install pycocotools
clear_output()
```

Figure IV-40: code d'importation du masque R-CNN

IV.4.2. Importation des bibliothèques

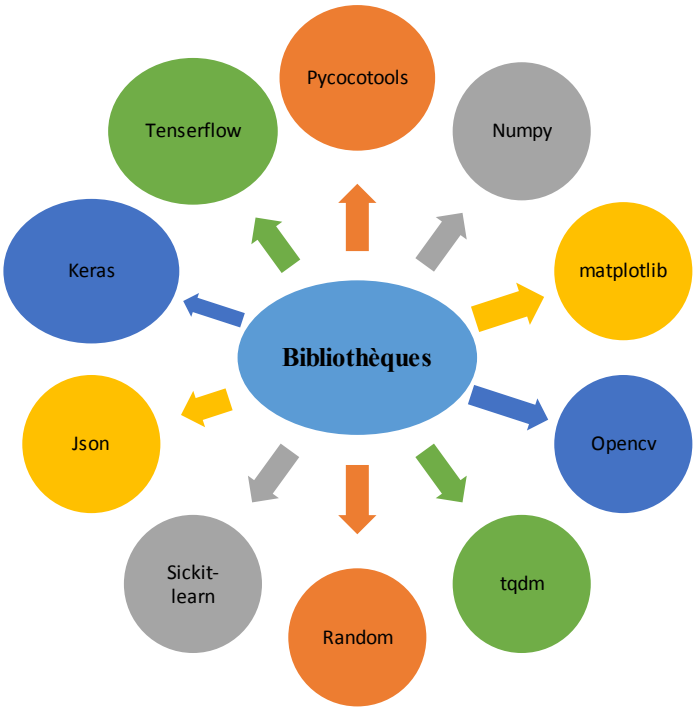


Figure IV-41: les bibliothèques utilisées

```
import os
import sys
from tqdm import tqdm
import cv2
import numpy as np
import json
import skimage.draw
import matplotlib
import matplotlib.pyplot as plt
import random
ROOT_DIR = os.path.abspath('Mask_RCNN/')
sys.path.append(ROOT_DIR)
from mrcnn.config import Config
from mrcnn import utils
from mrcnn.model import log
import mrcnn.model as modellib
from mrcnn import visualize
sys.path.append(os.path.join(ROOT_DIR, 'samples/coco/'))
import coco

plt.rcParams['figure.facecolor'] = 'white'

clear_output()
```

Figure IV- 42: code d'importation des bibliothèques utilisées

IV.4.3. Annotation et importation de la base de données

Avant d'importer la base de données utilisée pour l'entrainement sur la detection on doit faire les annotation avec la plateforme VIA. Son fonctionnement est expliqué dans la figure IV- 43:

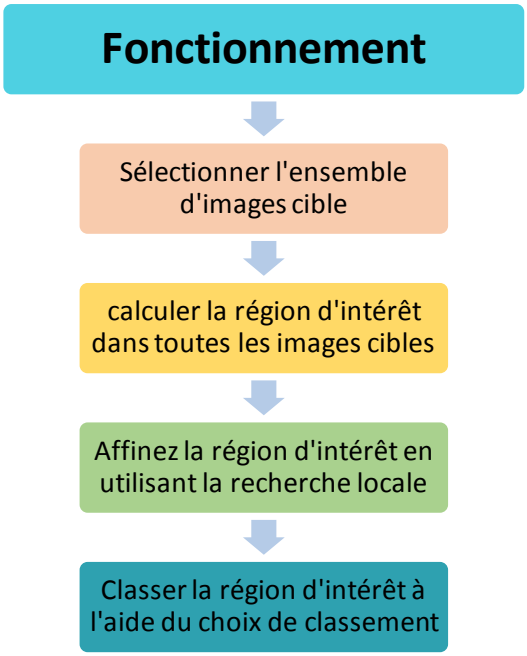


Figure IV- 43: le Fonctionnement de la Plateforme d'annotation VIA

Les figures IV-44, IV- 46, IV- 48 montrent quelques exemples d'annotations réalisées avec la platefrome VIA en utilisant le format polygone pour annoter la zone de la tumeur :

(a) (b)

Figure IV-44: exemple d'annotations de la zone de la tumeur avec la plateforme VIA
(a) mage sans annotation, (b) image avec annotation

Pour chaque image qu'on annote on peut avoir un fichier CSV contenant les donnés d'annotations comme dans la figure IV-45 :

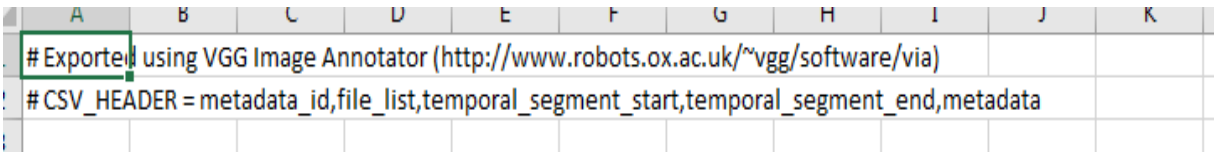


Figure IV-45: exemple du contenu du fichier CSV des annotations

A chaque fois qu'on ajoute les annotations d'une autre image le contenu du fichier CSV change et ajoute les nouvelles annotations aux valeurs précédentes.

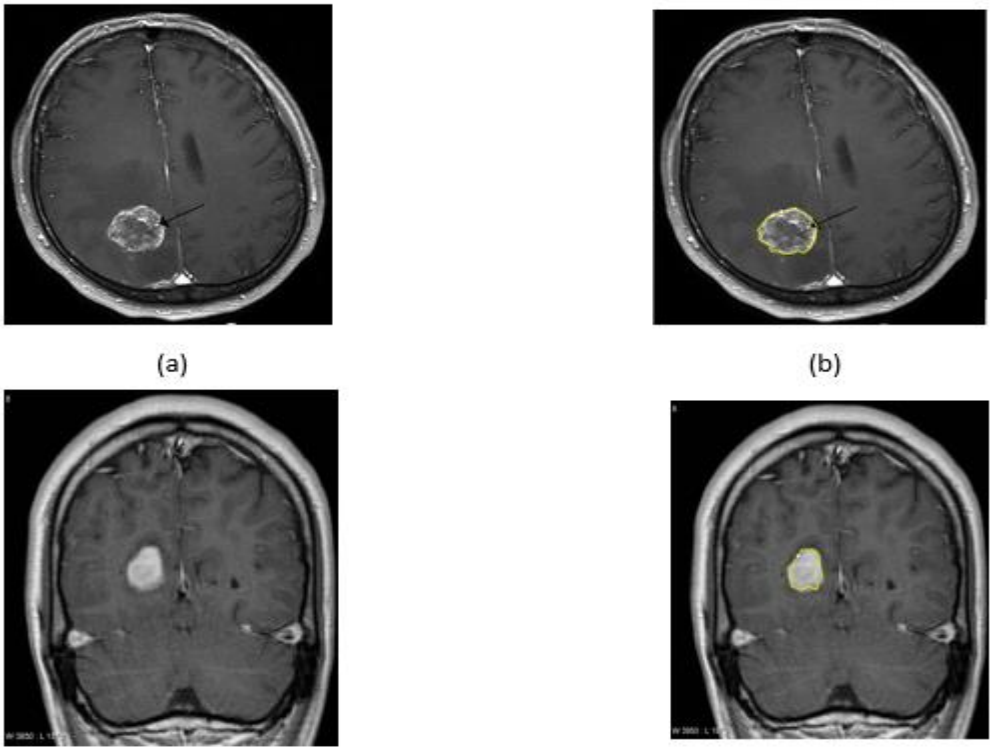
(a) (b)

Figure IV- 46: Deuxième exemple d'annotations (a) image sans annotation (b) image avec annotation

```
# Exported using VGG Image Annotator (http://www.robots.ox.ac.uk/~vgg/software/via)
# Notes:
# - spatial_coordinates of [2,10,20,50,80] denotes a rectangle (shape_id=2) of size 50x80 placed at (10,20)
# - temporal coordinate of [1.349,2.741] denotes a temporal segment from time 1.346 sec. to 2.741 sec.
# - temporal coordinate of [4.633] denotes a still video frame at 4.633 sec.
# - metadata of {""1"":""3""} indicates attribute with id "1" set to an attribute option with id "3"
# SHAPE_ID = {"POINT":1,"RECTANGLE":2,"CIRCLE":3,"ELLIPSE":4,"LINE":5,"POLYLINE":6,"POLYGON":7,"EXTREME_RECTANGLE":8,"EXTREME_CIRCLE":9}
# FLAG_ID = {"RESERVED_FOR_FUTURE":1}
# ATTRIBUTE = {"1":{"aname":"tumor","anchor_id":"FILE1_Z0_XY1","type":5,"desc":"","options":{},"default_option_id":""}}
# CSV_HEADER = metadata_id,file_list,flags,temporal_coordinates,spatial_coordinates,metadata
1_5LWmXXl5,"[""Y1.jpg""]",0,"[]","[7,19.54,123.75,34.865,136.011,46.742,135.627,58.619,139.459,70.112,137.543,76.243,132.562,80.84,127.965,86.587,116.471,86.97,107.659,86.204,96.548,86.204,96.548,81.606,86.587,77.
```

Figure IV- 47: exemple de mise à jour du contenu du fichier CSV des annotations

On continue de la meme façon pour les autres images, On peut annoter avec précision meme si la zone de la tumeur est ptite avec l'outil flexible d'annotation format polygone

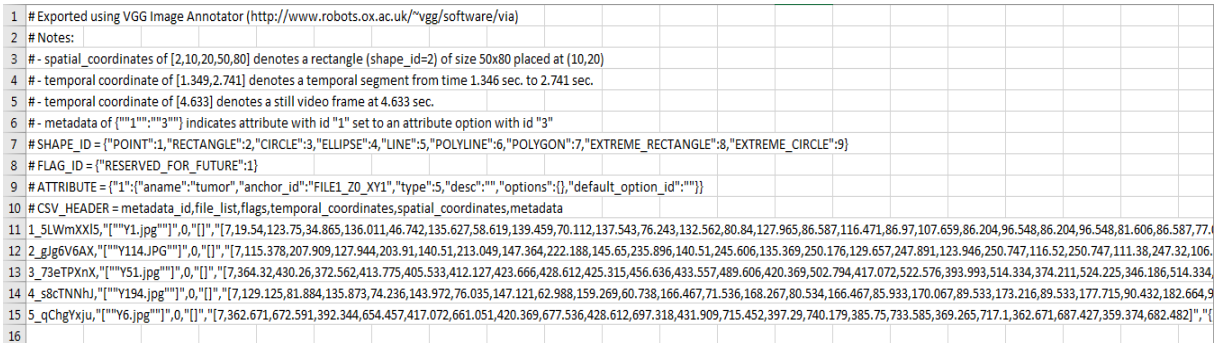


(a) (b)

(a) (b)

Figure IV-48: Annotation des images de la base de données (a)image sans annotation (b) image avec annotation

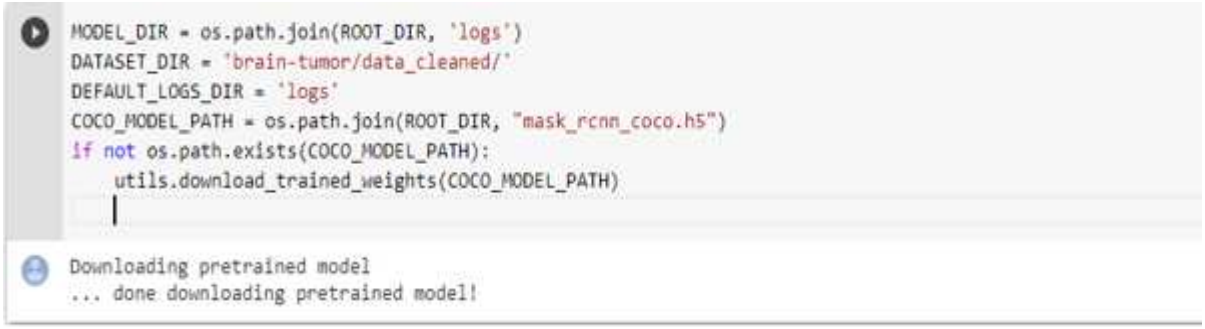
Après la mise à jour de toutes les annotations ajoutées le fichier CSV devient comme suit :



```
# Exported using VGG Image Annotator (http://www.robots.ox.ac.uk/~vgg/software/via)
# Notes:
# - spatial_coordinates of [2,10,20,50,80] denotes a rectangle (shape_id=2) of size 50x80 placed at (10,20)
# - temporal coordinate of [1.349,2.741] denotes a temporal segment from time 1.346 sec. to 2.741 sec.
# - temporal coordinate of [4.633] denotes a still video frame at 4.633 sec.
# - metadata of {""1"":""3""} indicates attribute with id "1" set to an attribute option with id "3"
# SHAPE_ID = {"POINT":1,"RECTANGLE":2,"CIRCLE":3,"ELLIPSE":4,"LINE":5,"POLYLINE":6,"POLYGON":7,"EXTREME_RECTANGLE":8,"EXTREME_CIRCLE":9}
# FLAG_ID = {"RESERVED_FOR_FUTURE":1}
# ATTRIBUTE = {"1":{"aname":"tumor","anchor_id":"FILE1_Z0_XY1","type":5,"desc":"","options":{},"default_option_id":""}}
# CSV_HEADER = metadata_id,file_list,flags,temporal_coordinates,spatial_coordinates,metadata
1_5LWmXXl5,"[""Y1.jpg""]",0,"[]","[7,19.54,123.75,34.865,136.011,46.742,135.627,58.619,139.459,70.112,137.543,76.243,132.562,80.84,127.965,86.587,116.471,86.97,107.659,86.204,96.548,86.204,96.548,81.606,86.587,77.
2_gJg6V6AX,"[""Y114.JPG""]",0,"[]","[7,115.378,207.909,127.944,203.91,140.51,213.049,147.364,222.188,145.65,235.896,140.51,245.606,135.369,250.176,129.657,247.891,123.946,250.747,116.52,250.747,111.38,247.32,106.
3_73eTPXnX,"[""Y51.jpg""]",0,"[]","[7,364.32,430.26,372.562,413.775,405.533,412.127,423.666,428.612,425.315,456.636,433.557,489.606,420.369,502.794,417.072,522.576,393.993,514.334,374.211,524.225,346.186,514.334,
4_s8cTNNhJ,"[""Y194.jpg""]",0,"[]","[7,129.125,81.884,135.873,74.236,143.972,76.035,147.121,62.988,159.269,60.738,166.467,71.536,168.267,80.534,166.467,85.933,170.067,89.533,173.216,89.533,177.715,90.432,182.664,9
5_qChgYxju,"[""Y6.jpg""]",0,"[]","[7,362.671,672.591,392.344,654.457,417.072,661.051,420.369,677.536,428.612,697.318,431.909,715.452,397.29,740.179,385.75,733.585,369.265,717.1,362.671,687.427,359.374,682.482]","{
```

Figure IV-49: contenu du fichier CSV après l'ajout de toutes les annotations

Après l'annotation de la base de données on importe maintenant les poids du masque R-CNN et on crée un chemin comme montré :



```
MODEL_DIR = os.path.join(ROOT_DIR, 'logs')
DATASET_DIR = 'brain-tumor/data_cleaned/'
DEFAULT_LOGS_DIR = 'logs'
COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")
if not os.path.exists(COCO_MODEL_PATH):
    utils.download_trained_weights(COCO_MODEL_PATH)
```

```
Downloading pretrained model
... done downloading pretrained model!
```

Figure IV-50 : code d'importation des poids du masque et création du chemin de la base

On importer les annotations réalisées précédemment apres la convertion de leur fichier de format CSV vers le format de fichier JSON en utilisant la plateforme « Any conv »

Figure IV- 51: la plateforme de conversion « Any Conv »

Le contenu du fichier .JSON des annotations peut être ouvert avec un editeur de texte (exemple : Windows Notepad) pour visualiser et verifier son contenu :

{"0.jpg25463":{"filename":"0.jpg","size":25463,"regions":[{"shap
egions":[{"shape_attributes":{"name":"polygon","all_points_x":[1
':23824,"regions":[{"shape_attributes":{"name":"polygon","all_po
,135,127,128,124,120]},"region_attributes":{}}],"file_attributes
9,165,172,182,183,182,173,158,149,135,125,116],"all_points_y":[2
es":{"name":"polygon","all_points_x":[6,17,28,41,55,58,56,49,38,
,"region_attributes":{}},{"shape_attributes":{"name":"polygon","
}}],"file_attributes":{}},"38.jpg25503":{"filename":"38.jpg","si
pe_attributes":{"name":"polygon","all_points_x":[100,114,115,112
}},{"shape_attributes":{"name":"polygon","all_points_x":[104,120
78,182,188,191,197,198,196,196,190,180,172,168,162,160]},"region
"filename":"63.jpg","size":31651,"regions":[{"shape_attributes":
n_attributes":{}}],"file_attributes":{}},"66.jpg25325":{"filenam
pg","size":20567,"regions":[{"shape_attributes":{"name":"polygon
es":{}},"77.jpg23096":{"filename":"77.jpg","size":23096,"regions
ributes":{}},"81.jpg23110":{"filename":"81.jpg","size":23110,"re
y":[148,151,154,161,167,172,180,188,194,204,213,210,209,207,192,
ibutes":{}}],"file_attributes":{}},"88.jpg21995":{"filename":"88
ttributes":{}},"92.jpg24780":{"filename":"92.jpg","size":24780,"
es":{"name":"polygon","all_points_x":[97,82,76,63,58,62,68,95,11
,"file_attributes":{}},"113.jpg20317":{"filename":"113.jpg","siz
"regions":[{"shape_attributes":{"name":"polygon","all_points_x":
187,180,169,165]},"region_attributes":{}}],"file_attributes":{}}
':[19,59,66,54,49,46,38,18,4,1,1,12]},"region_attributes":{}}],"
103,93],"all_points_y":[163,173,183,192,195,187,174,166,161,158]
},"147.jpg23629":{"filename":"147.jpg","size":23629,"regions":[{
],"file_attributes":{}},"150.jpg20171":{"filename":"150.jpg","si
":"polygon","all_points_x":[188,184,187,193,202,213,220,223,217,
ibutes":{}},"163.jpg24946":{"filename":"163.jpg","size":24946,"r

Figure IV- 52 : contenu du fichier json des annotations

La prochaine étape c'est de diviser la base de données et importer le fichier JSON des annotations :

```python
class BrainScanDataset(utils.Dataset):

    def load_brain_scan(self, dataset_dir, subset):
        """dataset_dir: Root directory of the dataset.
        subset: Subset to load: train or val
        """
        self.add_class("tumor", 1, "tumor")
        assert subset in ["train", "val", 'test']
        dataset_dir = os.path.join(dataset_dir, subset)

        annotations = json.load(open(os.path.join(DATASET_DIR, subset, 'annotations_'+subset+'.json')))
        annotations = list(annotations.values())
        annotations = [a for a in annotations if a['regions']]
        for a in annotations:
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in a['regions']]
            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
```

Figure IV-53: code de division de la base de données et importation du fichier JSON

IV.4.4. Géneration de l'instance du masque R-CNN

Maintenant, on va générer l'instance du masque R-CNN, sa configuration et faite avec le code dans la figure IV-54 :

```
        height, width = image.shape[:2]

        self.add_image(
            "tumor",
            image_id=a['filename'],
            path=image_path,
            width=width,
            height=height,
            polygons=polygons
        )

def load_mask(self, image_id):
    """Generate instance masks for an image.
   Returns:
    masks: A bool array of shape [height, width, instance count] with
        one mask per instance.
    class_ids: a 1D array of class IDs of the instance masks.
    """
    image_info = self.image_info[image_id]
    if image_info["source"] != "tumor":
        return super(self.__class__, self).load_mask(image_id)
    info = self.image_info[image_id]
    mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                    dtype=np.uint8)
```

```
        for i, p in enumerate(info["polygons"]):
            rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
            mask[rr, cc, i] = 1
        return mask.astype(np.bool), np.ones([mask.shape[-1]], dtype=np.int32)

    def image_reference(self, image_id):
        """Return the path of the image."""
        info = self.image_info[image_id]
        if info["source"] == "tumor":
            return info["path"]
        else:
            super(self.__class__, self).image_reference(image_id)
```

```
model = modellib.MaskRCNN(
    mode='training',
    config=config,
    model_dir=DEFAULT_LOGS_DIR
)

model.load_weights(
    COCO_MODEL_PATH,
    by_name=True,
    exclude=["mrcnn_class_logits", "mrcnn_bbox_fc", "mrcnn_bbox", "mrcnn_mask"]
)
```

Figure IV-54: code de génération d'instance du masque R-CNN

IV.4.5. L'entrainement

Puisqu'on va entrainer les dernières couches seulement (Heads) au niveau du deuxième étage « RPN » du masque R-CNN, on n'aura pas besoin d'un grand nombre d'époques. On a essayé avec 30, 20 et 15 époques et presque les mêmes résultats ont été obtenus, donc on a opté pour 15 époques.

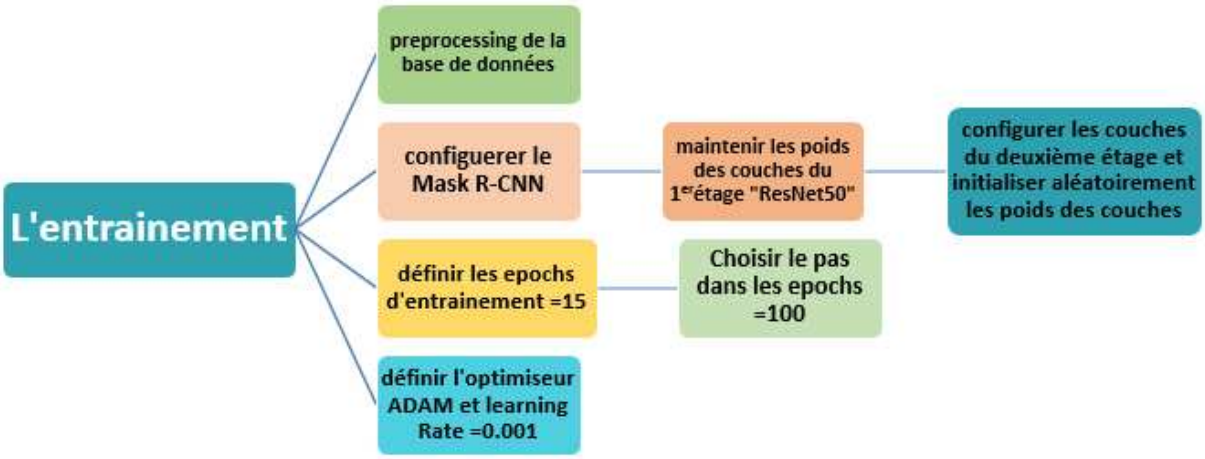


Figure IV-55: les configurations d'entrainement du masque R-CNN

La bibliothèque de Deep Learning Keras nous permet de configurer le taux d'apprentissage, Ce dernier contrôle la rapidité avec laquelle le modèle est adapté au problème. Les valeurs plus petites nécessitent plus d'époques de formation étant donné les petits changements apportés aux poids à chaque mise à jour, tandis que les taux d'apprentissage plus élevés entraînent des changements rapides et nécessitent moins d'epoques de formation. Le taux d'apprentissage est peut-être l'hyper paramètre le plus important qu'on doit le choisir avec conscience.

```
dataset_train = BrainScanDataset()
dataset_train.load_brain_scan(DATASET_DIR, 'train')
dataset_train.prepare()

dataset_val = BrainScanDataset()
dataset_val.load_brain_scan(DATASET_DIR, 'val')
dataset_val.prepare()

dataset_test = BrainScanDataset()
dataset_test.load_brain_scan(DATASET_DIR, 'test')
dataset_test.prepare()

print("Training network heads")
model.train(
    dataset_train, dataset_val,
    learning_rate=config.LEARNING_RATE,
    epochs=15,
    layers='heads'
)
```

Figure IV-56: code de configuration d'entrainement

```
Training network heads

Starting at epoch 0. LR=0.001

Checkpoint Path: logs/tumor_detector20190901T0743/mask_rcnn_tumor_detector_{epoch:04d}.h5
Selecting layers to train
fpn_c5p5               (Conv2D)
fpn_c4p4               (Conv2D)
fpn_c3p3               (Conv2D)
fpn_c2p2               (Conv2D)
fpn_p5                 (Conv2D)
fpn_p2                 (Conv2D)
fpn_p3                 (Conv2D)
fpn_p4                 (Conv2D)
In model:  rpn_model
    rpn_conv_shared        (Conv2D)
    rpn_class_raw          (Conv2D)
    rpn_bbox_pred          (Conv2D)
mrcnn_mask_conv1       (TimeDistributed)
mrcnn_mask_bn1         (TimeDistributed)
mrcnn_mask_conv2       (TimeDistributed)
mrcnn_mask_bn2         (TimeDistributed)
mrcnn_class_conv1      (TimeDistributed)
mrcnn_class_bn1        (TimeDistributed)
mrcnn_mask_conv3       (TimeDistributed)
mrcnn_class_conv2      (TimeDistributed)
mrcnn_class_bn2        (TimeDistributed)
mrcnn_mask_conv4       (TimeDistributed)
mrcnn_mask_bn4         (TimeDistributed)
mrcnn_bbox_fc          (TimeDistributed)
mrcnn_mask_deconv      (TimeDistributed)
mrcnn_class_logits     (TimeDistributed)
mrcnn_mask             (TimeDistributed)
/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/gradients_util.py:93: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown sh
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/gradients_util.py:93: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown sh
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/gradients_util.py:93: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown sh
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/opt/conda/lib/python3.6/site-packages/keras/engine/training_generator.py:47: UserWarning: Using a generator with `use_multiprocessing=True` and multiple worke
  UserWarning('Using a generator with `use_multiprocessing=True`'
Epoch 1/15
100/100 [==============================] - 96s 964ms/step - loss: 1.9466 - rpn_class_loss: 0.0299 - rpn_bbox_loss: 0.7874 - mrcnn_class_loss: 0.0781 - mrcnn_bb
Epoch 2/15
100/100 [==============================] - 51s 506ms/step - loss: 1.2038 - rpn_class_loss: 0.0234 - rpn_bbox_loss: 0.3541 - mrcnn_class_loss: 0.0550 - mrcnn_bb
Epoch 3/15
100/100 [==============================] - 50s 500ms/step - loss: 0.8259 - rpn_class_loss: 0.0148 - rpn_bbox_loss: 0.2216 - mrcnn_class_loss: 0.0446 - mrcnn_bb
Epoch 4/15
100/100 [==============================] - 50s 496ms/step - loss: 0.8250 - rpn_class_loss: 0.0162 - rpn_bbox_loss: 0.2491 - mrcnn_class_loss: 0.0416 - mrcnn_bb
Epoch 5/15
100/100 [==============================] - 48s 482ms/step - loss: 0.6779 - rpn_class_loss: 0.0131 - rpn_bbox_loss: 0.1800 - mrcnn_class_loss: 0.0477 - mrcnn_bb
Epoch 6/15
100/100 [==============================] - 49s 485ms/step - loss: 0.6269 - rpn_class_loss: 0.0129 - rpn_bbox_loss: 0.1976 - mrcnn_class_loss: 0.0342 - mrcnn_t
Epoch 7/15
100/100 [==============================] - 47s 469ms/step - loss: 0.5401 - rpn_class_loss: 0.0093 - rpn_bbox_loss: 0.1325 - mrcnn_class_loss: 0.0388 - mrcnn_t
Epoch 8/15
100/100 [==============================] - 47s 473ms/step - loss: 0.4432 - rpn_class_loss: 0.0095 - rpn_bbox_loss: 0.1097 - mrcnn_class_loss: 0.0327 - mrcnn_t
Epoch 9/15
100/100 [==============================] - 47s 475ms/step - loss: 0.4787 - rpn_class_loss: 0.0082 - rpn_bbox_loss: 0.1416 - mrcnn_class_loss: 0.0306 - mrcnn_t
Epoch 10/15
100/100 [==============================] - 46s 462ms/step - loss: 0.3981 - rpn_class_loss: 0.0074 - rpn_bbox_loss: 0.1046 - mrcnn_class_loss: 0.0269 - mrcnn_t
Epoch 11/15
100/100 [==============================] - 46s 464ms/step - loss: 0.4184 - rpn_class_loss: 0.0074 - rpn_bbox_loss: 0.1350 - mrcnn_class_loss: 0.0255 - mrcnn_t
Epoch 12/15
100/100 [==============================] - 47s 470ms/step - loss: 0.3953 - rpn_class_loss: 0.0071 - rpn_bbox_loss: 0.1188 - mrcnn_class_loss: 0.0244 - mrcnn_t
Epoch 13/15
100/100 [==============================] - 46s 458ms/step - loss: 0.3420 - rpn_class_loss: 0.0052 - rpn_bbox_loss: 0.1116 - mrcnn_class_loss: 0.0188 - mrcnn_t
Epoch 14/15
100/100 [==============================] - 46s 459ms/step - loss: 0.3569 - rpn_class_loss: 0.0067 - rpn_bbox_loss: 0.1015 - mrcnn_class_loss: 0.0272 - mrcnn_t
Epoch 15/15
100/100 [==============================] - 45s 451ms/step - loss: 0.2719 - rpn_class_loss: 0.0049 - rpn_bbox_loss: 0.0772 - mrcnn_class_loss: 0.0165 - mrcnn_t
```

Figure IV-57: Visualisation de l’entrainement

Il nous reste que configurer les paramètres d'instance du masque R-CNN pour commencer à Tester sa performance :

```
model = modellib.MaskRCNN(
    mode="inference",
    config=config,
    model_dir=DEFAULT_LOGS_DIR
)

model_path = model.find_last()
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)
```

```
Loading weights from  logs/tumor_detector20190901T0743/mask_rcnn_tumor_detector_0015.h5
Re-starting from epoch 15
```

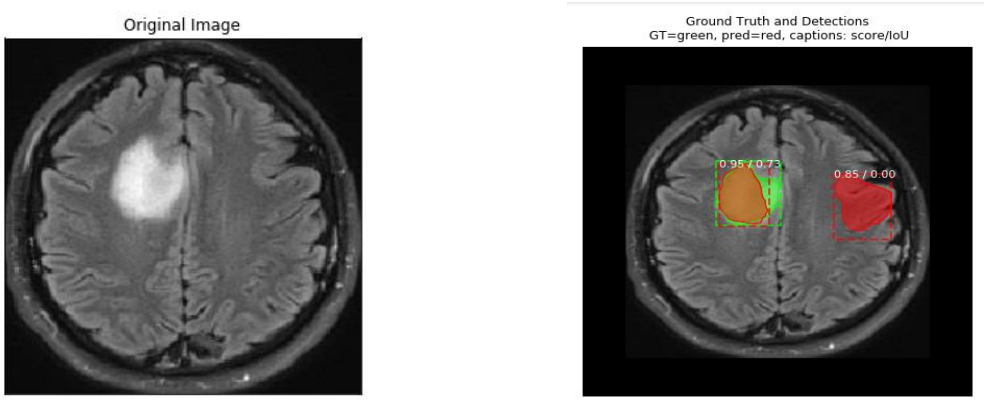
```
def predict_and_plot_differences(dataset, img_id):
    original_image, image_meta, gt_class_id, gt_box, gt_mask =\
        modellib.load_image_gt(dataset, config,
                               img_id, use_mini_mask=False)

    results = model.detect([original_image], verbose=0)
    r = results[0]

    visualize.display_differences(
        original_image,
        gt_box, gt_class_id, gt_mask,
        r['rois'], r['class_ids'], r['scores'], r['masks'],
        class_names = ['tumor'], title="", ax=get_ax(),
        show_mask=True, show_box=True)
```

Figure IV- 58: code de configuration d'instance du masque

IV.5. Tester la performance du masque R-CNN

```
def display_image(dataset, ind):
    plt.figure(figsize=(5,5))
    plt.imshow(dataset.load_image(ind))
    plt.xticks([])
    plt.yticks([])
    plt.title('Original Image')
    plt.show()
```

```
[ ] ind = 0
    display_image(dataset_val, ind)
    predict_and_plot_differences(dataset_val, ind)
```

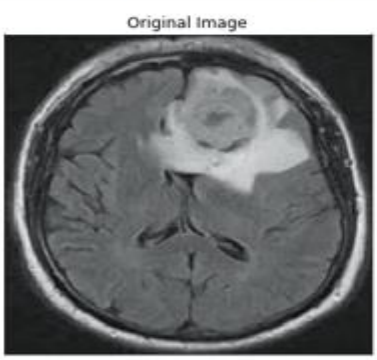


(a) (b)

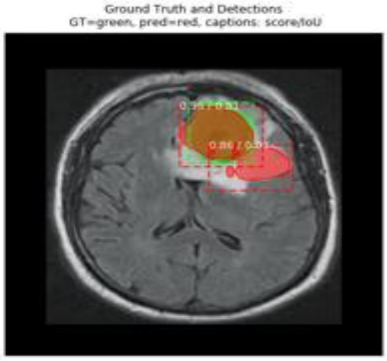
Figure IV-59: Exemple de résultat de Test du masque R-CNN
(a) image originale (b) image avec Mask R-CNN appliqué

Voici d'autres résultats du test de performance du masque avec d'autres images de test:

```
ind = 10
display_image(dataset_val, ind)
predict_and_plot_differences(dataset_val, ind)
```

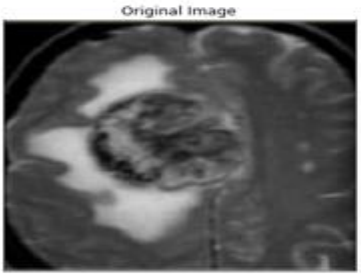


(a)

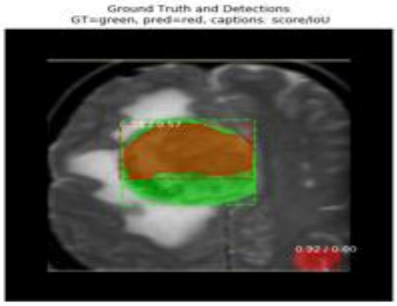


(b)

```
ind = 4
display_image(dataset_val, ind)
predict_and_plot_differences(dataset_val, ind)
```

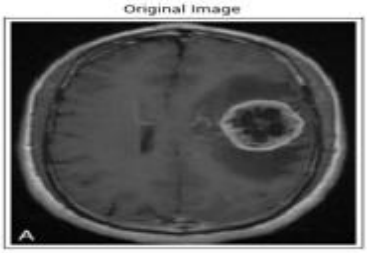


(a)

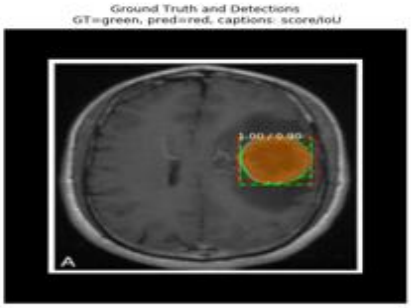


(b)

```
ind = 0
display_image(dataset_test, ind)
predict_and_plot_differences(dataset_test, ind)
```

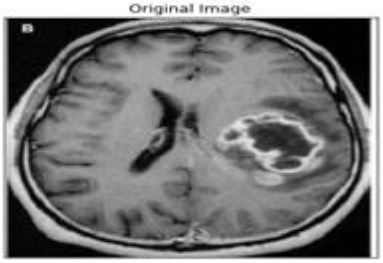


(a)

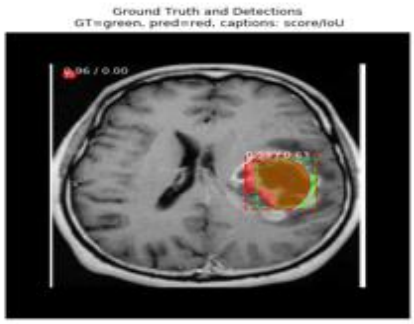


(b)

```
ind = 1
display_image(dataset_test, ind)
predict_and_plot_differences(dataset_test, ind)
```



(a)



(b)

Figure IV-60 : Résultats de test du masque R-CNN
(a) image originale (b) image avec Mask R-CNN appliqué

IV.6. La durée d'entrainement

Voici la comparaison des durées d'entrainement (temps de traitement) sur CPU/ GPU/TPU

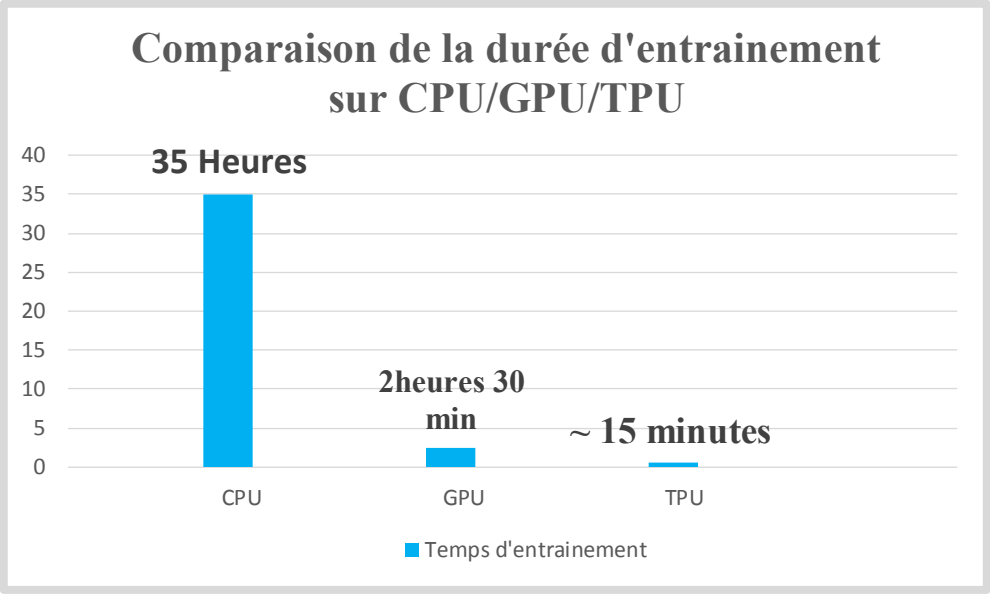


Figure IV-61: Comparaison des durées d'entrainement sur CPU/GPU/TPU

On remarque que la différence est énorme entre les durées de traitement, l'entrainement des réseaux avec un CPU de caractéristiques : Intel(R) Xeon(R) CPU @ 2.20GHz a durée près de 35 heures, tandis qu'avec un GPU l'entrainement a duré 2 heures et 30 minutes seulement ce que nous fait gagner énormément de temps.
Avec le TPU, Théoriquement, on peut booster la performance en arrivant jusqu'aux environs de 15 minutes, cela pourrait être très satisfaisant comme une durée d'entrainement d'un projet de Deep Learning si l'un avait son propre circuit/accélérateur TPU, Car celui qui est disponible sur la plateforme est –pour le moment- limité en quota à cause de la forte demande de ses ressources puisqu'il fait gagner du temps et que tout le monde veut bénéficier de cet avantage.

IV.7. Conclusion

Dans ce chapitre, nous avons implémenté nos réseaux de classification et de détection et avons présenté et comparé les différents résultats obtenus par ces différents réseaux modifiés et même avec de différents accélérateurs de matériel. On a pu marquer les points suivants :

Nous pouvons améliorer la prédiction et être en mesure d'avoir de résultats plus précis avec un ensemble de données plus grand.

Le Transfer Learning va certainement être l'un des principaux moteurs de succès de l'adoption du Machine Learning et de Deep Learning dans l'industrie médicale.

Finalement, le traitement sur le GPU et TPU permet de gagner en temps d'exécution (de quelques jours avec le CPU vers en quelques minutes avec le TPU) et éventuellement d'augmenter l'efficacité et le rendement des applications/projets du Deep Learning.

Conclusion générale

Nous avons essayé, au cours de cette étude, de concevoir un système de classification des images d'IRM cérébrale et détection précise des régions des tumeurs. Nous avons mis en évidence un certain nombre de notions et de définitions concernant le domaine de Deep Learning que nous avons utilisé pour réaliser notre travail.

Nous avons créé une base d'apprentissage, qui a été par la suite annotée, puis nous avons implémenté nos réseaux de Deep Learning à l'aide de VGG 16 convolutif, le Faster R-CNN Inception V3, le réseau Resnet 50 et le masque R-CNN, et avons obtenu des résultats satisfaisants, sachant que le premier (VGG16) avait eu les résultats les plus précis (plus grand pourcentage de réussite de détection) et a été le plus rapide entre les trois.

L'exécution de l'entrainement sur CPU et GPU nous a prouvés qu'il est préférable d'utiliser un accélérateur graphique (GPU) qui est bien plus puissants que le CPU, Ainsi, le TPU est mieux adaptés à la grande quantité de calculs requise par les applications de Deep Learning et les bases de données énormes qu'il requiert vu qu'il est conçu spécialement pour ce type de traitement et de données (Circuit dédié au Deep Learning).

Une fois l'apprentissage réalisé avec succès, le réseau de neurones obtenu peut être utilisé sur plusieurs plateformes ayant un simple CPU comme les Micro-ordinateurs de gamme moyenne ou les Raspberry Pi. Il est aussi à noter que ce réseau peut même être embarqué sur des systèmes connectés ou opérant en temps réel.

Bibliographie

[1] Vincent BARRA: Fusion d'Images 3D du Cerveau : Etude de Modèles et Applications, université d'Auvergne, le 10 Juillet 2000.
[2] X. Leclerc, C. Delmaire , J.-Y. Gauvrit : service de neuroradiologie, université de Lille II, IFR 114, CHU Lille, publié le 21 Mai 2007
[3] Pierre BUSER, Paul LAGET - « HÉMISPHÈRES CÉRÉBRAUX » - Encyclopædia Universalis [en ligne], consulté le 26 juin 2020
[4][5] Gérard Outrequin, Bertrand Boutillier - Neuro-Anatomie fonctionnelle– publié en 2007
[6] Stephen CHICHE - Christine BEYLOUNÉ MAINARDI - IRM – Radiologie du Sud Ouest Parisien 92
[7] Organisation mondiale de la santé https://www.who.int/fr
[8][9] Maximilien Vermandel : Mise en Correspondance Tridimensionnelle d'Images Multimodales Application aux Systèmes d'Imageries Projective et Tomographique d'Angiographie Cérébrale, université de Lille, le 02 juillet 2007
[10] Lisa L.Weyandt – Clinical Neuroscience - Foundations of psychological and neurodegenretive Disorders – Second Edition – page 115 – publié en 2019
[11] Felix P. Kuhn - TEP/IRM: l'imagerie hybride de l'avenir - Klinik für Nuklearmedizin, UniversitätsSpital Zürich – publié en 2011
[12] Rémi Duprès - Imagerie des tumeurs cérébrales intra parenchymateuses – CHRU Nancy
[13] [14] ZEHANI Soraya : Un système à base de réseau de neurones pour la reconnaissance de tumeurs de cancer, université de Biskra, 2008
[15] Minh Hong VU THI : développement d'une sonde peropératoire basée sur la détection d'auto fluorescence pour l'assistance au traitement chirurgical des tumeurs cérébrales, université Paris XI, le 28 Octobre 2008
[16] https://www.m-soigner.com/pratiques/sant%C3%A9-num%C3%A9rique/747-enjeux-defis-et-apports-de-l-intelligence-artificielle-dans-la-medecine-contemproraine.html
[17] https://software.intel.com/content/www/us/en/develop/training/course-artificial-intelligence.html
[18] Daniel Orringer -Une intelligence artificielle détecte les tumeurs du cerveau en temps réel - NYU Langone Health - NYU School of Medicine - Nature Medicine plateforme – publié le 6 janvier 2020.
[19] http://www-lisic.univ-littoral.fr/~verel/TEACHING/08-09/sac-M1/cRdNV9.pdf
[20] https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning/4020611-identifiez-les-differents-types-dapprentissage-automatiques
[21] [22] Bastien Maurice - Fonction d'activation - Deeply Learning Plateforme – publié le 26 Septembre 2018
[23] Prince Grover - 5 Regression Loss Functions All Machine Learners Should Know – HeartBeat Fritz plateforme – publié le 05 Juin 2018
[24] A.Belaid - Réseaux de neurones cours 6 - Réseaux multicouches - Esstt
[25] yacine ouassar : réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus, université pierre et marie curie paris , le 23 avril 2004
[26] https://software.intel.com/content/www/us/en/develop/training/course-deep learning.html#linklist_1231887423?multiplayer=5680465479001

[27] [28] Axel Thevenot - A visual and mathematical explanation of the 2D convolution layer and its arguments – publié le 02 Mai 2020
[29] Fisher Yu , Vladlen Koltun - MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS - Princeton University and Intel Labs – conference paper at ICLR - publié le 30 Avril 2016
[30] SAMUEL BÉDARD-VENNE - réseau de neurones artificiels et applications à la classification d'images - Université DU QUÉBEC À MONTRÉAL – Mai 2018
[31] Karen Simonyan, Andrew Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition, modifié le 10 Avril 2015
[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun - Identity Mappings in Deep Residual Networks – modifié le 25 Juillet 2016
[33] C. Szegedy, S. Ioffe, and V. Vanhoucke – "Inception-v4, inception-resnet and the impact of residual connections on learning" – publié en 2016.
[34] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna - Rethinking the Inception Architecture for Computer Vision – modifé le 11 Décembre 2015
[35] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick - Mask R-CNN - modifié le 24 janvier 2015
[36] Tomasz Malisiewicz, Abhinav Gupta, Alexei A. Efros - Ensemble of Exemplar-SVMs for Object Detection and Beyond - ICCV paper – publié en 2011
[37] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun - Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks - modifié le 06 Janvier 2016
[38] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick - Mask R-CNN - modifié le 24 janvier 2015
[39] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik - Rich feature hierarchies for accurate object detection and semantic segmentation – modifié le 22 Octobre 2014
[40][41] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie - Feature Pyramid Networks for Object Detection – modifié le 19 Avril 2017
[42] Luis Perez, Jason Wang - The Effectiveness of Data Augmentation in Image Classification using Deep Learning – publié le 13 Décembre 2017
[43] Abhishek Dutta, Andrew Zisserman - The VGG Image Annotator (VIA) - University of Oxford – modifié le 09 Aout 2019
[44] Qiang Yang, Sinno Jialin Pan, "A Survey on Transfer Learning", IEEE Transactions on Knowledge & Data Engineering, vol. 22, no. , pp. 1345–1359, October 2010, doi:10.1109/TKDE.2009.191
[45] Venali Sonone - Notes on Deep Learning" Softmax Classifier" – publié le 27 Avril 2019
[46] Jason Brownlee - How to Choose Loss Functions When Training Deep Learning Neural Networks – modifié le 24 Avril 2020 in Deep Learning Performance
[47] Zhenyue Qin, Dongwoo Kim, Tom Gedeon - Rethinking Softmax with Cross-Entropy: Neural Network Classifier as Mutual Information Estimator – modifié le 29 Mars 2020
[48] https://deepai.org/machine-learning-glossary-and-terms/epoch
[49] Jason Brownlee - Understand the Impact of Learning Rate on Neural Network Performance – publié le 25 Janvier 2019 in Deep Learning Performance
[50] Jason Brownlee -Gentle Introduction to the Adam Optimization Algorithm for Deep Learning– publié le 03 Juillet 2017 in Deep Learning Performance
[51] https://deepai.org/machine-learning-glossary-and-terms/accuracy-error
[52] SouvikMandal - How to use Google Colab – GeeksforGeeks - A computer science portal for geeks

Annexes

La diversité des tumeurs cérébrales

L'Organisation Mondiale de la Santé (OMS) a mis en place une nomenclature des tumeurs du système nerveux central dans le but d'aboutir, à terme, à une norme internationale.

	grade I	grade II	grade III	grade IV
Tumeurs des astrocytes				
astrocytome giganto-cellulaire subépendymaire	×			
astrocytome pilocytique	×			
astrocytome		×		
xanthoastrocytome polymorphe		×	×	
astrocytome anaplasique			×	
glioblastome				×
Tumeurs des oligodendrocytes				
oligodendrogliome		×		
oligodendrogliome anaplasique			×	
Tumeurs des oligoastrocytes				
oligoastrocytome		×		
oligoastrocytome anaplasique			×	
Tumeurs épendimales				
subépendymome	×			
myxopapillary	×			
épendymome		×		
épendymome anaplasique			×	
Tumeurs des plexus choroïd				
papillome	×			
carcinome			×	×
Tumeurs neuronales & gliales				
gangliocytome	×			
gangliogliome	×	×		
desmoplastic infantile ganglioglioma	×			
tumeur neuro-épithéliale dysembryoplasique	×			
neurocytome central	×			
Tumeurs pinéales				
pinéocytome		×		
pinéocytome/pinéoblastome			×	×
pinéoblastome				×
Tumeurs embryonnaires				
médulloblastome				×
autres PNETs				×
médulloepithelioma				×
neuroblastome				×
épendymoblastome				×
Tumeurs des nerfs craniens & spinaux				
schwannome	×			
tumeur maligne des gaines périneurales			×	×
Tumeurs des méninges				
méningiome	×			
méningiome atypique		×		
papillary meningioma		×	×	
hémangiopéricytome		×	×	
méningiome anaplasique			×	

Tableau – Classification des tumeurs par grade

Les principes physiques :

La technique de l'IRM exploite les propriétés magnétiques des éléments constitutifs de la matière et se fonde sur les phénomènes physiques de résonance et de relaxation.

Moment magnétique des noyaux :

L'existence du moment magnétique des noyaux a été montrée simultanément par Bloch et

Purcell. Le corps humain étant constitué en moyenne de 70% d'eau, on s'intéresse en pratique à la molécule d'eau et en particulier au noyau d'hydrogène (proton).

Le noyau d'hydrogène se comporte comme une charge en rotation autour de son axe : c'est le mouvement de spin. Les protons peuvent alors être assimilés à des dipôles magnétiques. En l'absence de tout champ magnétique, ceux-ci vont s'orienter dans l'espace de façon aléatoire. Ce mouvement confère au noyau un moment cinétique qui dépend de sa masse et un moment magnétique qui dépend de sa charge.

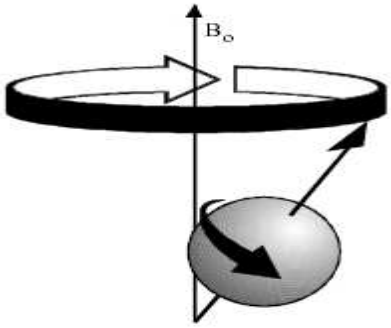


Figure: Le mouvement de spin

Dans un champ magnétique B0, les protons s'orientent alors par rapport à B0 et décrivent autour de ce champ un mouvement de précession, de vitesse angulaire constante. L'ensemble des protons s'orientant dans le sens de B0 forme alors une sous-population de spins +1/2 ; l'ensemble s'orientant dans le sens inverse forme la sous-population de spins −1/2. La sous-population de spins +1/2 étant la plus importante, il existe une aimantation résultante M0 proportionnelle au nombre de protons présents. M0 ne possède qu'une composante longitudinale, dans le sens de B0.

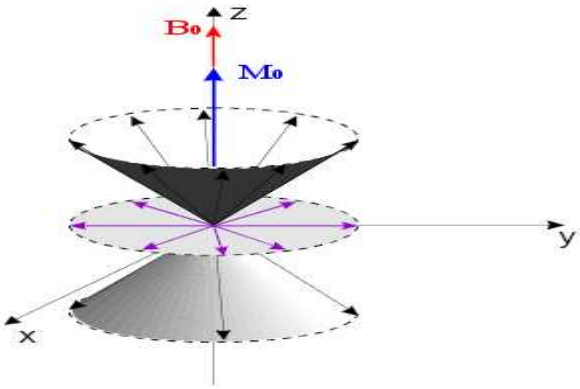


Figure: Mouvement de précession d'un ensemble de moments magnétiques

Phénomènes de résonance et de relaxation

Les phénomènes de résonance et de relaxation sont liés au principe énergétique de la matière. Lorsque des protons, placés dans un champ magnétique, reçoivent un apport d'énergie sous la forme d'ondes radiofréquences de pulsation égale à leur fréquence de résonance et émises par un champ magnétique B1, ils passent d'un niveau bas d'énergie à un niveau haut d'énergie ; cette transition correspond au phénomène de résonance magnétique. L'orientation du champ magnétique résultant change et passe à un nouvel état d'équilibre M tant que B1 dure. On décompose alors M en un moment magnétique longitudinal ML et en un moment magnétique transversal MT. Le retour à l'équilibre des protons, la relaxation, lors de la disparition du champ magnétique B1, s'accompagne d'un mouvement en spirale des protons autour du champ magnétique B0. Bloch a montré que l'évolution de ce mouvement est liée au temps de relaxation longitudinal T1 et au temps de relaxation transversal T2. Les valeurs de ces derniers dépendent des tissus biologiques rencontrés.

	T_1	T_2
liquide céphalo-rachidien	2500 ms	2000 ms
matière grise	900 ms	90 ms
matière blanche	750 ms	80 ms
graisse	300 ms	40 ms

Tableau : Ordre de grandeur des temps de relaxation à 1.5 Tesla

Les différentes pondérations

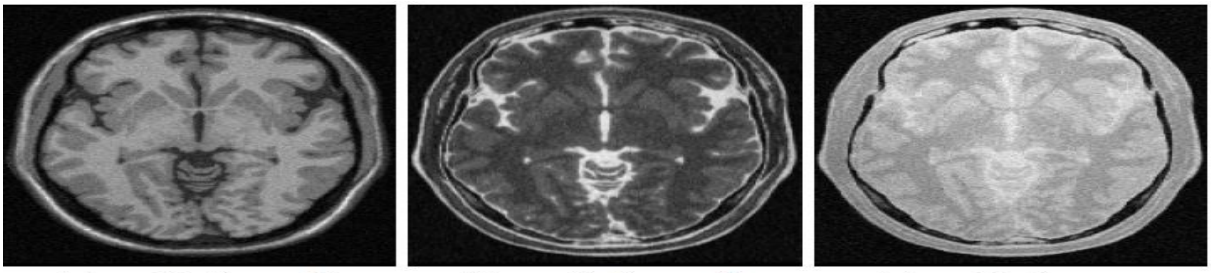
Nous pouvons identifier les paramètres qui influencent le contraste de l'image IRM en deux grandes classes :

– La première est constituée de paramètres intrinsèques liés directement aux tissus observés.

Il s'agit de la densité en protons **P**, des temps de relaxation T1 et T2, de la présence d'un agent de contraste ou encore de la vitesse des fluides circulant.

– La seconde est constituée de paramètres liés à l'appareil lui-même (en particulier l'intensité et la constance du champ magnétique B0) et à la séquence d'acquisition. Cette dernière dépend essentiellement du temps de répétition TR séparant deux impulsions de l'onde radiofréquence B1 et du temps d'écho TE séparant l'impulsion de la lecture du signal. Parmi les séquences classiques, on distingue les séquences spin écho, écho gradient et inversion-récupération.

Parmi ces paramètres, seuls les paramètres d'acquisition sont modulables et conduisent à l'obtention d'images dites « pondérées en T1 », « pondérées en T2 » ou encore « pondérées en **P** ».



(a) pondération en T_1 (b) pondération en T_2 (c) pondération en ρ

Figure: Une même coupe obtenue sous différents paramètres d'acquisition

Image pondérée en T1

Pour des TR de l'ordre de 600 ms, le contraste entre les tissus dépend essentiellement de leur vitesse d'aimantation, donc de T1. Pour des TE d'environ 20 ms, les différences de décroissance du signal entre les tissus n'ont pas le temps de s'exprimer, rendant le contraste indépendant de T2. Ainsi, on obtient une image pondérée en T1, où les tissus sont ordonnés par niveaux de gris croissants en liquide céphalo-rachidien, matière grise puis matière blanche.

Image pondérée en T2

Pour des TR de l'ordre de 2 s et des TE d'environ 90 ms, la décroissance du signal domine la différence de densité protonique entre tissus. Le signal est alors suffisant pour réaliser une image dite pondérée en T2, où les tissus sont ordonnés par niveaux de gris croissants en matière blanche, matière grise puis liquide céphalo-rachidien.

Image pondérée en densité de protons P

Pour un TR de l'ordre de 2 s et un TE court d'environ 20 ms, la différence de densité protonique entre la matière grise et la matière blanche s'exprime. On obtient une séquence qui reflète la localisation et la concentration des noyaux d'hydrogène des différentes structures. Les tissus sont ordonnés par niveaux de gris croissants en matière blanche, matière grise puis liquide céphalorachidien.

Influence du produit de contraste Gadolinium

L'injection de gadolinium présente un intérêt majeur dans le diagnostic des tumeurs cérébrales. En effet, sans ce produit de contraste, une image peut ne pas refléter la présence de la tumeur. La fixation du gadolinium, en modifiant les propriétés magnétiques des éléments, transforme leur réponse au champ radiofréquence en raccourcissant le T1 et en créant de l'hyper signal. En T2, l'injection de gadolinium soit ne modifie pas le contraste des lésions, soit, à de fortes concentrations, « éteint » l'hyper-signal spontané.

Même si le gadolinium permet la création d'un hyper-signal au niveau de la tumeur, il faut néanmoins utiliser cette information avec prudence. D'une part, l'hyper-signal ne peut s'exprimer qu'aux endroits où le gadolinium a circulé ; or, certaines tumeurs sont partiellement ou totalement imperméables à ce produit (figure). D'autre part, l'injection de gadolinium sous-estime le volume de la lésion. Dans ces deux cas, on mesure donc l'intérêt de réaliser une acquisition en T2 complémentaire.

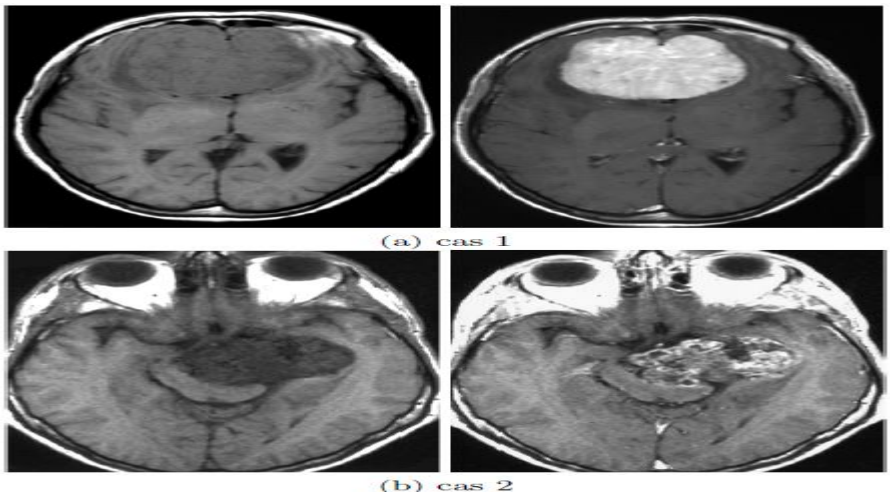


Figure: Intérêt d'une acquisition avec injection de gadolinium

Ces deux cas représentent une coupe pondérée en T1 et la même après injection de gadolinium. Dans le premier cas, le gadolinium a parfaitement rehaussé la tumeur.

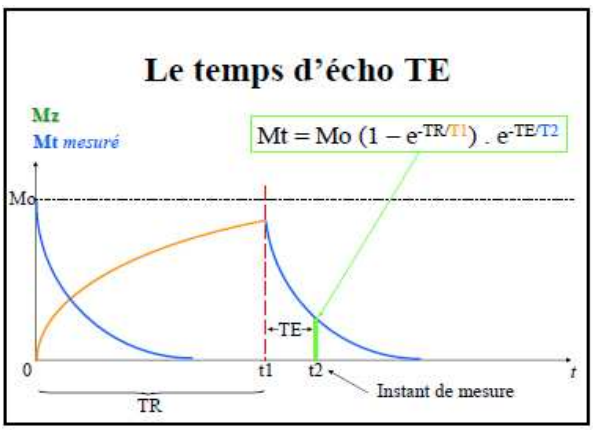
Dans le second, seule une partie de la tumeur est mise en évidence. Ceci illustre à la fois l'intérêt d'une acquisition après injection du produit de contraste mais aussi la prudence qu'il faut observer lors de l'analyse des clichés. L'utilisation conjointe des différentes pondérations d'acquisition est particulièrement intéressante dans le cadre du diagnostic des lésions cérébrales. En effet, il est fréquent qu'une tumeur n'apparaisse que partiellement sur une image. La solution adoptée est alors de multiplier les acquisitions et les pondérations utilisées et éventuellement de faire des acquisitions sous produit de contraste de manière à compléter successivement l'information sur le processus recherché. Chaque pondération, qui apporte ainsi une part d'information complémentaire, mais aussi redondante, peut ainsi être considérée comme une observation particulière de la zone cérébrale. Notons que ces observations peuvent parfois être conflictuelles, par exemple lorsque qu'une tumeur est complètement absente sur une pondération et apparaît sur une autre. L'utilisation conjointe par les médecins de l'ensemble des images, leur permet de synthétiser l'information et ainsi d'avoir une information plus complète sur la zone tumorale.

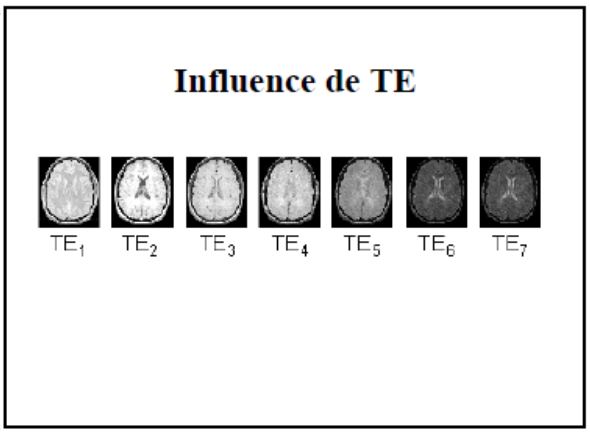
Paramètres de séquence

Les paramètres de séquence sont les paramètres que le manipulateur fixe sur la console pour définir la séquence IRM. Ils permettent de contrôler l'influence des différents paramètres tissulaires T_1, T_2 et densité de proton dans le signal («pondération») et de moduler ainsi le contraste dans l'image.

Temps d'écho

Le signal de précession libre ne peut être enregistré directement après l'excitation (déphasage parasite induit par les gradients). C'est pourquoi il est acquis sous la forme d'un écho de spin ou de gradient. Par définition, le délai entre le milieu de l'impulsion d'excitation et le sommet de l'écho est appelé temps d'écho, et noté TE. Dans la méthode d'écho de spin, les hétérogénéités de $\boldsymbol{B_0}$ et les différences d'aimantation des tissus sont compensées, alors qu'elles ne le sont pas en écho de gradient. La courbe de décroissance est donc différente pour ces deux techniques. Le temps de relaxation correspondant est donc lui aussi différent. Noté T_2 en écho de spin, il est noté T_2^* en écho de gradient (temps de relaxation transversale effectif) et fait intervenir des éléments extra tissulaires comme la non-uniformité de $\boldsymbol{B_0}$. D'une manière générale, le TE gouverne la pondération en T_2 dans le contraste de l'image.





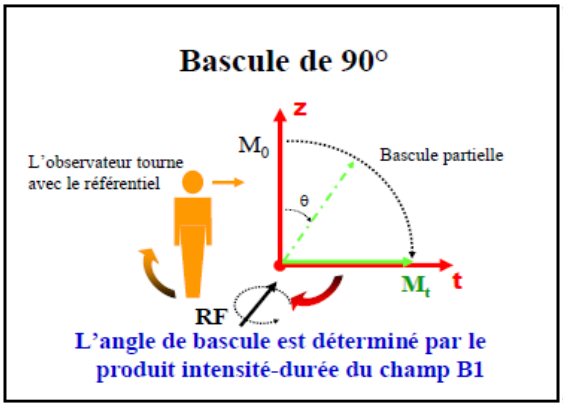
Temps de répétition

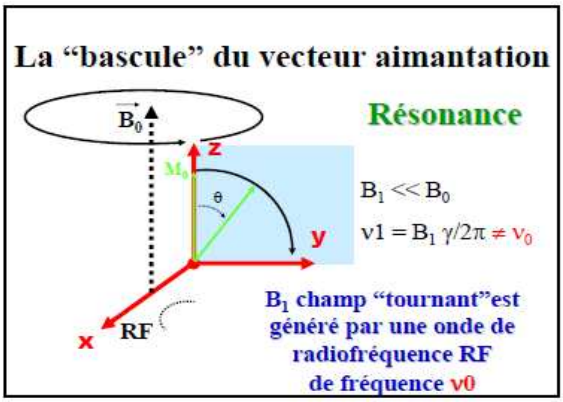
L'image est constituée à partir de la répétition de la même séquence avec un gradient de phase $\boldsymbol{G_p}$ d'amplitude différente. Le temps qui sépare deux répétitions est appelé temps de répétition et noté TR. Le TR, comme le TE, est un facteur de contraste. S'il est suffisamment long, toute l'aimantation repousse et le signal ne dépend pas de la vitesse d'aimantation (donc de T_1), mais essentiellement de la densité protonique. S'il est court, le système atteint après quelques répétitions un régime stationnaire et

l'aimantation tend vers une valeur d'équilibre dépendant de la vitesse d'aimantation des tissus, et donc de leur T_1. L'image révèle ainsi les différences de T_1 entre les tissus.

Angle de bascule

Si $\boldsymbol{B_1}$ est orthogonal à $\boldsymbol{B_0}$, ce qui est généralement le cas, le phénomène de résonance magnétique bascule l'aimantation $\boldsymbol{M}$ selon un axe perpendiculaire au champ principal $\boldsymbol{B_0}$. Si $\boldsymbol{M}$ est basculé à 90° (excitation par une impulsion $\pi/2$), toute l'aimantation est dans le plan transversal et $\boldsymbol{M_L}$ est nulle. En cas de bascule d'un angle inférieur à 90°, seule une partie de l'aimantation est convertie en signal ($\boldsymbol{M_T}$) et il persiste une aimantation $\boldsymbol{M_L}$ pouvant être utilisée pour une autre excitation. L'angle de bascule correspond donc à une énergie délivrée par le champ $\boldsymbol{B_1}$. Le signal S sera d'autant plus faible que cet angle sera petit. En régime stationnaire, l'angle de bascule α intervient dans le contraste de l'image et gouverne la réserve en aimantation. Pour des angles petits (α<20°), la densité protonique est prépondérante. Plus α est grand, plus le T_1 gouverne le contraste.





Annexes : TPU (Tensor Processing Units)

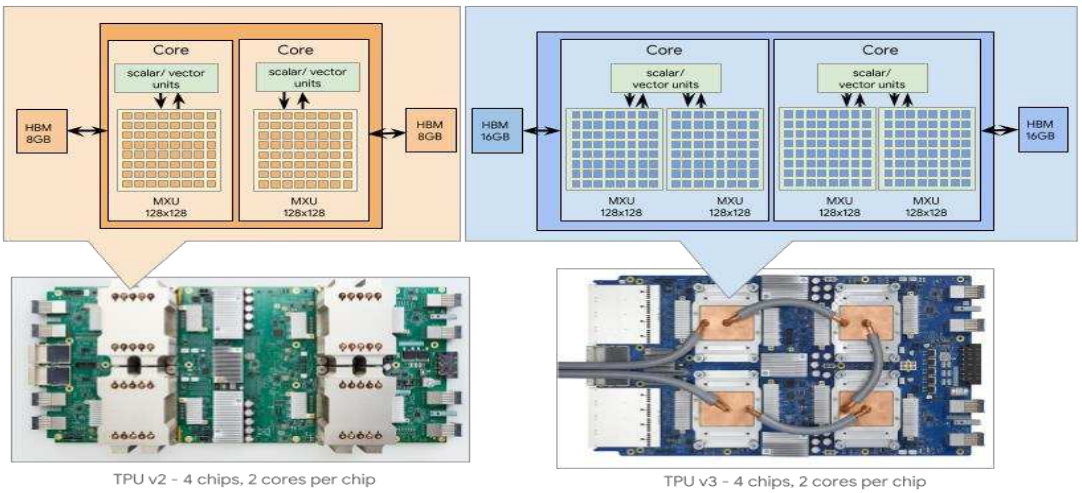
Aperçu du TPU

Les TPU (Tensor Processing Units) sont des circuits intégrés spécifiques aux applications (Application-Specific Integrated Circuit ou ASIC), développés spécifiquement par Google et permettant d'accélérer les charges de travail de deep learning.

On peut utiliser Cloud TPU pour exécuter nos propres charges de travail sur le matériel accélérateur des TPU de Google. Cloud TPU est conçu pour offrir des performances et une flexibilité maximales afin d'aider les chercheurs, les développeurs et les entreprises à créer des clusters de calcul capables d'exploiter des processeurs, des GPU et des TPU. Les API de haut niveau facilitent l'exécution de modèles dupliqués sur le matériel Cloud TPU.

Versions de TPU

Chaque version de TPU définit les caractéristiques matérielles spécifiques d'un appareil TPU. La version de TPU définit l'architecture de chaque cœur de TPU, la quantité de mémoire à haut débit (HBM) des cœurs, les interconnexions entre les cœurs de chaque appareil TPU et les interfaces réseau disponibles pour la communication entre appareils.



Chaque cœur de TPU comprend des unités scalaires, vectorielles et matricielles (MXU). Les unités matricielles fournissent l'essentiel de la puissance de calcul d'une puce TPU. Chaque unité matricielle est capable d'effectuer 16 000 opérations multiply-accumulate par cycle. Tandis que les entrées et sorties des unités matricielles sont des valeurs à virgule flottante 32 bits, les unités matricielles effectuent des multiplications avec une précision réduite de bfloat16.

Chacun des cœurs d'un appareil TPU peut effectuer des calculs utilisateur (opérations XLA) indépendamment. Des interconnexions à bande passante élevée permettent aux puces de communiquer entre elles directement sur l'appareil TPU. Dans une configuration de pod TPU, des interfaces réseau haut débit dédiées relient plusieurs appareils TPU pour fournir davantage de cœurs de TPU et un pool de mémoire TPU plus important pour les charges de travail de deep learning.

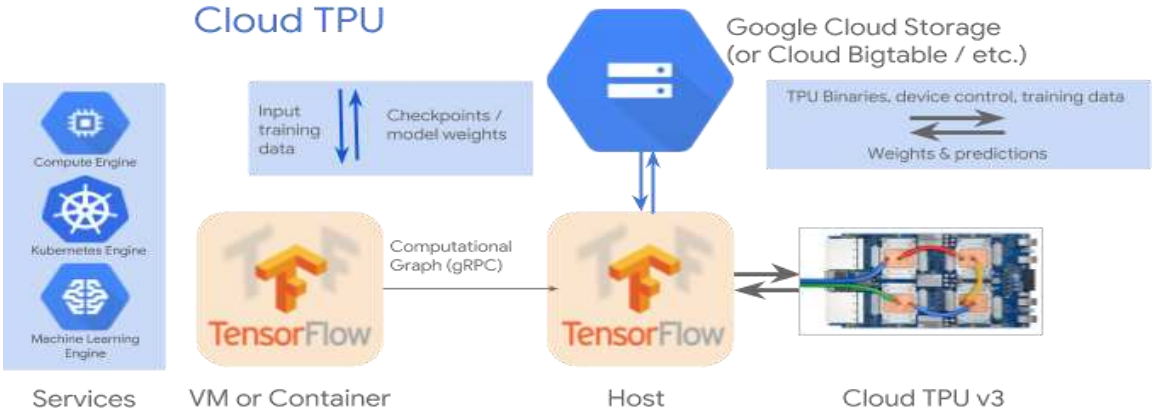
Configurations TPU

Dans un centre de données Google, les appareils TPU sont disponibles dans les configurations suivantes pour Cloud TPU v2 et Cloud TPU v3 :

- Appareils TPU uniques : appareils TPU individuels qui ne sont pas interconnectés sur un réseau haut débit dédié. Il n'est pas possible de combiner plusieurs types d'appareils TPU uniques en vue de collaborer sur une charge de travail unique.
- Pods TPU : clusters d'appareils TPU interconnectés sur des réseaux haut débit dédiés.

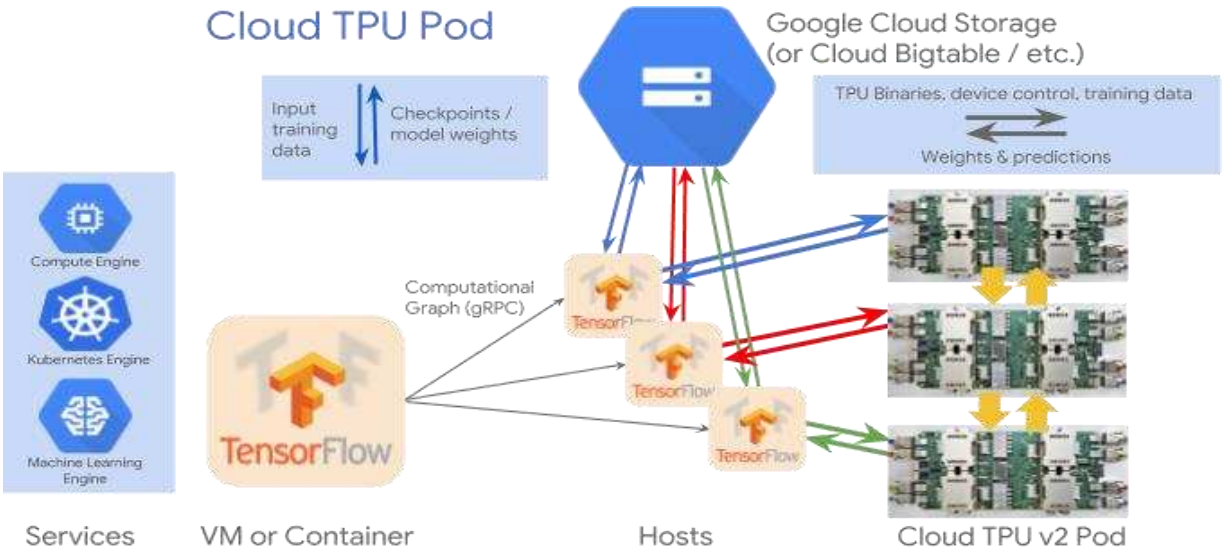
Appareil TPU unique

Dans un centre de données Google, une configuration d'appareil TPU unique consiste en seul appareil TPU sans connexion réseau haut débit dédiée à d'autres appareils TPU.



Pods TPU

Dans un centre de données Google, une configuration de pod TPU comprend plusieurs appareils TPU interconnectés sur une connexion réseau haut débit dédiée.



Les puces d'un pod TPU sont interconnectées sur l'appareil. Par conséquent, la communication entre les puces ne nécessite pas de ressources processeur hôtes ni de ressources réseau hôtes. En outre, tous les appareils TPU d'un pod TPU sont interconnectés sur des réseaux haut débit dédiés qui ne nécessitent pas non plus de ressources processeur hôtes ni de ressources réseau hôtes.

On peut utiliser l'API Cloud TPU pour automatiser la gestion des TPU pour nos nœuds TPU, quelle que soit leur taille. De cette manière, on peut facilement déployer d'énormes clusters de calcul, exécuter nos charges de travail, puis réduire ces clusters une fois les charges de travail terminées. L'assistance matérielle intégrée aux puces permet de bénéficier d'un scaling linéaire et performant pour une large gamme de charges de travail. En pratique, la pile logicielle Cloud TPU élimine la complexité au niveau de la création, l'exécution et l'alimentation des programmes Cloud TPU.

Architecture logicielle

Le nœud TPU compile le graphe de calcul à la volée et envoie le binaire du programme à un ou plusieurs appareils TPU pour exécution. Les entrées du modèle sont souvent stockées dans Cloud Storage. Le nœud TPU transmet les entrées à un ou plusieurs appareils TPU pour qu'elles soient traitées.

Le diagramme illustre l'architecture logicielle Cloud TPU constituée du modèle de réseau de neurones (TPU Estimator), du client TensorFlow, du serveur TensorFlow et du compilateur XLA.