الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

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

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie
قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière électronique

Spécialité électronique des systèmes embarqués

Présenté par

Abid Fatima
&
Brahim Boulares Lobna

Thème

Détection de la rétinopathie diabétique avec le deep learning, transfer learning, CNN, U-Net.

Encadré par : Mme Bougherira Hamida

Année universitaire 2020/2021

Remerciements :

Nous saisissons cette occasion pour exprimer notre gratitude et nos vifs remerciements à notre encadrante, Madame **BOUGHERIRA HAMIDA**, pour la confiance qu'elle nous a accordé, chose qui nous a incité à multiplier nos efforts pour être à la hauteur de ses attentes.

Un merci spécial à Monsieur **FARES KARAOUI** pour son attention dans ce travail et pour toute l'aide qu'il nous a apportée.

On remercier aussi madame **NACEUR DJAMILA**, notre chef de spécialité en master électronique des systèmes embarquées.

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.

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 (colab) et jupyter notbooks , pour détecter la rétinopathie diabétique dans les images en utilisant le « Deep Learning » et le « Transfer Learning ». Nous avons utilisé une base de données contenant 5000 images augmentées images pour l'entrainement. Nous avons ensuite utilisé le Transfer Learning en modifiant les structures des réseaux VGG 16, alex net, et Inception V3 pour la classification. Des résultats d'apprentissage, de test, et de validation satisfaisants ont été obtenus.

Mots clés : rétinopathie diabétique, deep Learning, transfert Learning, Google colab, jupyter, CNN, data augmentation, VGG, Inception, Alex net, intelligence artificielle, imagerie médicale.

Abstract: The objective of this project is to implement a medical diagnostic aid system, based on different architectures of convolutional neural networks in a Cloud(colab) and jupyter notbooks environment, to detect diabetic retinopathy in images using "Deep Learning" and "Transfer Learning". We used a database containing 5000 images augmented images for training. We then used Transfer Learning by modifying the structures of the VGG 16, alex net, and Inception V3 networks for classification. Satisfactory Learning, testing, and validation results were obtained.

Keywords: diabetic retinopathy, deep Learning, transfer Learning, Google colab, jupyter, CNN, data augmentation, VGG, inception, alex net, artificial intelligence, medical imaging.

الملخص: الهدف من هذا المشروع هو تنفيذ نظام مساعدة تشخيصية طبية، يعتمد على بنى مختلفة للشبكات العصبية التلافيفية في بيئة سحابية، لاكتشاف اعتلال الشبكية السكري في الصور باستخدام "التعلم العميق" و "نقل التعلم". استخدمنا قاعدة بيانات تحتوي على 5000 صورة مكثفة للتدريب. ثم استخدمنا Transfer Learningعن طريق تعديل هياكل شبكات VGG 16 و alex netو Inception V3للتصنيف. تم الحصول على نتائج التعلم والاختبار والتحقق المرضية.

الكلمات الرئيسية: اعتلال الشبكية السكري، التعلم العميق، نقل التعلم، زيادة البيانات

Liste des acronymes et abréviation

IRM : imagerie par résonance magnétique

RD : rétinopathie diabétique

VEGF : vascular endothélium growth factor

IA : intelligence artificielle

CNN : convolutional neural network

R-CNN : region-based concolutional neural network

RNA : réseau de neurones artificiel

PMC : perceptron multi couche

FC : fully connected

VGG : Visual geometry group

CONV : convolution

ResNet : residual Network

SGD : stochastic gradient descent

GPU : graphics processing unit (unité de traitement graphique)

TPU : tensor processing unit

CPU : central processing unit (unité centrale de traitement)

CSV : comma-separated Values

LR : Learning Rate

Adam : adaptive moment estimation

VAL : validation

RAM : Random Access memory

ReLU : rectified linear unit

U-NET : convolutional networks for biomedical image segmentation

Table de matières :

Liste des figures

Chapitre I généralités sur la rétinopathie diabétique et le deep Learning

Chapitre II : développement de la méthode de détection et reconnaissance de la rétinopathie diabétique par les algorithmes de deep Learning

Chapitre III implémentation et résultats

Liste des tableaux :

Introduction générale :

L'Intelligence Artificielle va bouleverser durant les prochaines années de nombreux secteurs dont la santé et l'imagerie médicale. Par l'utilisation d'algorithmes de Deep Learning, les diagnostics sont de plus en plus précis, les risques d'erreur amoindris et les coûts abaissés.

Un système d'apprentissage profond (deep Learning) est une machine d'intelligence artificielle capable d'apprendre de chaque image qu'elle reçoit et progressivement d'améliorer ses performances. Le dépistage de la rétinopathie diabétique et des pathologies ophtalmologiques en rapport avec le diabète est un bon exemple de ce que peut apporter cette technologie à la médecine.

La rétinopathie diabétique (atteinte des yeux : œil et rétine) est une grave complication du diabète qui touche 50% des patients diabétiques de type 2. Les yeux sont particulièrement sensibles à l'atteinte des petits vaisseaux.

Dans ce projet, nous créons, sur la plateforme Colab et jupyter notbooks et à l'aide du langage de programmation Python et ses bibliothèques (notamment **Tensorflow**) et bien 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 de la rétine.

L'ensemble des images utilisées se compose de 5000 images de la rétine divisées en cinq classe comme ceci : « classe (sain), en classe (légère), en classe (Modéré), en class (grave), en classe (prolifération) ». La même base de données a été divisée pour la deuxième fois en trois classe comme ceci : « classe (sain), classe (légère), classe (grave) », ensuite nous l'avons divisé en deux classe sain et malade, nous l'avons divisé trois fois afin de pouvoir comparer les résultats d'apprentissage.

Ce mémoire est composé de 3 chapitres :

- Chapitre1 : généralités sur la rétinopathie diabétique et le deep Learning
- Chapitre 2 : développement de la méthode de détection et reconnaissance de la rétinopathie diabétique par les algorithmes de deep Learning.
- Chapitre 3 : implémentation et résultats

Chapitre I :

Généralités sur la rétinopathie diabétique et le Deep Learning

I.1 Introduction :

La rétinopathie diabétique est une complication oculaire du diabète.

Dans ce chapitre, nous étudierons l'œil et la rétinopathie diabétique, puis nous présenterons des méthodes de détection de la rétinopathie diabétique par imagerie médicale.

Ensuite, nous présenterons les réseaux de neurones et leurs topologies, les CNN et Ses différentes couches et algorithmes de détection, notamment R-CNN et Fast RCNN, R-CNN faster et Mask R-CNN, ainsi que certaines des bases mathématiques sur lesquelles repose le deep Learning.

I.2 L'œil et le cerveau :

I.2.1 L'œil :

a. Définition :

L'œil, correspond à l'organe de la vue, captant le signal lumineux avant que l'information ne soit réinterprétée par le cerveau et transformée en formes et en couleurs. Il se compose de différentes régions lui permettant d'assurer sa fonction, de la cornée jusqu'à la rétine.

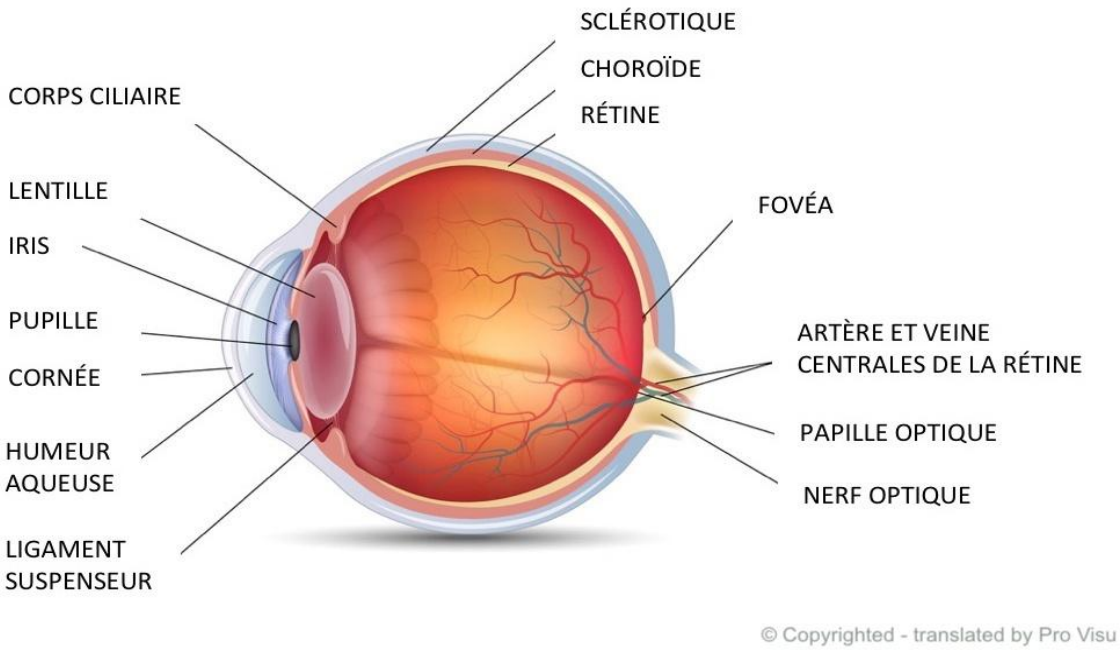


Figure I.1 : les différentes couches et les divers composants de l'œil.

b. L'œil humain :

L'œil humain correspond grossièrement à une petite boule de 2,5 cm de diamètre, pesant sept à huit grammes. Le globe oculaire se compose de différentes couches. La plus externe correspond à la sclère, membrane résistante qui donne sa couleur blanche à l'œil en le recouvrant à 80 %, sur laquelle s'insèrent les muscles oculomoteurs. Au centre se trouve la cornée, qui permet de capter les premiers rayons de lumière et de commencer la réfraction.

Sous la sclère, on trouve la choroïde, couche fine irriguée de vaisseaux sanguins et assurant la nutrition de la rétine et de l'iris.

La rétine, située dans le fond du globe oculaire, au contact de la choroïde, correspond à la région directement sensible à la lumière, polarisée principalement par la cornée et le cristallin. Elle est composée de cellules en cônes et en bâtonnets, sensibles aux détails de l'environnement et à la luminosité. C'est à ce niveau que le signal lumineux va être traduit en message nerveux, et être envoyé au cerveau par le nerf optique.

Au centre de l'œil, derrière la cornée, on trouve l'iris, région qui confère sa couleur particulière au globe oculaire. En son centre, on trouve la pupille, région sombre qui, sous l'action de muscles, se contracte ou se dilate pour laisser passer plus ou moins de lumière, à l'instar d'un diaphragme sur un appareil photo.

L'essentiel du volume de l'œil (90 %) est dû à l'humeur vitrée, liquide gélatineux situé en arrière du cristallin, qui donne au globe oculaire sa consistance. Il contribue également à la réfraction de la lumière.

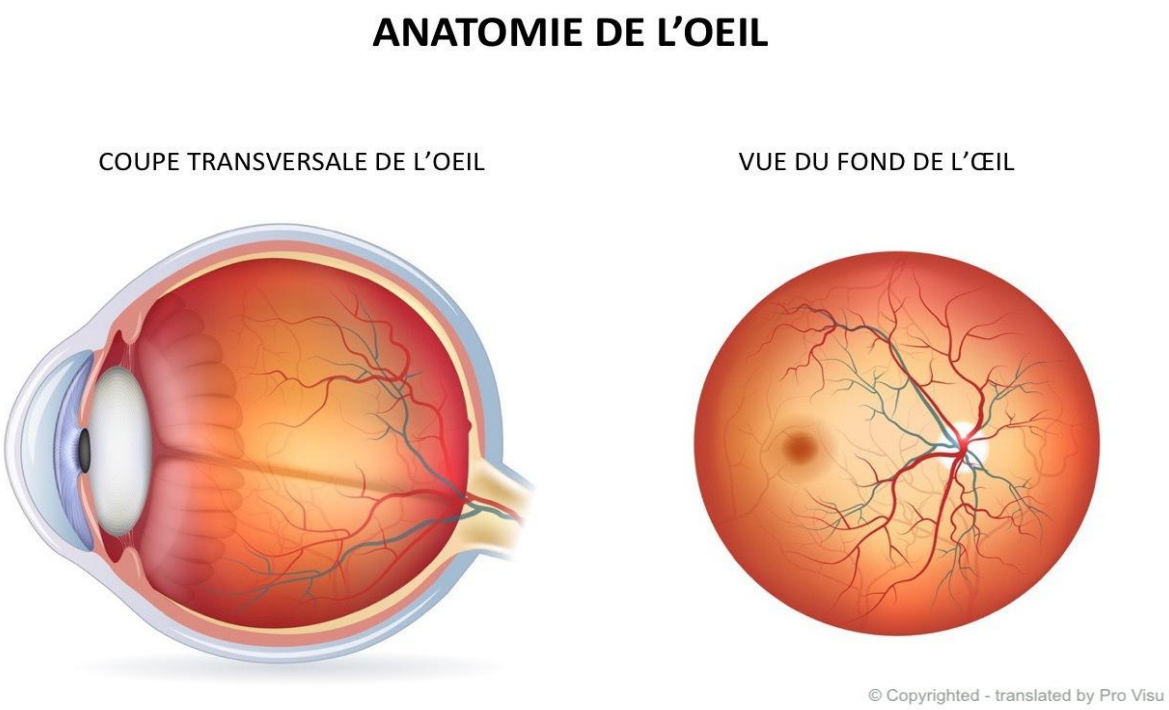


Figure I.2 : Anatomie de l'œil

c. La vision :

Ce sont les **rayons lumineux** présents dans notre environnement qui permettent la vision, et les différents organes de l'œil sont à l'origine d'un ensemble de mécanismes de perception de la lumière.

Le flux lumineux est tout d'abord perçu par l'iris, qui adapte la taille de la pupille en fonction de celui-ci.

La lumière traverse ensuite les milieux oculaires, à savoir le cristallin et le corps vitré, qui doivent donc être transparents pour que celle-ci soit correctement transmise.

Puis, elle atteint la rétine et ses **cellules photo-réceptrices** :

- Les **cônes**, principalement situés dans la rétine centrale (macula), sont responsables de la vision des couleurs, des formes des détails, et sont associés à la vision de jour.
- Les **bâtonnets**, principalement situés dans la rétine périphérique, sont beaucoup plus sensibles à la lumière, responsables de la vision des contours et des mouvements, et sont associés à la vision en basse luminosité.

Cet ensemble d'organes transforme la lumière en **signaux électriques** alors envoyés au cerveau via le nerf optique, l'image est alors interprétée.

d. L'œil et le cerveau :

Les aires cérébrales visuelles représentent près d'un tiers de notre cerveau et sont chacune spécialisée dans un type de traitement particulier, du plus perceptif au plus cognitif, précisant de mieux en mieux la scène visuelle observée, jusqu'à aboutir à une représentation visuelle complète qui a du sens pour l'observateur.

A la sortie de la rétine, il n'existe plus de scène visuelle à proprement parler. Les informations visuelles sont transmises sous forme d'influx électrique de l'œil au cerveau, et c'est au niveau cérébral que la scène va être reconstruite en fonction des différentes informations portant sur la couleur, la forme, le mouvement, la localisation spatiale etc… que les aires cérébrales vont analyser. Ce que nous voyons est donc une construction de notre cerveau et non une stricte photographie du monde extérieur que nos yeux auraient prise.

De l'œil au cerveau, il va y avoir une réorganisation des fibres nerveuses au niveau du chiasma optique qui fait que chaque hémisphère cérébral va traiter les informations issues d'une seule moitié du champ visuel ou hémi champ. Cette représentation est croisée c'est-à-dire que l'hémisphère droit traitera les informations de l'hémi champ gauche et inversement.

Au-delà des régions occipitales, les aires visuelles vont réaliser des traitements de plus en plus complexes, et vont être organisées en deux voies principales : une voie dorsale qui rejoint le cortex pariétal et une voie ventrale qui est reliée au cortex temporal. La voie dorsale sera spécialisée dans le traitement de l'espace, utile soit pour agir sur un objet ou pour comprendre l'organisation spatiale de la scène. La voie ventrale relaie quant à elle l'information des détails et des couleurs et sera plus impliquée dans la reconnaissance de formes.

De plus, le cerveau visuel communique continuellement avec le reste des aires cérébrales comme celles du langage, de la mémoire ou des émotions qui amènent du sens à ce que nous voyons, et qui peuvent également influencer notre perception visuelle [1].

I.3 La maladie de diabète :

I.3.1 Définition :

Le diabète est une maladie qui empêche le corps d'utiliser correctement l'énergie fournie par les aliments ingérés. Par ailleurs, la maladie survient lorsque le pancréas ne sécrète plus d'insuline ou lorsque le corps devient résistant à l'insuline produite.

Si le glucose reste dans le sang, la glycémie augmente. À long terme, cela peut entraîner le dysfonctionnement et la détérioration de nombreux organes comme les yeux et les reins [2].

I.3.2 Les complications :

Les complications du diabète sont nombreuses et peuvent être sévères. Ces complications aggravent le diabète et tendent à faire baisser l'espérance de vie des personnes atteintes. La majorité des complications liées au diabète peuvent être évitées, réduites ou retardées si le diabète est dépisté et traité précocement et correctement. Les principales complications du diabète sont :

- La rétinopathie diabétique (responsable à terme d'une cécité),
- Les complications cardio-vasculaires,
- La néphropathie diabétique aboutissant à 'insuffisance rénale, la neuropathie diabétique,
- Les infections, les ulcères de pied et de jambe [3].

I.4 La rétinopathie diabétique :

I.4.1 Définition :

La rétinopathie diabétique, caractérisée par des lésions de la rétine de l'œil, est une grave complication du diabète qui touche 50 % des patients diabétiques de type 2 (à différents stades). Le diabète est ainsi responsable de 12 % de l'ensemble des cas de cécité dans les pays de monde ! Plusieurs facteurs favorisent la survenue d'une rétinopathie diabétique et accélèrent sa progression : l'ancienneté du diabète, le niveau de glycémie, un diabète instable, l'hypertension artérielle, le tabagisme, etc. Cette pathologie peut accélérer la survenue d'autres maladies des yeux comme les glaucomes ou la cataracte, et même conduire à la cécité en l'absence d'un traitement adapté.

I.4.2 L'origine :

Réceptionnant les ondes lumineuses et les transmettant au cerveau via le nerf optique, la rétine (voir schéma ci-dessous) est une fine membrane de l'œil parcourue par une multitude de petits vaisseaux, les capillaires. L'excès de sucre dans le sang - comme dans le cas d'un diabète - fragilise la paroi de ces derniers, entraînant une perte d'étanchéité. Il s'ensuit la rupture puis l'éclatement de ces vaisseaux (on parle de "micro-anévrismes"). Au fur et à mesure, des zones étendues de la rétine ne sont plus oxygénées. En réaction, la rétine produit de nouveaux vaisseaux encore plus fragiles.

Le phénomène s'amplifie et s'étend jusqu'à la macula (zone au milieu de la rétine), où se situe le centre de la vision. La macula s'épaissit, et il se produit alors un œdème maculaire responsable d'une baisse de l'acuité visuelle qui peut être très importante et que partiellement réversible. De surcroît, les vaisseaux nouvellement produits par la rétine peuvent saigner dans le vitré (zone située devant la rétine), conduisant dans certains cas à un risque de déchirure et donc de décollement de la rétine, responsable d'une perte définitive de la vision [4].

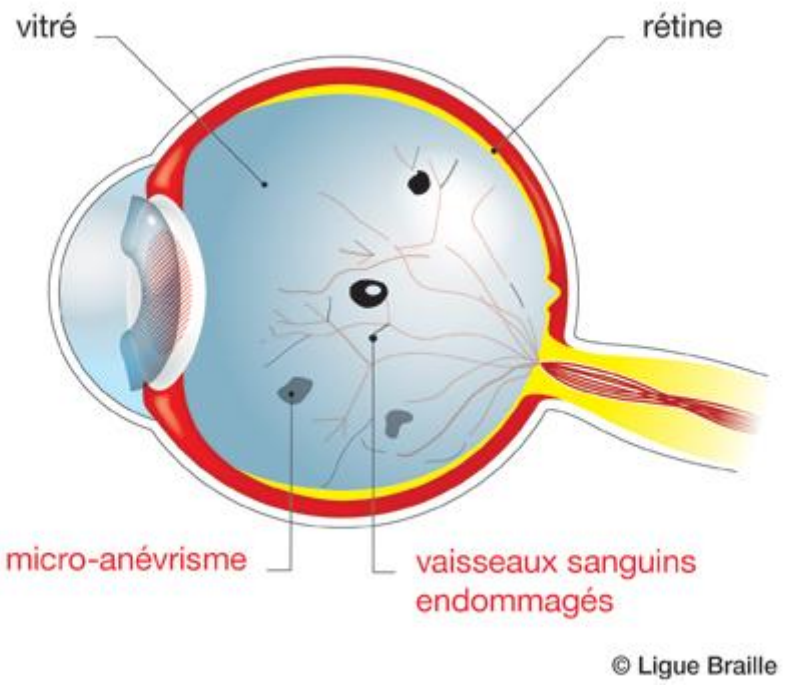


Figure I.3 : Œil avec rétinopathie

I.4.3 Les classifications de la rétinopathie diabétique :

a. La rétinopathie non proliférante :

La rétinopathie non proliférante (également appelée rétinopathie diabétique simple) débute par un stade de rétinopathie diabétique non proliférant qui entraîne une augmentation de la perméabilité capillaire, des micro anévrismes, des hémorragies, des exsudats, une ischémie maculaire et un œdème maculaire (épaississement rétinien lié à la diffusion du liquide à partir des capillaires).

- Moins grave
- On distingue quelque micro – anévrismes
- Quelque hémorragie

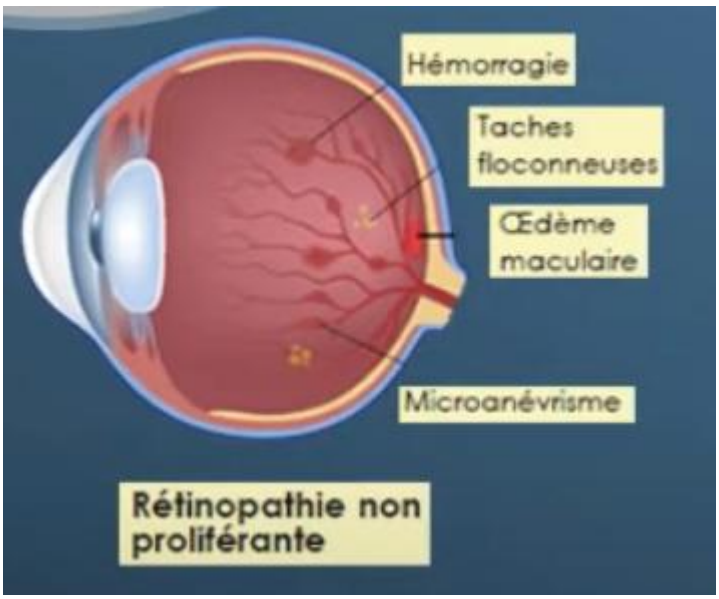


Figure I.4 : rétinopathie non proliférante

b. Rétinopathie proliférant :

Une rétinopathie proliférant se développe après la rétinopathie non proliférant et est plus sévère ; elle peut conduire à une hémorragie du vitré et au décollement de rétine par traction. La rétinopathie proliférant est caractérisée par le développement de néo vaisseaux pré rétiniens (néo vascularisation) sur l'interface vitro-rétinienne (vitrée) et qui peuvent s'étendre dans la cavité vitréenne et entraîner des hémorragies intra vitréennes. La néo vascularisation est souvent accompagnée par une prolifération fibreuse pré rétinienne, qui, en contact avec le corps vitré, peut se contracter, résultant en un décollement de la rétine tractionnel. La néo vascularisation peut également se développer dans le segment antérieur de l'œil sur l'iris ; la membrane néo vasculaire peut augmenter dans l'angle de la chambre antérieure de l'œil au bord de l'iris, ce qui peut provoquer un glaucome néo vasculaire. La perte visuelle due à la rétinopathie proliférant peut-être grave.

L'œdème maculaire cliniquement significatif peut être observé en présence d'une rétinopathie non proliférant ou proliférant et est la cause la plus fréquente de perte de vision due à la rétinopathie diabétique.

- On distingue de vastes hémorragies
- Des vastes territoires d'ischémie

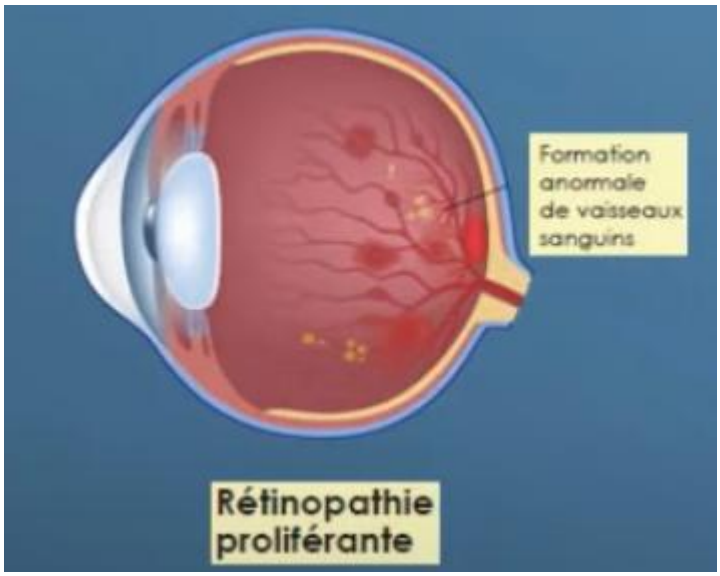


Figure I.5 : rétinopathie proliférante

I.4.4 Les symptômes :

Au début la rétinopathie ne s'accompagne souvent d'aucun symptôme visuel. La baisse de la vision est souvent progressive et lente. Mais elle peut être brutale lors de l'hémorragie du vitré dans le cas d'une rétinopathie diabétique proliférative

- Diminution de l'acuité visuelle
- Perte de la vision des détails
- Vision trouble

Champs visuelle parsemé de taches noires ou de points lumineux

I.4.5 Diagnostic :

Le diagnostic de la rétinopathie diabétique se déroule comme suit :

Examen la rétine au moyen d'un ophtalmoscope, qui éclaire d'une lumière vive le fond de l'œil et permet d'examiner de près les vaisseaux sanguins de la rétine

Examen de la rétine au moyen d'une angiographie a la fluorescéine qui permet de détecter les signes de saignements des vaisseaux sanguins. Cette technique permet de localiser avec précision les régions de la rétine susceptibles de saigner

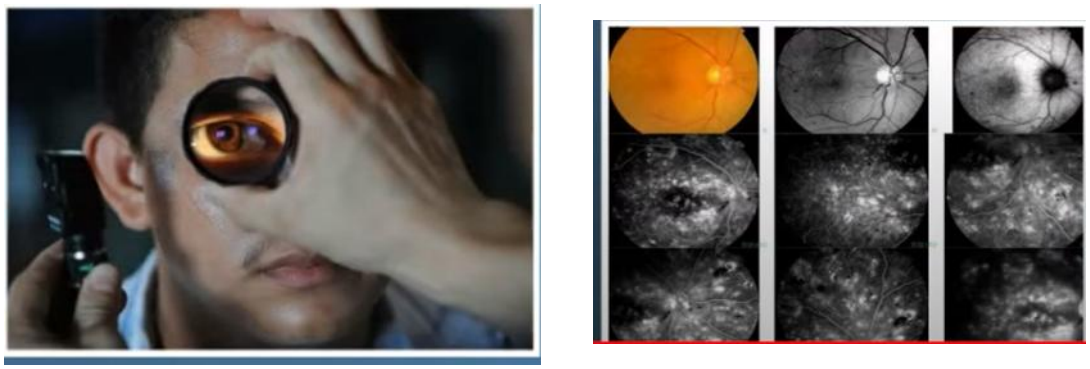


Figure I.6 : diagnostic de la rétinopathie diabétique

I.4.6 Le traitement :

Outre l'équilibration du diabète et de l'hypertension artérielle, le traitement de la rétinopathie diabétique consiste à traiter les zones pathologiques au laser. Certaines complications nécessitent une intervention chirurgicale.
Au stade précoce, la rétinopathie diabétique ne nécessite aucun traitement spécifique. Un équilibre strict du diabète ainsi que des autres facteurs de risque cardiovasculaires comme l'hypertension artérielle et le bilan lipidique sont en revanche indispensables pour limiter sa progression.

Au stade évolué, différents traitements sont réalisés

-Une photo coagulation au laser en plusieurs séances à intervalles variables en fonction de la gravité de la rétinopathie. Ce traitement vise à détruire les zones ischémiques non vascularisées de la rétine périphérique pour permettre une meilleure irrigation de la rétine centrale et éviter le développement de vaisseaux anormaux ou néo vaisseaux qui évolueraient vers un décollement de rétine ou une hémorragie intra oculaire.

- Le laser est aussi utilisé pour traiter un œdème maculaire localisé.

pour ces différents traitement des lasers de dernière génération multi spot micro pulse beaucoup plus rapide confortable et efficace

- Les injections intraoculaires de corticoïdes ou d'antiVEGF en cas d'œdème maculaire diffus. Des injections répétées sont souvent nécessaires.

- La chirurgie avec vitrectomie postérieure est indiquée en cas d'œdème maculaire tractionnel d'hémorragie intra vitréenne ou le décollement de rétine [5].

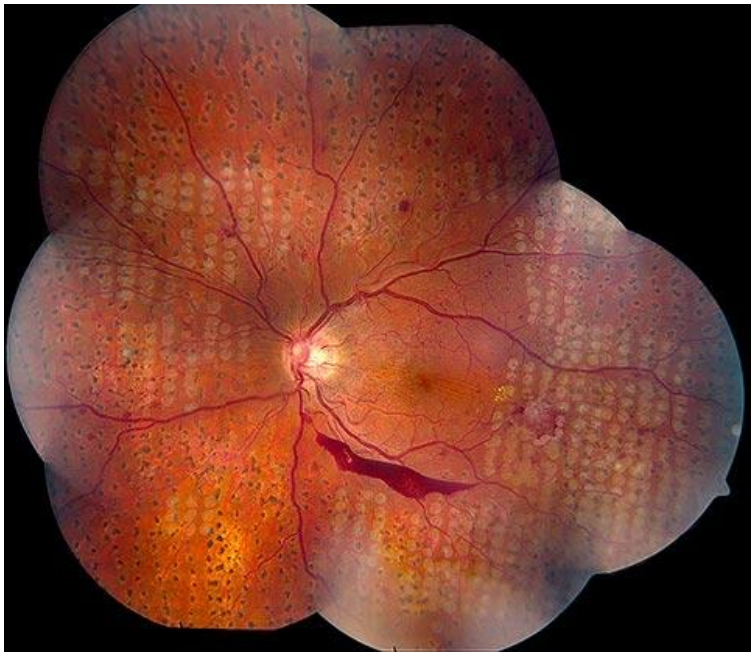


Figure I.7 : Rétinophotographie d'impacts de photo coagulation laser

I.5 Fond d'œil :

I.5.1 Définition :

Le fond d'œil est un examen ophtalmologique indolore qui permet de visualiser les structures profondes de l'œil. Il est utile pour le diagnostic de maladies ophtalmologiques mais aussi pour le diagnostic et le suivi d'atteintes de la rétine dues à des maladies générales comme le diabète.

I.5.2 Les trois différents techniques :

Réalisé à chaque visite de contrôle, l'examen simple du fond d'œil est rapide et indolore. Il peut être réalisé selon trois techniques :

La première technique consiste à projeter une petite lumière à travers votre pupille à l'aide d'un ophtalmoscope ;

Une deuxième technique, toujours sans contact avec l'œil, repose sur l'observation de votre rétine à l'aide d'une lentille d'examen convexe.

Ces deux techniques ne durent que quelques minutes.

Une troisième technique va être utilisée quand le spécialiste a besoin d'une vision complète (centrale et périphérique) de fond d'œil. L'examen va être un peu plus long (une trentaine de minutes) et votre pupille sera dilatée à l'aide d'un collyre. Une vingtaine de minutes après l'instillation du collyre, le médecin va utiliser un verre à trois miroirs posé directement sur votre œil de manière à bien examiner votre rétine. "L'examen avec dilatation de la pupille peut être un peu désagréable et éblouissant mais un collyre contenant un anesthésiant local va être utilisé de manière à ce que ce ne soit pas douloureux", explique le Pr Gilles Renard, directeur scientifique de la Société Française d'Ophtalmologie

Suite à la dilatation de la pupille, la vue sera floue et ne reviendra nette que dans les 2 à 4 heures qui suivent l'examen.

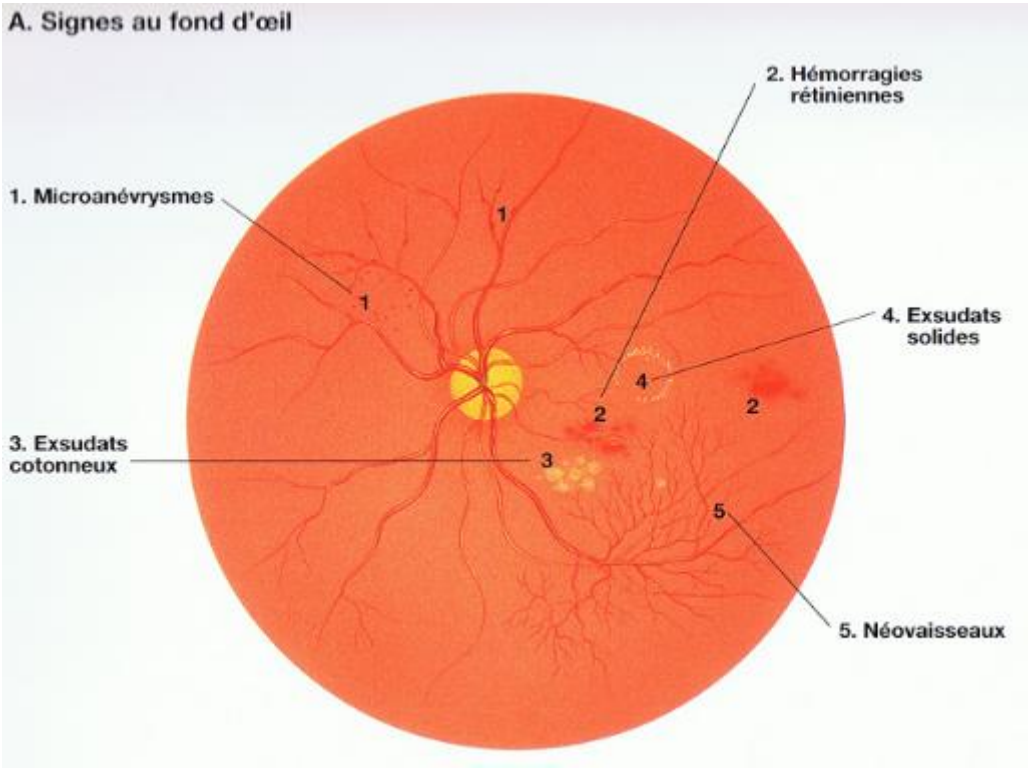


Figure I.8 : Signes fond d'œil

Nous utiliserons une grande quantité d'images de fond d'œil comme base de données, afin d'appliquer un algorithme d'intelligence artificielle, le deep Learning ou apprentissage profond, pour détecter et reconnaitre de la rétine affectée par la rétinopathie diabétique.

Nous introduisons dans la section qui suit, les notions d'intelligence artificielle et de deep Learning.

I.6 L'intelligence artificielle en ophtalmologie :

L'intelligence artificielle (IA) est une filière de l'informatique qui cherche à simuler l'intelligence humaine. Elle est généralement définie comme un groupe de capacités telles que la compréhension, l'apprentissage et le raisonnement pour prendre des décisions et résoudre des problèmes. C'est essentiellement la capacité d'un système informatisé à montrer des capacités cognitives.

Bien que l'IA ait une large application dans de nombreux domaines médicaux. L'ophtalmologie est la spécialité médicale qui utilise d'avantage l'intelligence artificielle, elle continue d'être une pionnière dans l'utilisation de cette intelligence pour détecter, diagnostiquer et traiter différentes pathologies oculaires telles que le glaucome, la dégénérescence maculaire liée à l'âge et la rétinopathie diabétique, ce qui peut faciliter au patient l'accès aux soins et potentiellement soulager les problèmes du système de santé.

Il est important que les médecins de premier recours se familiarisent avec les futures avancées de l'IA et le nouveau territoire inconnu vers lequel le monde de la médecine se dirige. L'objectif devrait être de trouver un équilibre bénéfique entre l'utilisation efficace de l'IA et le jugement humains des médecins de soins primaires.

I.7 Les réseaux de neurones :

I.7.1 Introduction :

Les réseaux de neurones, communément appelés des réseaux de neurones artificiels sont des imitations simples des fonctions d'un neurone dans le cerveau humain pour résoudre des problématiques d'apprentissage de la machine (Machine Learning).

I.7.2 L'historique :

Le concept des réseaux de neurones artificiels fut inventé en 1943 par deux chercheurs de l'Université de Chicago : le neuro-physicien **Warren McCullough**, et le mathématicien **Walter Pitts**. Dans un article publié dans le journal Brain Theory, les deux chercheurs présentent leur théorie selon laquelle l'activation de neurones est l'unité de base de l'activité cérébrale.

En 1957, le Perceptron fut inventé. Il s'agit du plus ancien algorithme de Machine Learning, conçu pour effectuer des tâches de reconnaissance de patterns complexes. C'est cet algorithme qui permettra plus tard aux machines d'apprendre à reconnaître des objets sur des images.

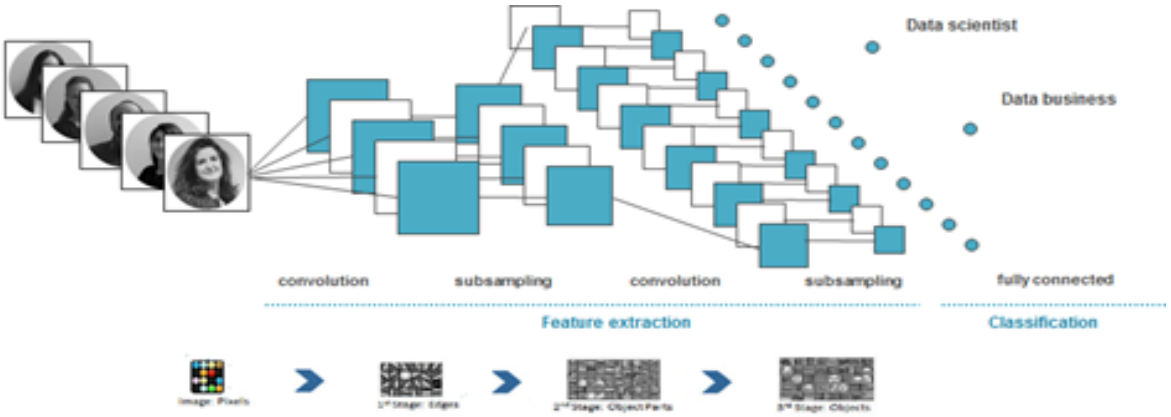


Figure I.9 : Réseaux neuronaux classiques pour reconnaissance d'image.

I.7.3 L'architecture d'un réseau neuronal :

Un réseau de neurones peut prendre des formes différentes selon l'objet de la donnée qu'il traite et selon sa complexité et la méthode de traitement de la donnée.

Les architectures ont leurs forces et faiblesses et peuvent être combinées pour optimiser les résultats. Le choix de l'architecture s'avère ainsi crucial et il est déterminé principalement par l'objectif.

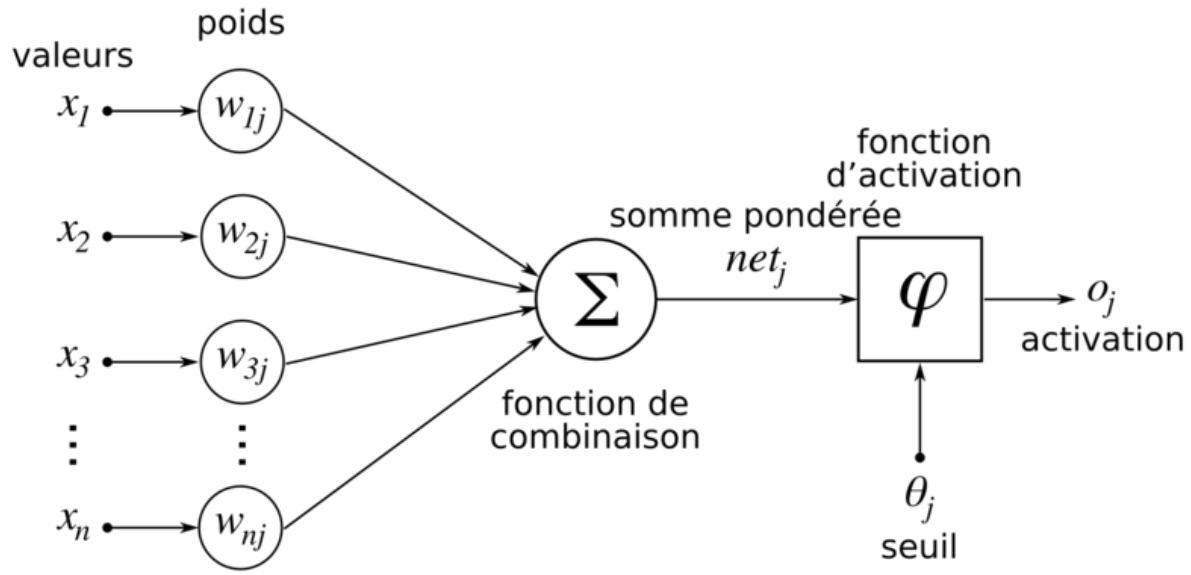


Figure I.10 : L'architecteur de réseau de neurone

En règle générale, un réseau de neurones repose sur un grand nombre de processeurs opérant en parallèle et organisés en tiers. Le premier tiers reçoit les entrées d'informations brutes, un peu comme les nerfs optiques de l'être humain lorsqu'il traite des signaux visuels.

Par la suite, chaque tiers reçoit les sorties d'informations du tiers précédent. On retrouve le même processus chez l'Homme, lorsque les neurones reçoivent des signaux en provenance des neurones proches du nerf optique. Le dernier tiers, quant à lui, produit les résultats du système [6].

I.7.4 Le neurone formel :

Le neurone formel est comme son nom l'indique un neurone. Contrairement au neurone biologique, il n'est pas, justement biologique et pour cause : c'est un neurone mathématique. C'est lui qui est utilisé dans le fonctionnement de l'intelligence artificielle, donc dans le domaine informatique.et qu'il a un certain nombre de points communs avec son homologue nerveux, et bien sûr la façon dont il fonctionne.

a. Fonctionnement de neurone formel :

$$in = \sum_{i=0}^{k} x_i \times w_i$$

- k le nombre d'entrées.
- xi les entrées, avec i variant entre 1 et k (x1 sera donc la première information, x2 la deuxième, et xk la dernière).

•De la même façon, wi les poids synaptiques correspondants aux entrées (w1 sera le poids de x1 et wk celui de xk).

•L'information x0 valant -1, dont le poids est w0

Le neurone formel a besoin d'informations, appelées entrées, et il renvoie une autre information, appelée sortie. Il peut y avoir plusieurs entrées, mais seulement une seule sortie. Toutes ces valeurs sont binaires, égales à 0 ou 1.

Chaque "synapse" du neurone formel possède un poids, c'est-à-dire une importance plus ou moins élevée par rapport aux autres. Une information passant par une synapse au poids élevé sera donc plus importance qu'une provenant d'une synapse au poids faible.

De plus, une information égale à -1 est toujours soumise au neurone, avec un poids particulier

I.7.5 Le perceptron :

Le perceptron est l'un des tout premiers algorithmes de Machine Learning, et le réseau de neurones artificiels le plus simple.

Dans le domaine du Machine Learning, le perceptron est un algorithme d'apprentissage supervisé de classifies binaires (séparant deux classes). Il s'agit alors d'un type de classifier linéaire, et du type de réseau de neurones artificiels le plus simple.

Dans le cas d'un classifier linéaire, les données d'entraînement doivent être classifiées en catégories correspondantes de sorte que si des classifications sont appliquées à deux catégories, toutes les données d'entraînement doivent être rangées dans ces deux catégories. Dans le cas d'un classifier binaire, il ne peut y avoir que deux catégories de classification.

a. Perceptron multicouche :

Le perceptron multicouche (multilayer perceptron MLP) organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau à propagation directe (feedforward). Chaque couche est constituée d'un nombre variable de neurones, les neurones de la dernière couche (dite « de sortie ») étant les sorties du système global.

Le Perceptron multicouche se caractérise par :

- Il est basé sur le modèle du Perceptron
- Il a plusieurs couches de neurones liées entre elles
- Chaque couche a un ou plusieurs neurones

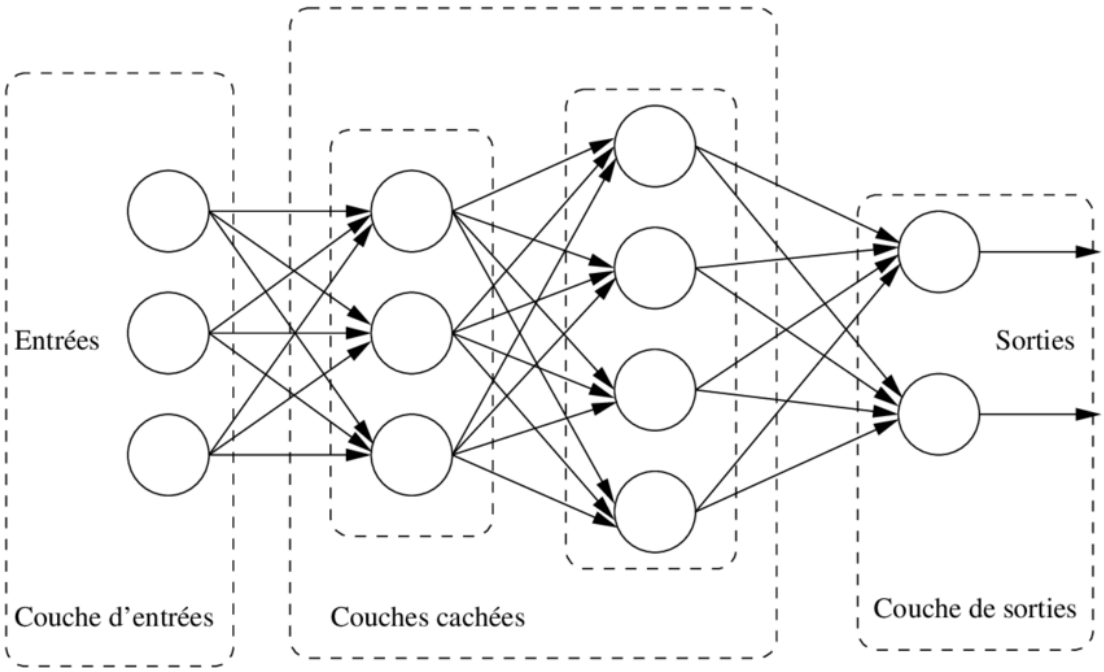


Figure I.11 : architecture de perceptron multicouche.

I.7.6 L'apprentissage :

L'apprentissage, consiste à calculer les paramètres de telle manière que les sorties du réseau de neurones soient, pour les exemples utilisés lors de l'apprentissage, aussi proches que possible des sorties « désirées ».

a. Les types d'apprentissage :

On a 4 types d'apprentissage

- **Apprentissage supervisé :**

L'apprentissage supervisé consiste en l'entraînement d'une machine en utilisant des données labellisées. C'est-à-dire des données qui ont déjà été étiquetées avec le bon label (classe, valeur continue…).

Cet apprentissage sur des données ayant déjà la ''réponse correcte'' permet de prédire par la suite le label de données nouvelles non étiquetées.

- **Apprentissage non supervisé :**

La différence lorsqu'on parle du type d'apprentissage non-supervisé, c'est que les réponses que l'on cherche à prédire ne sont pas disponibles dans les jeux de données. L'algorithme utilise un jeu de données non étiquetées. On demande alors à la machine de créer ses propres réponses. Elle propose ainsi des réponses à partir d'analyses et de groupement de données. Pour y voir plus clair.

- **Apprentissage semi supervisé :**

L'apprentissage semi-supervisé est une classe de techniques d'apprentissage automatique qui utilise un ensemble de données étiquetées et non étiquetées. Il se situe ainsi entre l'apprentissage supervisé qui n'utilise que des données étiquetées et l'apprentissage non

supervisé qui n'utilise que des données non étiquetées. Il a été démontré que l'utilisation de données non étiquetées, en combinaison avec des données étiquetées, permet d'améliorer significativement la qualité de l'apprentissage.

- **Apprentissage renforcé :**

L'apprentissage par renforcement consiste, pour un agent autonome (robot, etc.), à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps. L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative. L'agent cherche, au travers d'expériences itérées, un comportement décisionnel (appelé stratégie ou politique, et qui est une fonction associant à l'état courant l'action à exécuter) optimal, en ce sens qu'il maximise la somme des récompenses au cours du temps.

I.7.7 La fonction d'activation :

La fonction d'activation est une formule mathématique (algorithme) activée dans certaines circonstances. Lorsque les neurones calculent la somme pondérée des valeurs d'entrée + le biais, elles sont transmises à la fonction d'activation, qui vérifie si la valeur calculée est supérieure au seuil requis.

Si la valeur calculée est supérieure au seuil requis, la fonction d'activation est activée et une valeur de sortie est calculée.

Cette valeur de sortie est ensuite transmise aux couches suivantes ou précédentes (en fonction de la complexité du réseau), ce qui peut aider les réseaux de neurones à modifier le poids de leurs neurones.

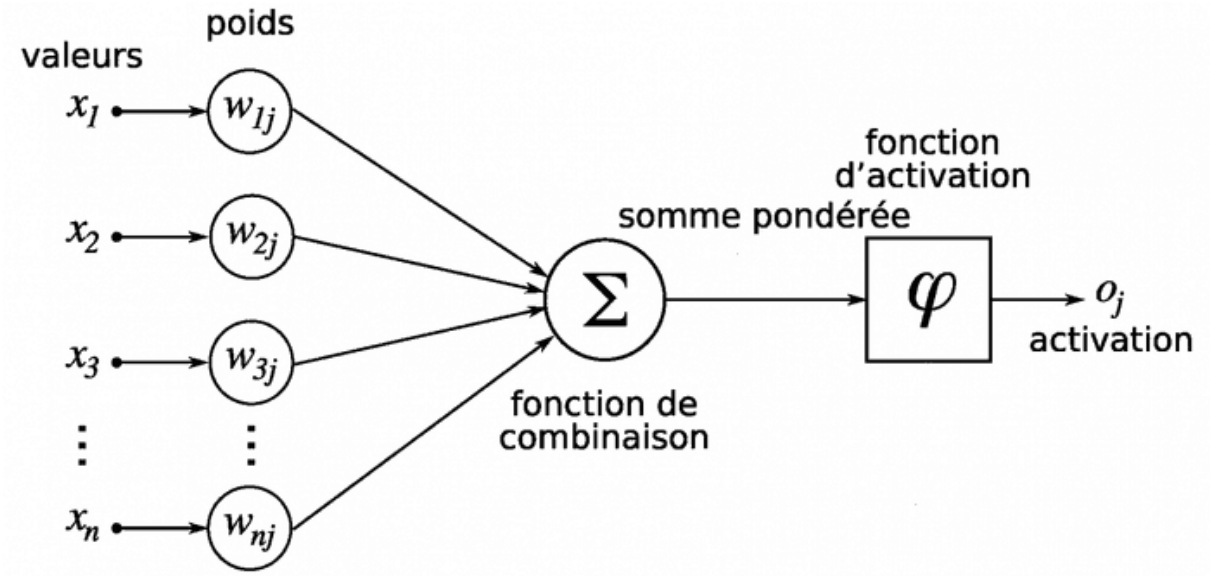


Figure I.12 : illustration de la fonction d'activation.

Les fonctions d'activation introduisent la non-linéarité dans les réseaux de neurones, nécessaire pour résoudre des problèmes complexes.

Si nous traçons les sorties non linéaires produites par les fonctions d'activation, nous obtiendrons une courbure. La pente de la courbe est utilisée pour calculer le gradient. Et le gradient nous aide à comprendre le taux de changement et les relations entre les variables.

À partir des relations, les algorithmes sont optimisés et les poids sont mis à jour.

a. Les types de fonction d'activation :

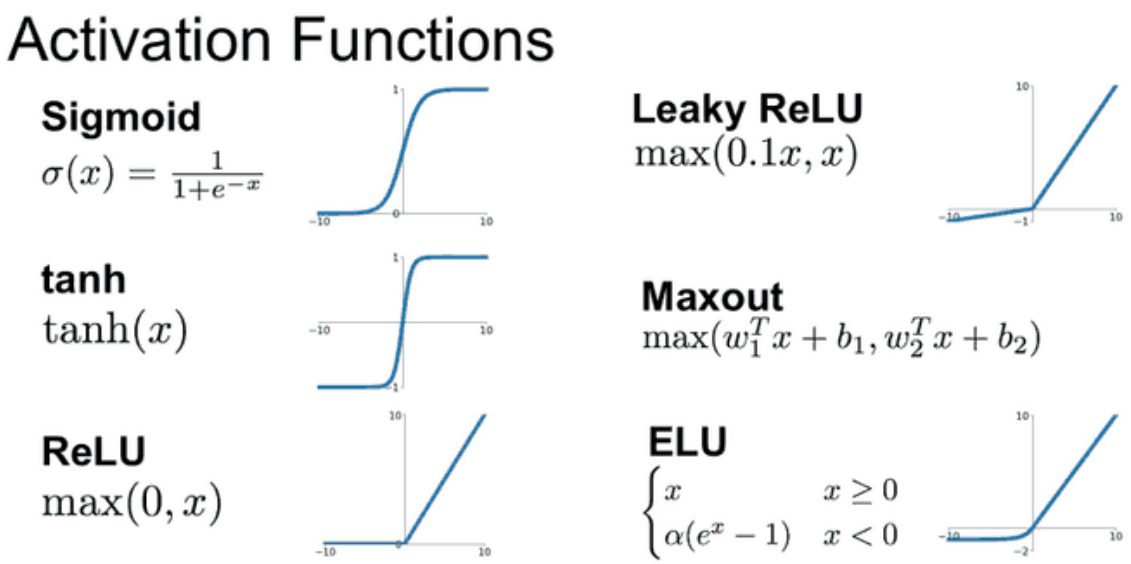
Type de fonction d'activation

Unité linéaire rectifiée (ReLu) : Cette fonction converge plus rapidement, optimise et produit la valeur souhaitée plus rapidement. C'est de loin la fonction d'activation la plus populaire utilisée dans les couches cachées.

Fonctions de tangente hyperbolique (tanh) : Il s'agit d'une fonction supérieure comparée à Sigmoïde. Cependant, elle rend moins bien compte des relations et elle est plus lente à converger.

Sigmoïde : produit une courbe en forme de S. Bien que de nature non linéaire, il ne tient toutefois pas compte des légères variations des entrées, ce qui entraîne des résultats similaires.

Figure I.13: schéma synoptique des types de la fonction d'activation.



I.8 Réseau de neurones convolutifs :

Les réseaux convolutifs sont une forme particulière de réseau neuronal multicouches dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères.

Leur conception suit la découverte de mécanismes visuels dans les organismes vivants. Ces réseaux de neurones artificiels sont capables de catégoriser les informations des plus simples aux plus complexes. Ils consistent en un empilage multicouche de neurones, des fonctions mathématiques à plusieurs paramètres ajustables, qui prétraitent de petites quantités d'informations. Les réseaux convolutifs sont caractérisé par leurs premières couches convolutionelles (généralement une à trois). Une couche convolutive, est basée comme son nom l'indique sur le principe mathématique de convolution, et cherche à repérer la présence d'un motif (dans un signal ou dans une image par exemple).

Pour une image, la première couche convolutionelle peut détecter les contours des objets (par exemple un cercle), la seconde couche convolutionelle peut combiner les contours en objets (par exemple une roue), et les couches suivantes (non nécessairement convolutionelles) peuvent utiliser ces informations pour distinguer une voiture d'une moto. Une phase d'apprentissage sur des objets connus permet de trouver les meilleurs paramètres en montrant par exemple à la machine des milliers d'images d'un chien, d'une voiture ou d'un sport… L'un des enjeux est de trouver des méthodes pour ajuster ces paramètres le plus rapidement et le plus efficacement possible. Les réseaux neuronaux convolutifs ont de nombreuses applications dans la reconnaissance d'images, de vidéos ou le traitement du langage naturel.

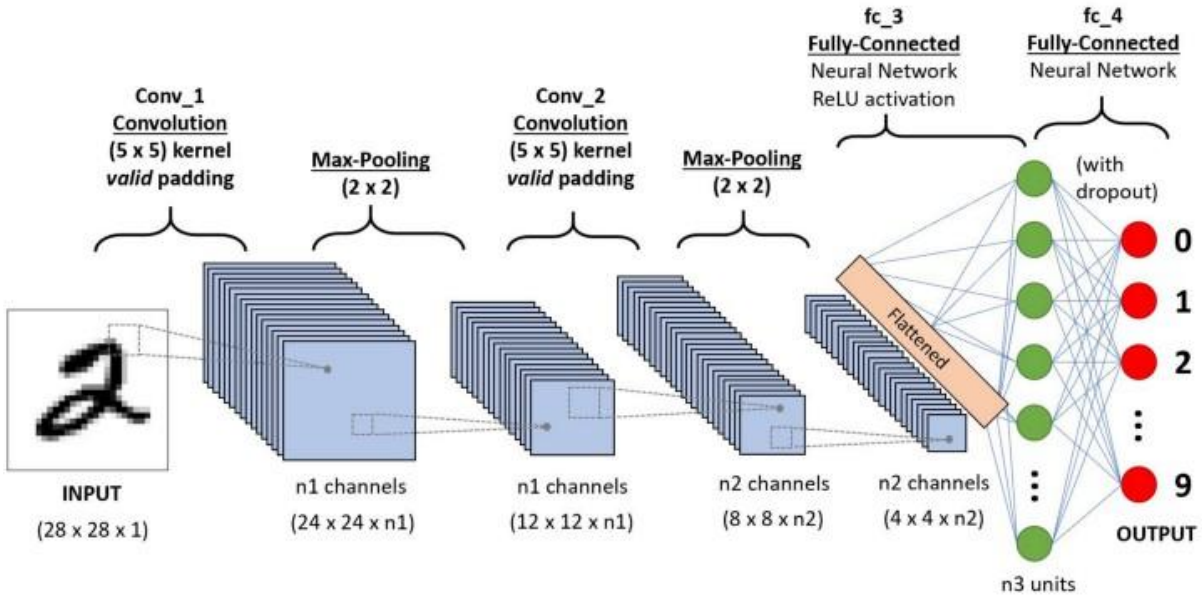


Figure I.14: Réseau de neurones convolutifs.

I.8.1 La convolution :

La convolution est un outil mathématique simple qui est très largement utilisé pour le traitement d'image, ce qui explique que les réseaux de neurones à convolution soient particulièrement bien adaptés à la reconnaissance d'image.

La convolution agit comme un filtrage. On définit une taille de fenêtre qui va défiler à travers toute l'image. Au tout début de la convolution, la fenêtre sera positionnée tout en haut à gauche de l'image puis elle va se décaler d'un certain nombre de cases (c'est ce que l'on appelle le pas) vers la droite et

lorsqu'elle arrivera au bout de l'image, elle se décalera d'un pas vers le bas ainsi de-suite jusqu'à ce que le filtre est parcourue la totalité de l'image :

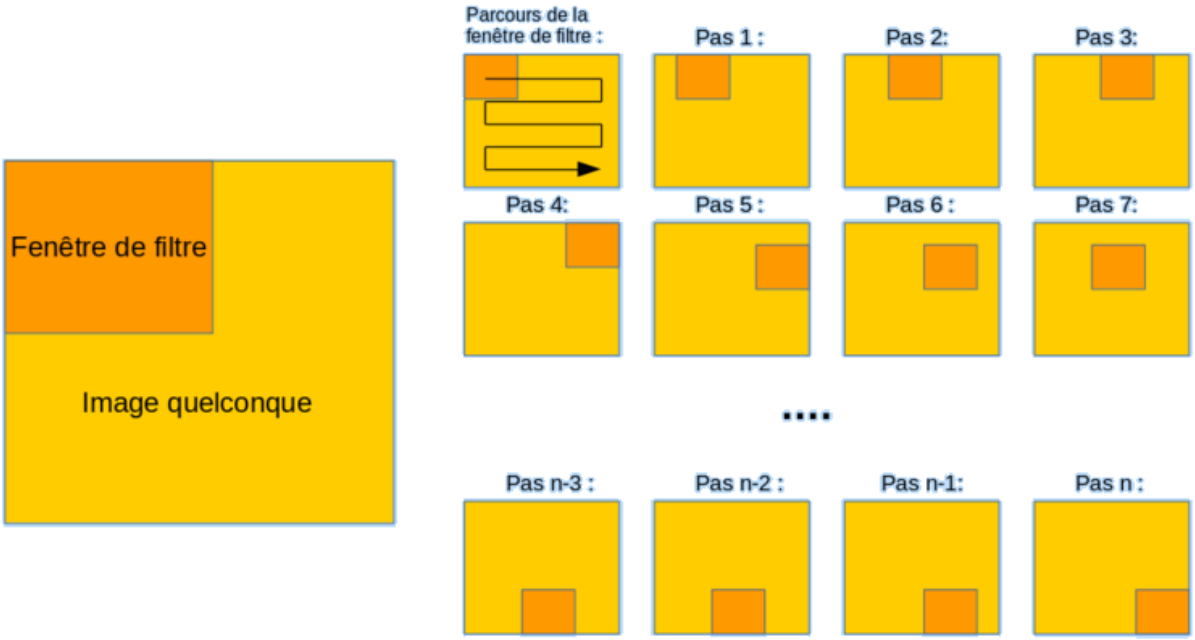


Figure I.15 : Schéma du parcours de la fenêtre de filtre sur l'image.

Le but est de se servir des valeurs présentes dans le filtre à chaque pas. Par exemple si l'on définit une fenêtre 3 par 3, cela représentera 9 cases du tableau (c'est à dire 9 pixels). La convolution va effectuer une opération avec ces 9 pixels. Il peut s'agir de n'importe quelle opération, par exemple on extrait la valeur la plus grande (soit le pixel avec la plus grande valeur) [7].

I.8.2 Le pooling :

La couche de pooling (en anglais *pooling layer*) (POOL) est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle. En particulier, les types de pooling les plus populaires sont le max et l'average pooling, où les valeurs maximales et moyennes sont prises, respectivement [8].

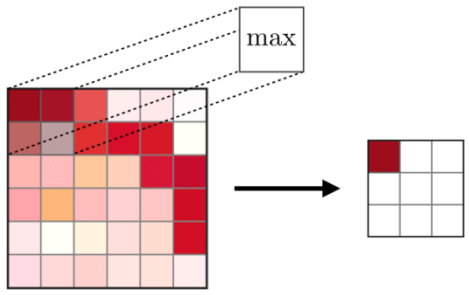
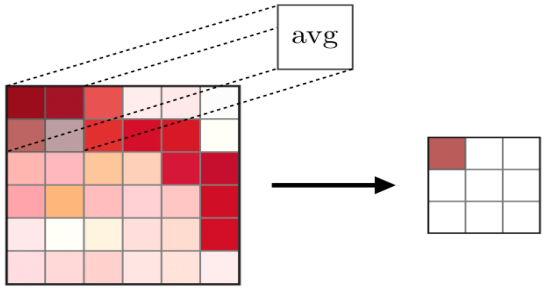
Max pooling	**Moyenne pooling**
Chaque opération de pooling sélectionne la valeur maximale de la surface	Chaque opération de pooling sélectionne la valeur moyenne de la surface
	
• Garde les caractéristiques détectées • Plus communément utilisé	• Sous-échantillonne la *feature map* • Utilisé dans LeNet

Figure 1 : max et moyenne pooling.

I.8.3 Fully connected

La couche de fully connected (en anglais *fully connected layer*) (FC) s'applique sur une entrée préalablement aplatie où chaque entrée est connectée à tous les neurones. Les couches de fully connected sont typiquement présentes à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe [9].

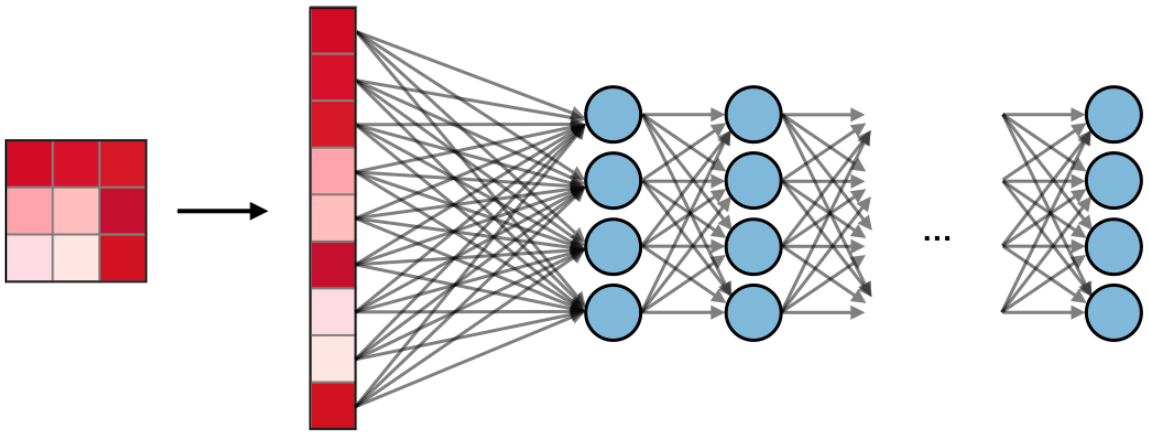


Figure I.16 :fully connected

I.9 Le modèle VGG :

VGG est un algorithme connu en Computer Vision très souvent utilisé par transfert d'apprentissage pour éviter d'avoir à le réentraîner et résoudre des problématiques proches sur lesquelles VGG a déjà été entraîné

I.9.1 VGG16 :

VGG-16 est constitué de plusieurs couches, dont 13 couches de convolution et 3 *fully-connected*. Il doit donc apprendre les poids de 16 couches.

Il prend en entrée une image en couleurs de taille 224 × 224 px et la classifie dans une des 1000 classes. Il renvoie donc un vecteur de taille 1000, qui contient les probabilités d'appartenance à chacune des classes.

Figure I.17 : architecture de VGG16

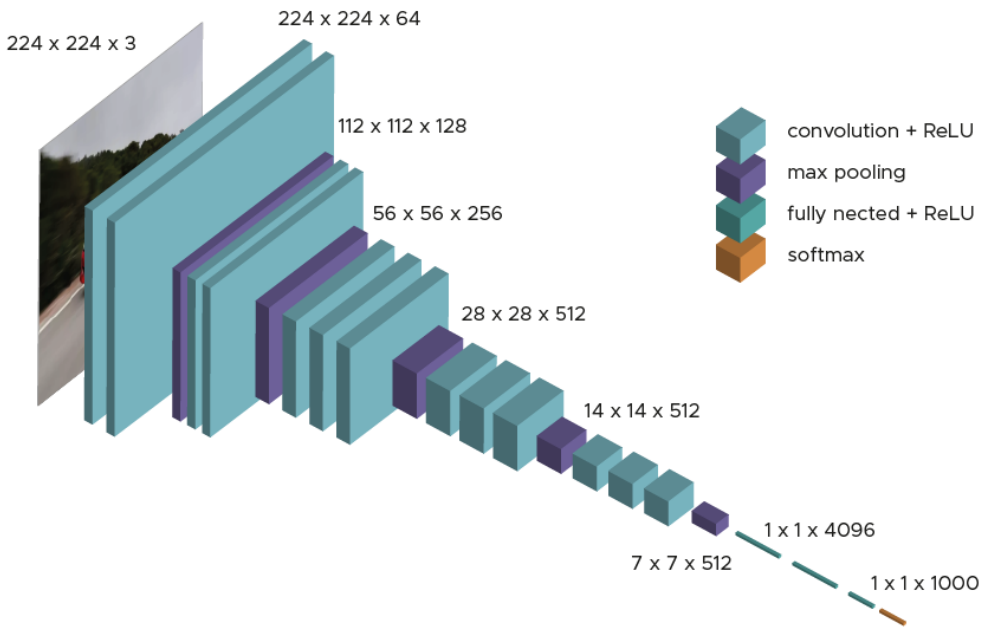


Figure I.18 : Représentation 3D de l'architecture de VGG-16

Chaque couche de convolution utilise des filtres en couleurs de taille 3 ×3 px, déplacés avec un pas de 1 pixel. Le zero-padding vaut 1 pixel afin que les volumes en entrée aient les mêmes dimensions en sortie. Le nombre de filtres varie selon le "bloc" dans lequel la couche se trouve. De plus, un paramètre de biais est introduit dans le produit de convolution pour chaque filtre.

Chaque couche de convolution a pour fonction d'activation une ReLU. Autrement dit, il y a toujours une couche de correction ReLU après une couche de convolution.

L'opération de pooling est réalisée avec des cellules de taille 2 ×2 px et un pas de 2 px – les cellules ne se chevauchent donc pas.

Les deux premières couches fully-connected calculent chacune un vecteur de taille 4096, et sont chacune suivies d'une couche ReLU. La dernière renvoie le vecteur de probabilités de taille 1000 (le nombre de classes) en appliquant la fonction softmax. De plus, ces trois couches utilisent un paramètre de biais pour chaque élément du vecteur en sortie.

I.9.2 VGG19 :

VGG-19 est un réseau de neurones convolutifs de 19 couches de profondeur. Il est possible de charger une version pré-entraînée du réseau entraîné sur plus d'un million d'images de la base de données ImageNet. Le réseau pré-entraîné peut classer les images en 1000 catégories d'objets, telles que le clavier, la souris, le crayon et de nombreux animaux. En conséquence, le réseau a appris de riches représentations de caractéristiques pour un large éventail d'images. Le réseau a une taille d'entrée d'image de 224 par 224.

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**
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**
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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure I.19 : Configuration réelle des réseaux

I.10 Le modèle ResNet :

Les avantages de ces réseaux sont leur capacité à réduire la fuite de gradient dans les couches inférieures du réseau, et la possibilité de leur mise à l'échelle en profondeur. Cette dernière permet l'ajustement de l'architecture à la tâche de classification à accomplir. Bien

que coûteux en ressources de calcul, ils obtiennent de très bonnes performances en classification sur tous les niveaux de détails de l'image, même les plus fins comme les nodules pulmonaires ou les hémorragies iintracrâniennes.

De par leur architecture en module, ils appartiennent à la famille de modèles appelée "Network in Network". Ces réseaux sont un ensemble de micro-architecture ou de blocs de traitement. Cet ensemble permet d'obtenir la macro-architecture (c'est-à-dire le réseau final lui-même).

I.11 Le modèle inception V3 :

Inception v3 est un modèle de reconnaissance d'images couramment utilisé qui a démontré, sur l'ensemble de données ImageNet, une justesse supérieure à 78,1 %. Ce modèle est l'aboutissement de nombreuses idées développées par plusieurs chercheurs au fil des ans. Il est basé sur l'article originel Rethinking the Inception Architecture for Computer Vision (Repenser l'architecture Inception pour la vision par ordinateur) de Szegedy, et. al.

Le modèle lui-même est constitué de composants de base symétriques et asymétriques incluant convolutions, pooling moyen, pooling maximal, concaténations, abandons et couches entièrement connectées. La normalisation par lots (batchnorm) est amplement utilisée dans le modèle et appliqué aux entrées d'activation. La perte est calculée via Softmax.

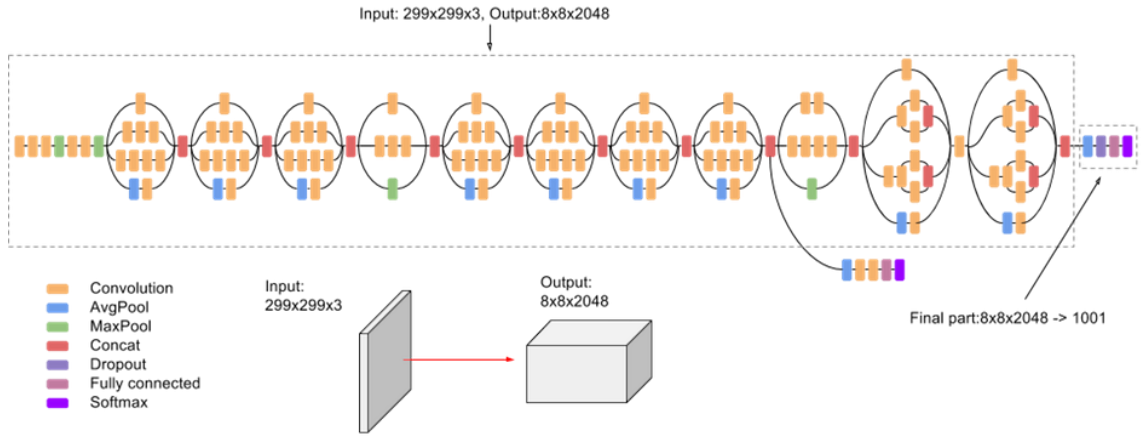


Figure I.20 : diagramme général du modèle Inception V3

I.12 Réseaux de Neurones Covolutifs Régionaux (R-CNN) :

Le R-CNN est une architecture de détection d'objet. Le R-CNN commence par extraire des régions intéressantes de l'image, puis il utilise ces régions comme données d'entrée pour un CNN. Cette s´réparation en régions permet de d´détecter plusieurs objets de plusieurs classes différentes dans une même image. Cette solution proposée par Girshick et al., 2013 a permis d'améliorer la précision des modèles de d´détection.

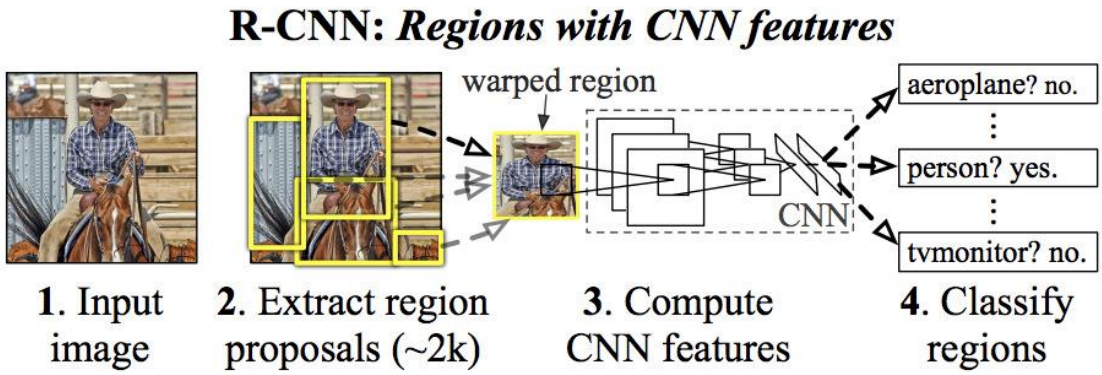


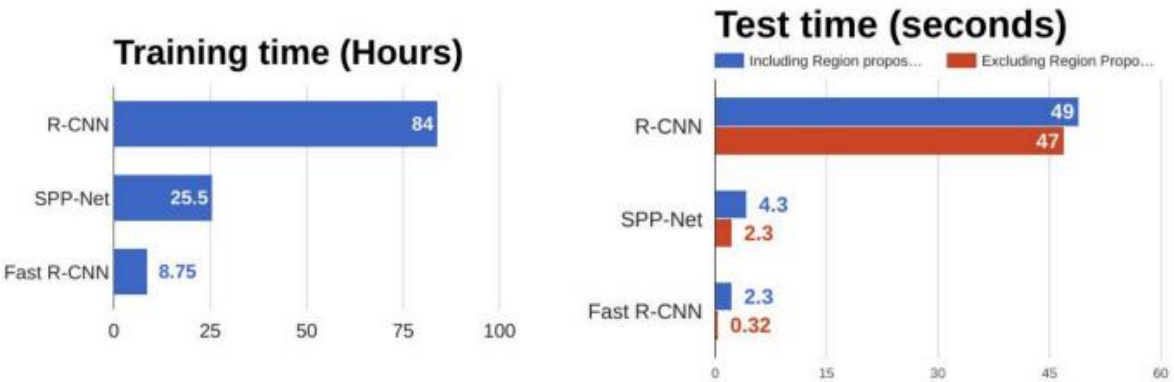
Figure I.21 : l'architecture de R-CNN

Les régions sont extraites grâce `a une recherche sélective proposée par Uijlings, van de Sande, Gevers et Smeulders, 2013. Celle-ci utilise la structure de l'image et plusieurs techniques de partitionnement pour récupérer toutes les régions intéressantes possibles.

I.12.1 Fast R-CNN :

Le même auteur de l'article précédent (R-CNN) résolu certains des inconvénients de R-CNN pour construire un algorithme de détection d'objet plus rapide et il a été appelé Fast R-CNN. L'approche est similaire à l'algorithme R-CNN. Mais, au lieu d'alimenter les propositions de région à CNN, nous alimentons l'image d'entrée à CNN pour générer une carte de caractéristiques convolutionnelle. A partir de la carte des caractéristiques convolutionnelles, nous identifions la région des propositions et les déformons en carrés et en utilisant une couche de mise en commun RoI nous les remodèlons en une taille fixe afin qu'elles puissent être introduites dans une couche entièrement connectée. A partir du vecteur de fonctionnalité RoI, nous utilisons une couche softmax pour prédire la classe de la région proposée ainsi que les valeurs de décalage pour la boîte de délimitation.

La raison pour laquelle « Fast R-CNN » est plus rapide que R-CNN est qu'il n'est pas nécessaire de transmettre 2000 propositions de région au réseau neuronal convolutionnel à chaque fois. Au lieu de cela, l'opération de convolution n'est effectuée qu'une seule fois par image et une carte de fonctionnalités est générée à partir de celle-ci.



I.12.2 Faster R-CNN :

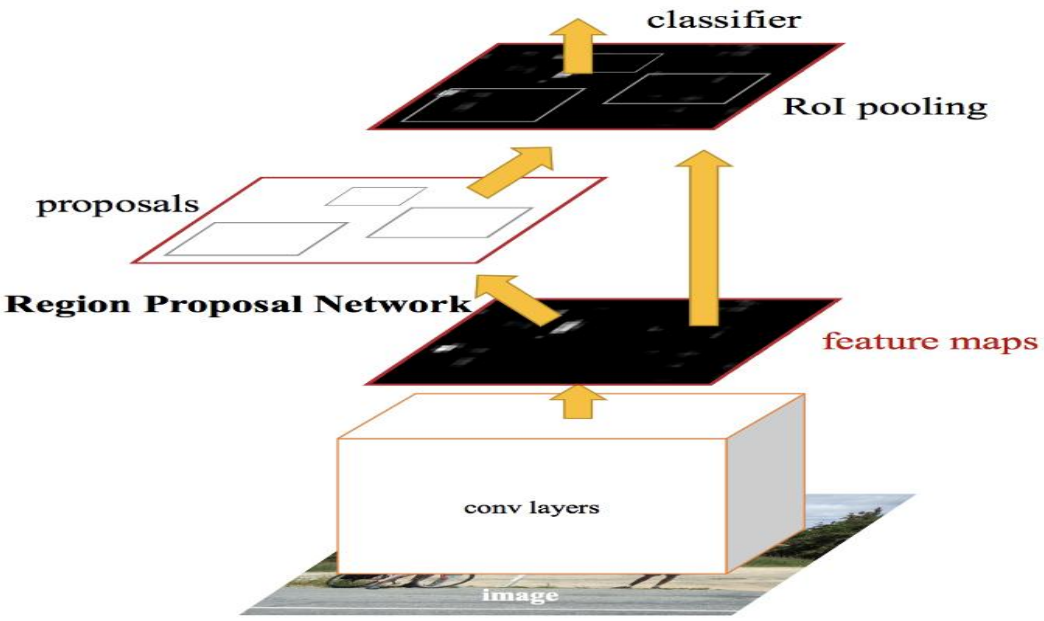


Figure I.22 : l'architecture « faster R-CNN »

Les deux algorithmes ci-dessus (R-CNN & Fast R-CNN) utilisent la recherche sélective pour trouver les propositions de région. La recherche sélective est un processus lent et long qui affecte les performances du réseau. Par conséquent, Shaoqing Ren et al. Ont mis au point un algorithme de détection d'objets qui élimine l'algorithme de recherche sélective et permet au réseau d'apprendre les propositions de régions.

Semblable à Fast R-CNN, l'image est fournie comme une entrée à un réseau convolutionnel qui fournit une carte de caractéristiques convolutionnelle. Au lieu d'utiliser un algorithme de recherche sélective sur la carte des caractéristiques pour identifier les propositions de région, un réseau distinct est utilisé pour prédire les propositions de région. Les propositions de régions prévues sont ensuite remodelées à l'aide d'une couche de mise en commun RoI qui est ensuite utilisée pour classer l'image dans la région proposée et prédire les valeurs de décalage pour les boîtes limitatives.

I.12.3 Mask-R-CNN :

Mask R-CNN, ou Mask RCNN, est un réseau neuronal convolutionnel (CNN) et un état de l'art en termes de segmentation d'image et de segmentation d'instance. Mask R-CNN a été développé en plus de Faster R-CNN, un réseau de neurones convolutionnels basé sur la région. La première étape pour comprendre comment fonctionne Mask R-CNN nécessite une compréhension du concept de segmentation d'image. Dans La tâche de vision par ordinateur La segmentation d'image est le processus de partitionnement d'une image numérique en plusieurs segments (ensembles de pixels, également appelés objets d'image). Cette segmentation permet de localiser des objets et des limites (lignes, courbes, etc.) [10].

Il existe 2 principaux types de segmentation d'image qui relèvent de Mask R-CNN :

-Segmentation sémantique

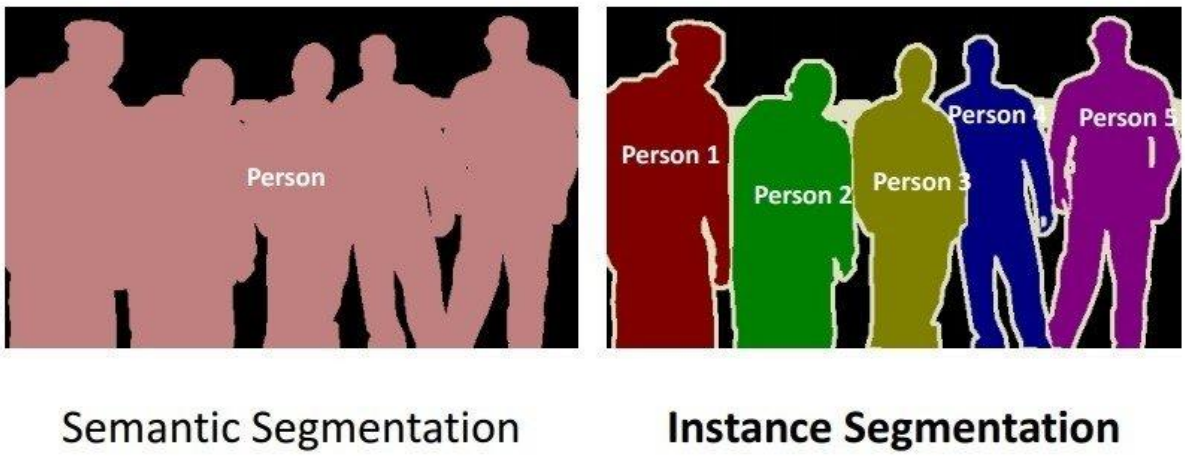
- La segmentation d'instance

➢ Segmentation sémantique :

La segmentation sémantique classe chaque pixel dans un ensemble fixe de catégories sans différencier les instances d'objets. En d'autres termes, la segmentation sémantique traite de l'identification/classification d'objets similaires en une seule classe à partir du niveau des pixels. Comme le montre l'image ci-dessus, tous les objets ont été classés comme une seule entité (personne). La segmentation sémantique est également connue sous le nom de segmentation d'arrière-plan parce qu'elle sépare les sujets de l'image de l'arrière-plan.

➢ La segmentation d'instance :

La segmentation d'instance, ou reconnaissance d'instance, traite de la détection correcte de tous les objets d'une image tout en segmentant précisément chaque instance. C'est donc la combinaison de la détection d'objets, de la localisation d'objets et de la classification d'objets. En d'autres termes, ce type de segmentation va plus loin pour donner une distinction claire entre chaque objet classé comme des instances similaires. Comme le montre l'exemple d'image ci-dessus, pour la segmentation d'instance, tous les objets sont des personnes, mais ce processus de segmentation sépare chaque personne en une seule entité. La segmentation sémantique est aussi appelée segmentation de premier plan parce qu'elle accentue les sujets de l'image au lieu de l'arrière-plan.



FigureI.23 : Différences entre la segmentation sémantique et la segmentation d'instance.

a. La fonction de Mask R-CNN :

Mask R-CNN a été construit avec Faster R-CNN. Alors que Faster R-CNN a 2 sorties pour chaque objet candidat, une étiquette de classe et un décalage de boîte de délimitation, Mask R-CNN est l'ajout d'une troisième branche qui affiche le masque objet. La sortie de masque supplémentaire est distincte des sorties de classe et de boîte, nécessitant l'extraction d'une disposition spatiale beaucoup plus fine d'un objet. Masque R-CNN est une

extension de Faster R-CNN et fonctionne en ajoutant une branche pour prédire un masque d'objet (Région d'Intérêt) en parallèle avec la branche existante pour la reconnaissance de boîte limitative.

b. Avantages du masque R-CNN :

- ✓ **Simplicité :** Le masque R-CNN est simple à former.
- ✓ **Performance** : Mask R-CNN surpasse toutes les entrées de modèle unique existantes pour chaque tâche.
- ✓ **Efficacité :** La méthode est très efficace et n'ajoute qu'une petite surcharge à R-CNN plus rapide.
- ✓ **Flexibilité :** Masque R-CNN est facile à généraliser à d'autres tâches.

L'élément clé de Mask R-CNN est l'alignement pixel-à-pixel, qui est la pièce manquante principale de Fast/Faster R-CNN. Masque R-CNN adopte la même procédure en deux étapes avec une première étape identique (qui est RPN). Dans la deuxième étape, en parallèle de la prédiction de la classe et du décalage de la boîte, Mask R-CNN affiche également un masque binaire pour chaque RoI. Cela contraste avec les systèmes les plus récents, où la classification dépend des prévisions du masque. En outre, Mask R-CNN est simple à mettre en œuvre et à former compte tenu du cadre R-CNN plus rapide, qui facilite un large éventail de conceptions d'architecture flexibles. De plus, la branche masque n'ajoute qu'une petite surcharge informatique, permettant un système rapide et une expérimentation rapide.

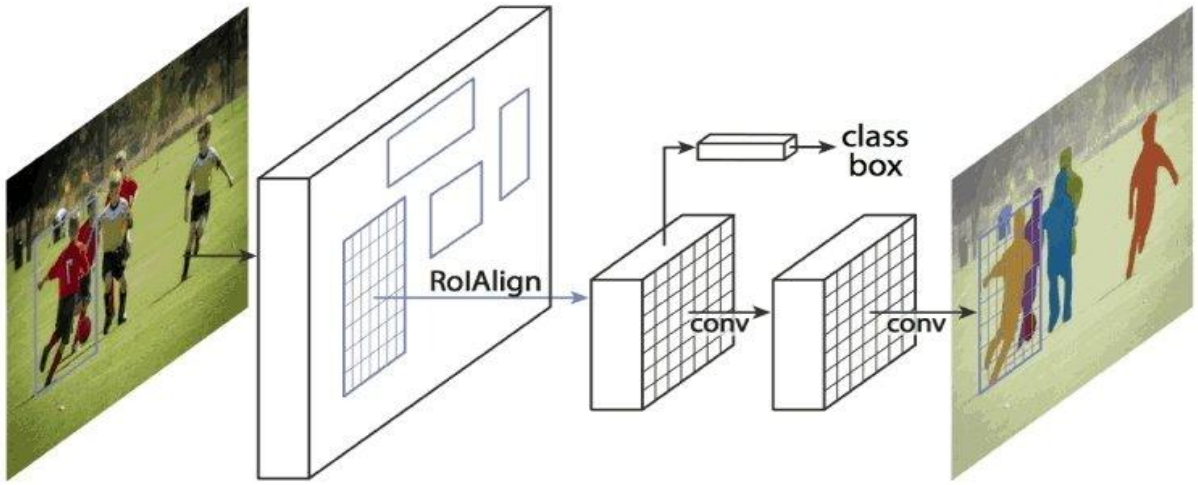


Figure I.24: Le cadre Mask R-CNN pour la segmentation d'instance

I.13 Conclusion :

Dans ce chapitre nous comprenons ce qu'est la rétinopathie diabétique et comment la détecter en utilisant l'imagerie médicale et en même temps nous voyons les différents types de réseaux neuronaux, leur fonctionnement et leurs propriétés.

Dans le prochain chapitre on va présenter les outils utilisés dans notre travail ainsi que les étapes d'installation et de manipulation.

Chapitre II

Développement de la méthode de détection et reconnaissance de la rétinopathie diabétique par les algorithmes de deep Learning

II.1. Introduction :

Dans ce chapitre, nous allons voir et étudier les différentes plateformes et outils utilisés dans notre projet (Google colab, Jupiter notebooks, python et ces bibliothèques), les méthodes et techniques appliquées (Transfer Learning, data augmentation), l'architecture modifiée de chacun de nos réseaux (VGG, Alex net, inception_v3 et U-Net) et les paramètres nécessaires pour la configuration de leur entrainement et apprentissage.

II.2. Etapes de développement :

Pour commencer le travail, nous devons d'abord connaitre les étapes :

Dans un premier temps nous avons choisi la base de données à partir de la plateforme open source Kaggle.

Après avoir choisi la bonne base de données, elle doit être divisée en dossiers, chaque dossier contenant des images de la rétine.

Ensuite, nous appliquons la méthode d'augmentation de données à nos images.

Importez ensuite la base de données dans Google Drive afin que nous puissions l'utiliser dans Google colab où nous ferons tout le travail avec les algorithmes pour pouvoir faire l'apprentissage et la classification des images.

Ensuite dans la deuxième partie, nous avons utilisé l'architecture U-NET pour détecter la rétinopathie diabétique

Nous allons montrer ces étapes dans l'illustration ci-dessous :

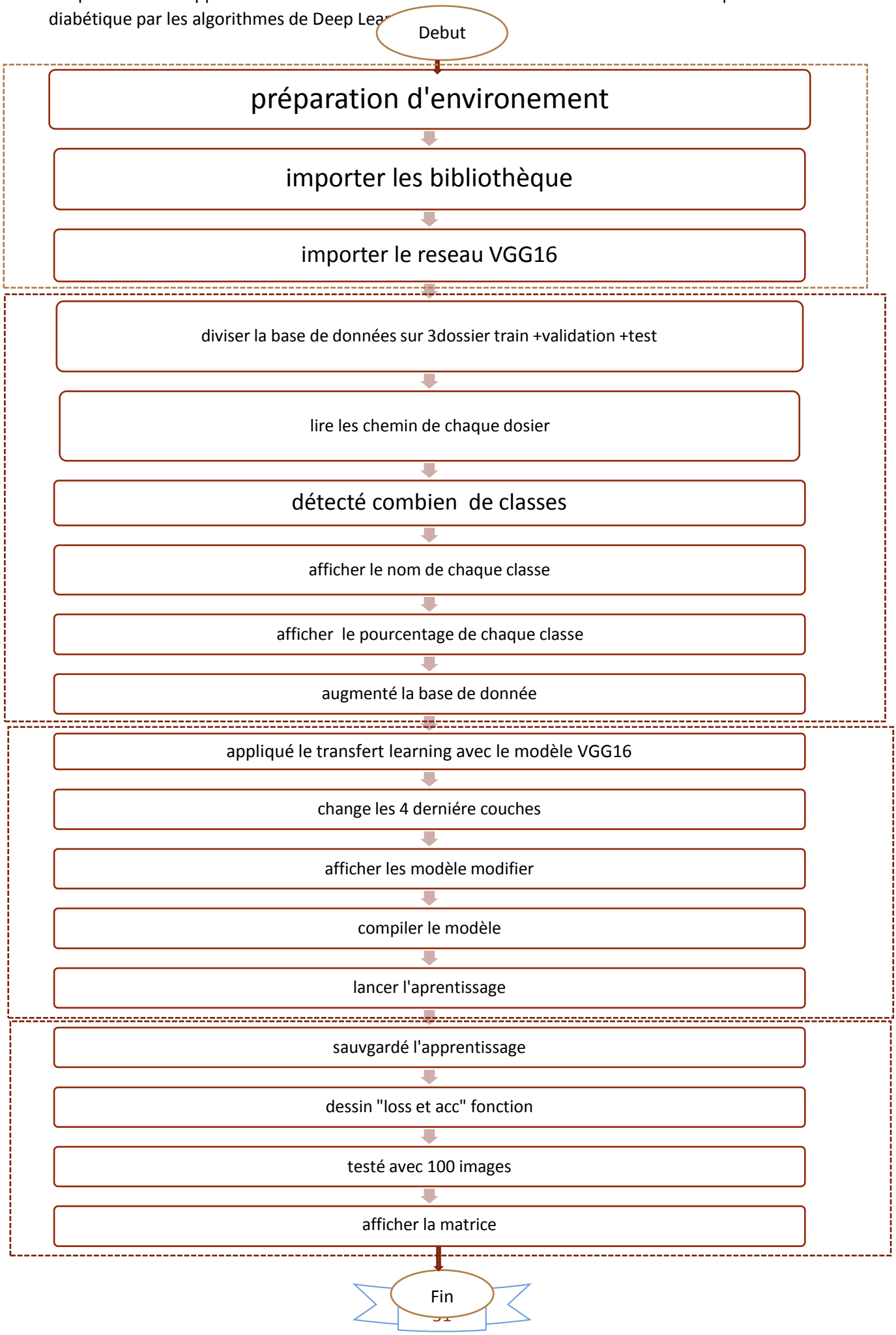

Debut
préparation d'environement
importer les bibliothèque
importer le reseau VGG16
diviser la base de données sur 3dossier train +validation +test
lire les chemin de chaque dosier
détecté combien de classes
afficher le nom de chaque classe
afficher le pourcentage de chaque classe
augmenté la base de donnée
appliqué le transfert learning avec le modèle VGG16
change les 4 derniére couches
afficher les modèle modifier
compiler le modèle
lancer l'aprentissage
sauvgardé l'apprentissage
dessin "loss et acc" fonction
testé avec 100 images
afficher la matrice
Fin

Figure II.1 : illustration des étapes

II.3. La base de données :

Est un outil essentiel pour faire des évaluations et des comparaisons fiables d'algorithmes de traitement d'images médicales est une base de données d'images médicales de haute qualité spécifiquement choisie pour représenter le problème.

Pour cela on a choisi notre base de données à partir d'une combinaison de deux bases de données « Diabetic Retinopathy 224x224 (2019 Data) » et « Diabetic Retinopathy 2015 Data Colored Resized » de la plateforme « kaggle ».

Cette base de données a été vérifiée par des experts, Elle est composée d'environ 5000 images du fond d'œil au format PNG, nous diviserons cette base de données trois fois distinctes, chaque fois à un nombre différent de classes afin que nous puissions comparer les résultats de l'apprentissage et trouver la meilleure division :

La première division de la base de données est représentée en cinq classes :

- La première classe avec des images de fond d'œil sans rétinopathie nommée « 0-pas de RD »
- Le reste des classes on a des images de fond d'œil atteint de la rétinopathie diabétique mais à des degrés divers. Le dossier a un petit pourcentage de la maladie et commence à s'augmenter dans les autres dossiers jusqu'à ce que nous atteignions le 5emme dossier qui se compose d'un grand pourcentage.
    - ✓ Deuxième classe « 1-légère
    - ✓ Troisième classe « 2-modéré »
    - ✓ Quatrième classe « 3-grave »
    - ✓ Cinquième classe « 4-RD proliférative »

La deuxième division de La même base de données est représentée cette fois en trois classe :

- ✓ La première classe avec des images de fond d'œil sans rétinopathie diabétique nommée « 0-pas de RD
- ✓ La deuxième classe est une combinaison de deux classes de la première division « légère et modéré « nommée
- ✓ La troisième classe est une combinaison de deux classes de la première division « grave et RD proliférative « nommée

Ensuite la dernière division de cette base de 5000 images représente deux classes :

- ✓ La première classe contient des images de rétine sain nommée « non »
- ✓ La deuxième classe contient des images de rétine malade nommée « oui »

Mais le nombre d'images est très peut pour entrainer nos réseaux de neurones. C'est pourquoi on a décidé d'augmentation le nombre d'images et d'utiliser une technique très

pratique pour résoudre ce problème. C'est la technique de « data augmentation «, qui sera définie ci-dessous.

II.4. Augmentation de données (data augmentation) :

L'augmentation des données est un processus intégré dans le cadre de l'apprentissage en profondeur (deep Learning), ou, par exemple, nous voulons former un réseau profond (deep network) a l'aide d'un algorithme d'apprentissage profond (deep Learning algorithm). Nous avons besoin d'une grande quantité de données, et parfois nous ne pouvons pas collecter des miliers et des millions d'images, l'augmentation de donnée (data augmentation) dont c'est la seule solution la taille de l'ensemble de données et peut modifier les données.

II.4.1 Augmentation de données et keras :

Nous avons une image originale dans nos données et nous souhaitons augmenter les données en appliquant différentes techniques a data augmentation dans keras, nous pouvons effectuer une augmentation de données en utilisant une fonction appelée IMAGDATAGENERATOR, cette fonction elle existe déjà dans la bibliothèque keras, il suffit de l'appeler.

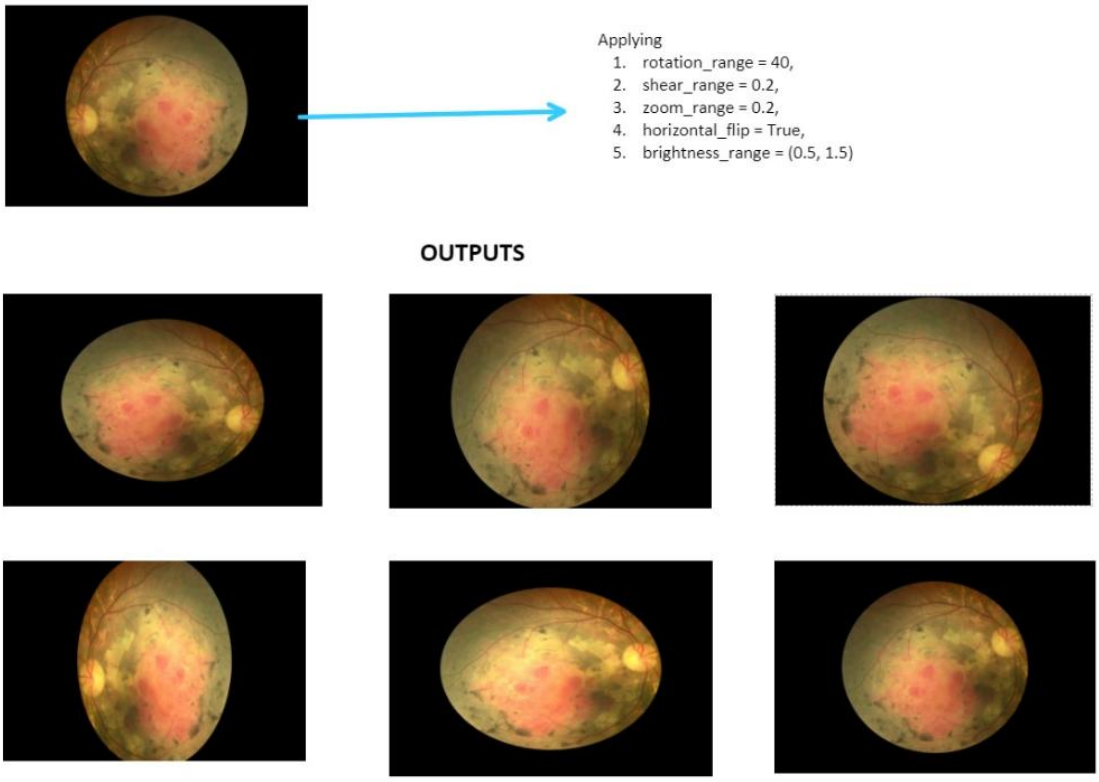


Figure II.2 : augmentation d'images

II.4.2 Types de data augmentation :

Il existe plusieurs techniques d'augmentation des données, on a choisi quelques-unes, elles sont reprises dans la figure ci-dessus :

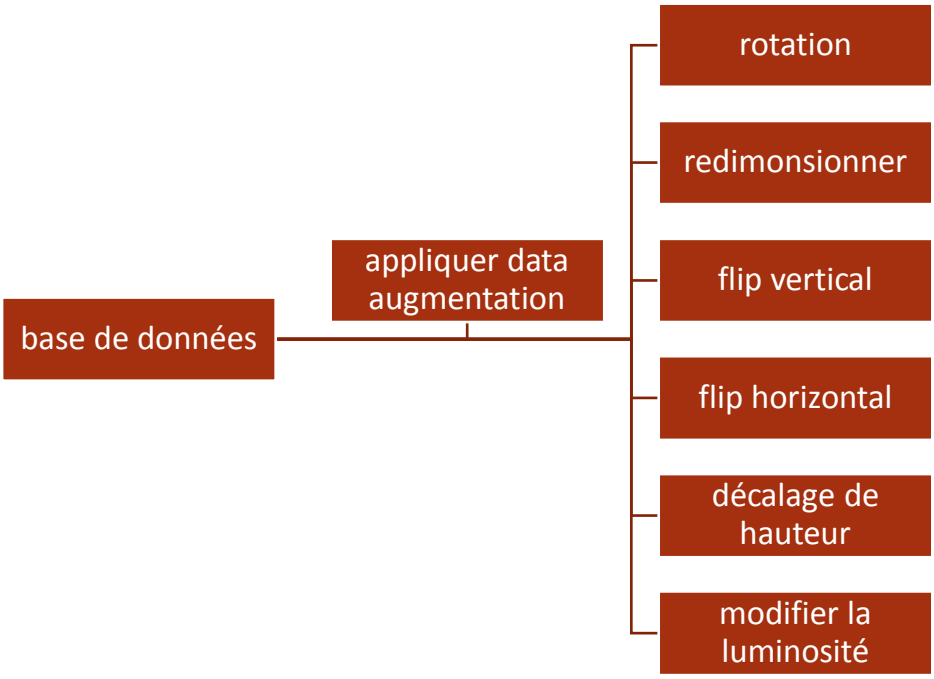


Figure II.3 : schéma synoptique des types de data augmentation

II.5. Transfert Learning pour le deep Learning:

II.5.1 Introduction :

L'apprentissage par transfert est une méthode d'apprentissage automatique dans laquelle un modèle développé pour une tâche est réutilisé comme point de départ d'un modèle sur une deuxième tâche.

C'est une approche populaire dans l'apprentissage en profondeur où des modèles pré-entraînés sont utilisés comme point de départ pour les tâches de vision par ordinateur et de traitement du langage naturel étant donné les vastes ressources de calcul et de temps nécessaires pour développer des modèles de réseaux neuronaux sur ces problèmes et des énormes sauts de compétences. Qu'ils fournissent sur des problèmes connexes [11].

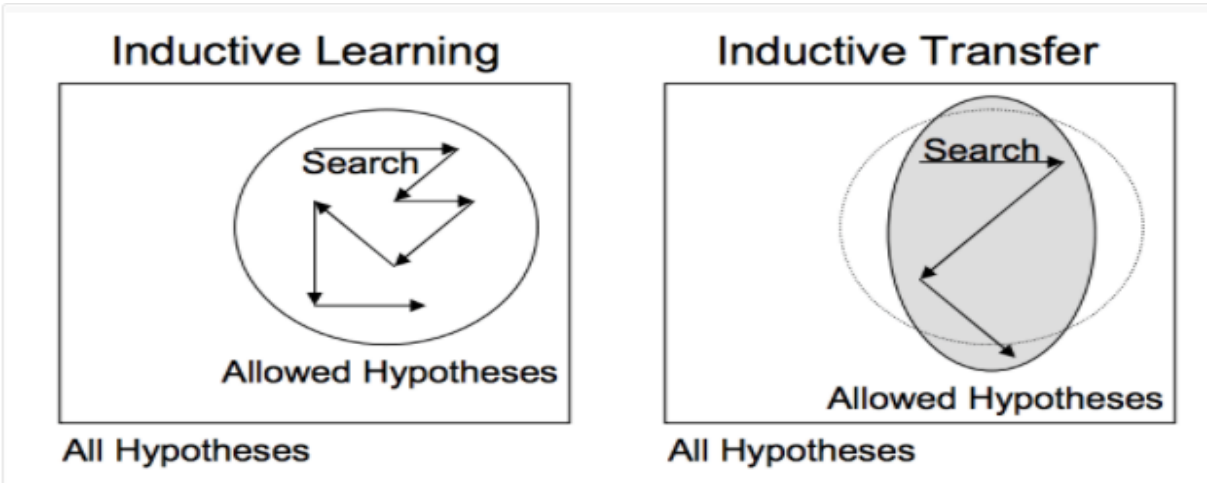


Figure II.4: Représentation du transfert inductif Extrait de « Transfert de l'apprentissage »

II.5.2 Comment utiliser l'apprentissage par transfert ?

Il faut utiliser l'apprentissage par transfert sur les propres problèmes de modélisation prédictive.

Deux approches courantes sont les suivantes :

a. Développer une approche modèle :

- Sélectionnez la tâche source. Vous devez sélectionner un problème de modélisation prédictive associé avec une abondance de données où il existe une relation entre les données d'entrée, les données de sortie et/ou les concepts appris lors du mappage des données d'entrée aux données de sortie.
- Développer un modèle source. Ensuite, vous devez développer un modèle habile pour cette première tâche. Le modèle doit être meilleur qu'un modèle naïf pour garantir qu'un certain apprentissage des fonctionnalités a été effectué.
- Réutiliser le modèle. L'ajustement du modèle sur la tâche source peut ensuite être utilisé comme point de départ pour un modèle sur la deuxième tâche d'intérêt. Cela peut impliquer l'utilisation de tout ou partie du modèle, selon la technique de modélisation utilisée.
- Modèle d'accord. Facultativement, le modèle peut avoir besoin d'être adapté ou affiné sur les données de paires d'entrées-sorties disponibles pour la tâche d'intérêt.

b. Approche du modèle pré-entraîné :

- Sélectionnez le modèle source. Un modèle source pré-entraîné est choisi parmi les modèles disponibles. De nombreux instituts de recherche publient des modèles sur des ensembles de données volumineux et difficiles qui peuvent être inclus dans le pool de modèles candidats parmi lesquels choisir.
- Réutiliser le modèle. Le modèle pré-entraîné peut ensuite être utilisé comme point de départ pour un modèle sur la deuxième tâche d'intérêt. Cela peut impliquer l'utilisation de tout ou partie du modèle, selon la technique de modélisation utilisée.
- Modèle d'accord. Facultativement, le modèle peut avoir besoin d'être adapté ou affiné sur les données de paires d'entrées-sorties disponibles pour la tâche d'intérêt.

Ce deuxième type d'apprentissage par transfert est courant dans le domaine de l'apprentissage profond

II.5.3 Quand utiliser l'apprentissage par transfert ?

L'apprentissage par transfert est une optimisation, un raccourci pour gagner du temps ou obtenir de meilleures performances.

En général, il n'est pas évident qu'il y aura un avantage à utiliser l'apprentissage par transfert dans le domaine jusqu'à ce que le modèle ait été développé et évalué.

Il y a trois avantages possibles à rechercher lors de l'utilisation de l'apprentissage par transfert :

- Début plus élevé. La compétence initiale (avant d'affiner le modèle) sur le modèle source est plus élevée qu'elle ne le serait autrement.
- Pente plus élevée. Le taux d'amélioration des compétences pendant la formation du modèle source est plus élevé qu'il ne le serait autrement.
- Asymptote supérieure. La compétence convergée du modèle formé est meilleure qu'elle ne le serait autrement.

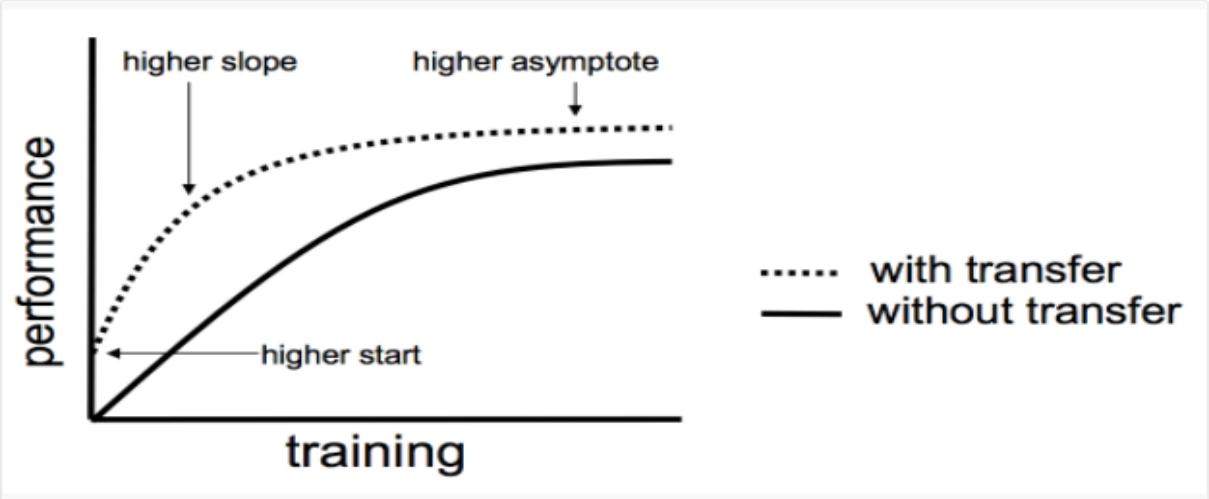


Figure II.5 : Trois façons dont le transfert pourrait améliorer l'apprentissage.

Idéalement, vous verriez les trois avantages d'une application réussie de l'apprentissage par transfert.

C'est une approche à essayer si vous pouvez identifier une tâche connexe avec des données abondantes et si vous avez les ressources pour développer un modèle pour cette tâche et le réutiliser sur votre propre problème, ou s'il existe un modèle pré-entraîné disponible que vous pouvez utiliser comme un point de départ pour votre propre modèle.

Sur certains problèmes pour lesquels vous n'avez peut-être pas beaucoup de données, l'apprentissage par transfert peut vous permettre de développer des modèles habiles que vous ne pourriez tout simplement pas développer en l'absence d'apprentissage par transfert.

Le choix des données sources ou du modèle source est un problème ouvert et peut nécessiter une expertise du domaine et/ou une intuition développée par l'expérience

II.6. Structures modifiées des réseaux utilisées :

II.6.1 Le modèle vgg16 :

Le jeu de données ImageNet contient des images de taille fixe de 224*224 et possède des canaux RVB. Donc, nous avons un tenseur de (224, 224, 3) comme entrée. Ce modèle traite l'image d'entrée et génère un vecteur de 1000 valeurs.

VGG-16 était l'une des architectures les plus performantes du défi ILSVRC 2014. C'était le finaliste de la tâche de classification avec une erreur de classification dans le top 5 de 7,32 % (seulement derrière Google Net avec une erreur de classification de 6,66 %). Il a également remporté la tâche de localisation avec une erreur de localisation de 25,32 %. [12]

Pour que ce réseau soit adapté à notre travail, Nous devons appliquer Le transfert Learning, On garde les mêmes poids des premières couches du réseau original mais on enlève les quatre dernières couches « fully connected ».

Ensuite, on ajoute notre propre couche « fully connected » avec « Sigmoïde » comme fonction d'activation si on a deux classes et aussi le classifieur « Softmax » si on a plus que deux classes. Les poids au niveau de ces couches modifiées seront entrainés sur notre base de donnée.

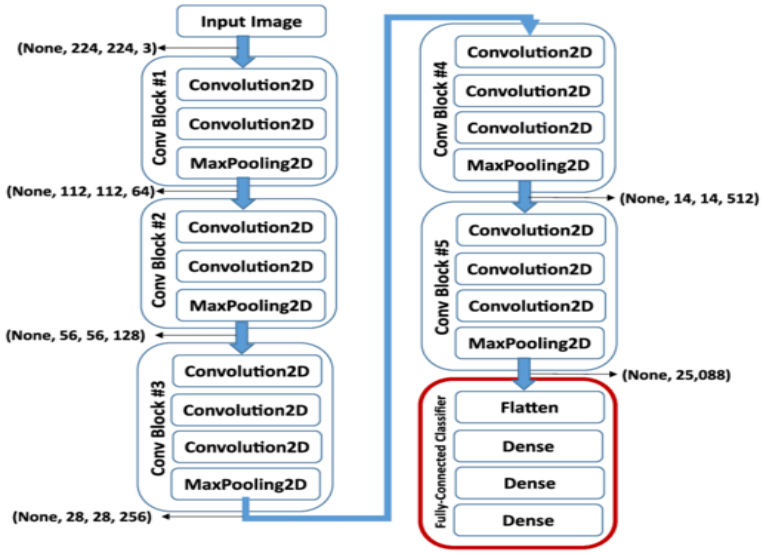


Figure II.6: architecture original de VGG16

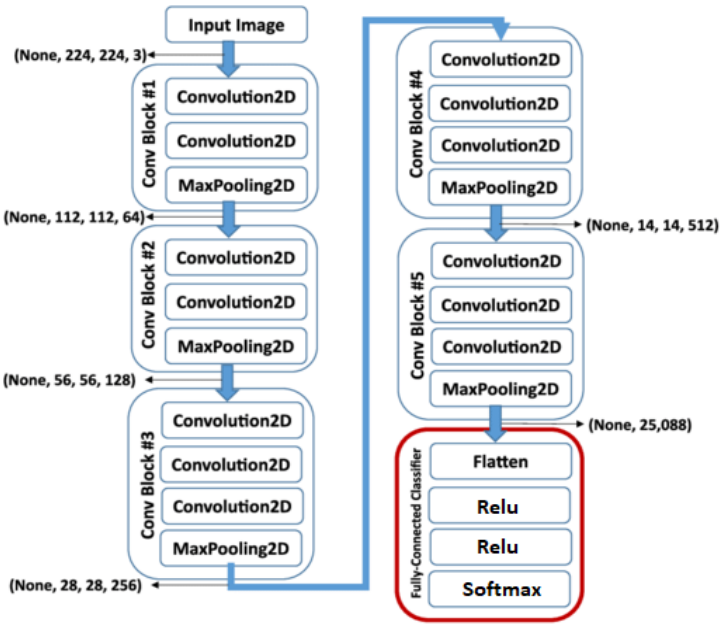


Figure II.7 : architecture modifier de le reseau VGG16

II.6.2 Inception_v3 :

L'architecture d'un réseau Inception_v3 se construit progressivement, étape par étape, comme expliqué ci-dessous :

1. **Convolutions factorisées :** cela permet de réduire l'efficacité de calcul car il réduit le nombre de paramètres impliqués dans un réseau. Il contrôle également l'efficacité du réseau.
2. **Des circonvolutions plus petites :** remplacer les circonvolutions plus grandes par des circonvolutions plus petites conduit certainement à un entraînement plus rapide. Disons qu'un filtre 5 × 5 à 25 paramètres ; deux filtres 3 × 3 remplaçant une convolution 5 × 5 n'ont que 18 paramètres (3*3 + 3*3) à la place.
3. **Convolutions asymétriques :** Une convolution 3 × 3 pourrait être remplacée par une convolution 1 × 3 suivie d'une convolution 3 × 1. Si une convolution 3 × 3 est remplacée par une convolution 2 × 2, le nombre de paramètres serait légèrement supérieur à la convolution asymétrique proposée.

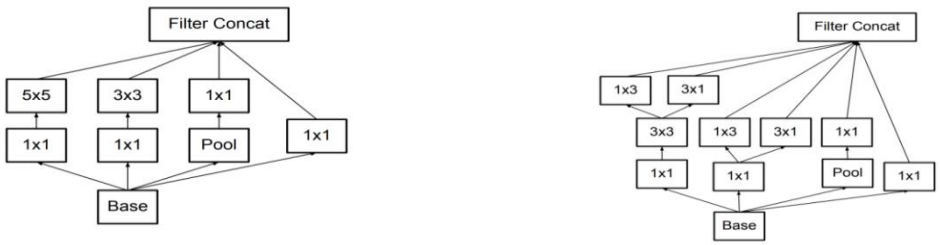


Figure II.8 : architecture de convolutions asymétriques

4. **Classificateur auxiliaire :** un classificateur auxiliaire est un petit CNN inséré entre les couches pendant l'apprentissage, et la perte subie s'ajoute à la perte du réseau principal. Dans Google Net, les classificateurs auxiliaires ont été utilisés pour un réseau plus profond, tandis que dans Inception_v3, un classificateur auxiliaire agit comme un régularisateur.

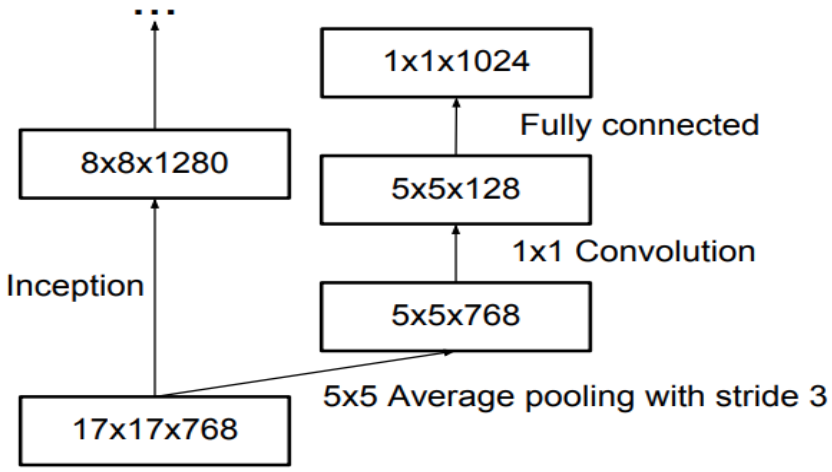


Figure II.9 : architecture de classificateur auxiliaire

5. **Réduction de la taille de la grille :** La réduction de la taille de la grille se fait généralement en regroupant les opérations. Cependant, pour lutter contre les goulots d'étranglement du coût de calcul, une technique plus efficace est proposée [13]:

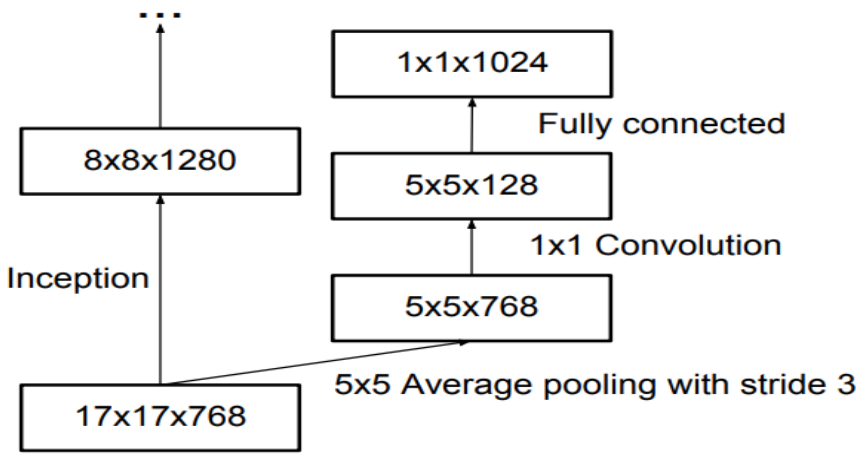


Figure II.10 : réduction de la taille de grille

II.6.3 Alex-net :

L'architecture se compose de huit couches : cinq couches convolutives et trois couches entièrement connectées. Mais ce n'est pas ce qui rend AlexNet spécial ; voici quelques-unes des fonctionnalités utilisées qui constituent de nouvelles approches des réseaux de neurones convolutifs :

- Non-linéarité ReLU. AlexNet utilise des unités linéaires rectifiées (ReLU) au lieu de la fonction tanh, qui était standard à l'époque. L'avantage de ReLU réside dans le temps

de formation ; un CNN utilisant ReLU a pu atteindre une erreur de 25 % sur l'ensemble de données CIFAR-10 six fois plus rapidement qu'un CNN utilisant tanh.

- Plusieurs GPU. À l'époque, les GPU fonctionnaient toujours avec 3 gigaoctet de mémoire (aujourd'hui, ces types de mémoire seraient des numéros de débutants). C'était particulièrement mauvais parce que l'ensemble d'entraînement contenait 1,2 million d'images. AlexNet permet une formation multi-GPU en plaçant la moitié des neurones du modèle sur un GPU et l'autre moitié sur un autre GPU. Non seulement cela signifie qu'un modèle plus grand peut être formé, mais cela réduit également le temps de formation.
- Chevauchement de la mise en commun. Les CNN « regroupent » traditionnellement les sorties de groupes de neurones voisins sans chevauchement. Cependant, lorsque les auteurs ont introduit le chevauchement, ils ont constaté une réduction de l'erreur d'environ 0,5 % et ont constaté que les modèles avec mise en commun avec chevauchement ont généralement plus de difficulté à su rajuster [14].

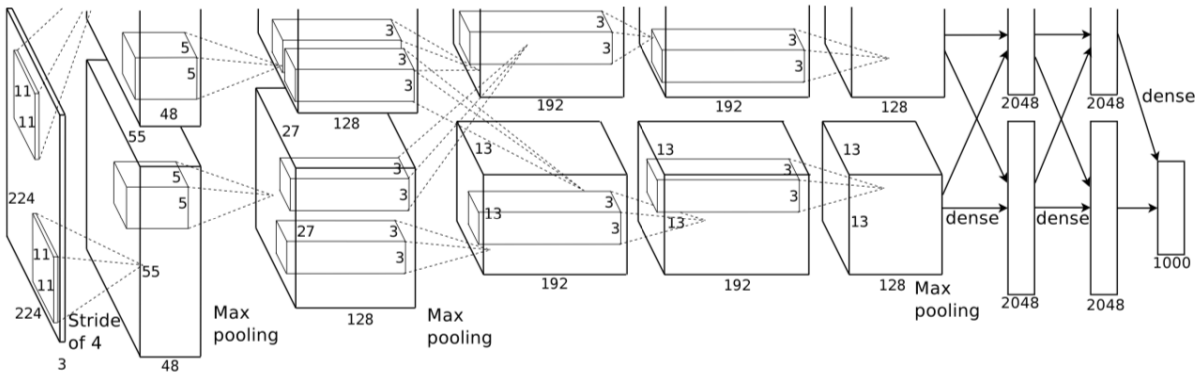


Figure II.11 : Illustration de l'architecture d'AlexNet.

II.6.4 U-NET :

L'architecture du réseau est illustrée à la figure. Elle se compose d'un chemin de contraction (côté gauche) et d'un chemin expansif (côté droit). Le chemin contractuel suit l'architecture typique d'un réseau convolutif. Il consiste en l'application répétée de deux convolutions 3x3 (convolutions non rembourrées), chacune suivie d'une unité linéaire rectifiée (ReLU) et d'une opération de pooling 2x2 max avec foulée 2 pour le sous-échantillonnage. À chaque étape de sous-échantillonnage, nous doublons le nombre de canaux de fonctionnalités. Chaque étape du chemin expansif consiste en un suréchantillonnage de la carte des caractéristiques suivi d'une convolution 2x2 (« convolution vers le haut ») qui divise par deux le nombre de canaux de caractéristiques, une concaténation avec la carte des caractéristiques rognée en conséquence du chemin de contraction, et deux 3x3 circonvolutions, chacune suivie d'une ReLU. Le recadrage est nécessaire en raison de la perte de pixels de bordure dans chaque convolution. Au niveau de la couche finale, une convolution 1x1 est utilisée pour mapper chaque vecteur d'entités à 64 composants au nombre de classes souhaité. Au total, le réseau compte 23 couches convolutives.

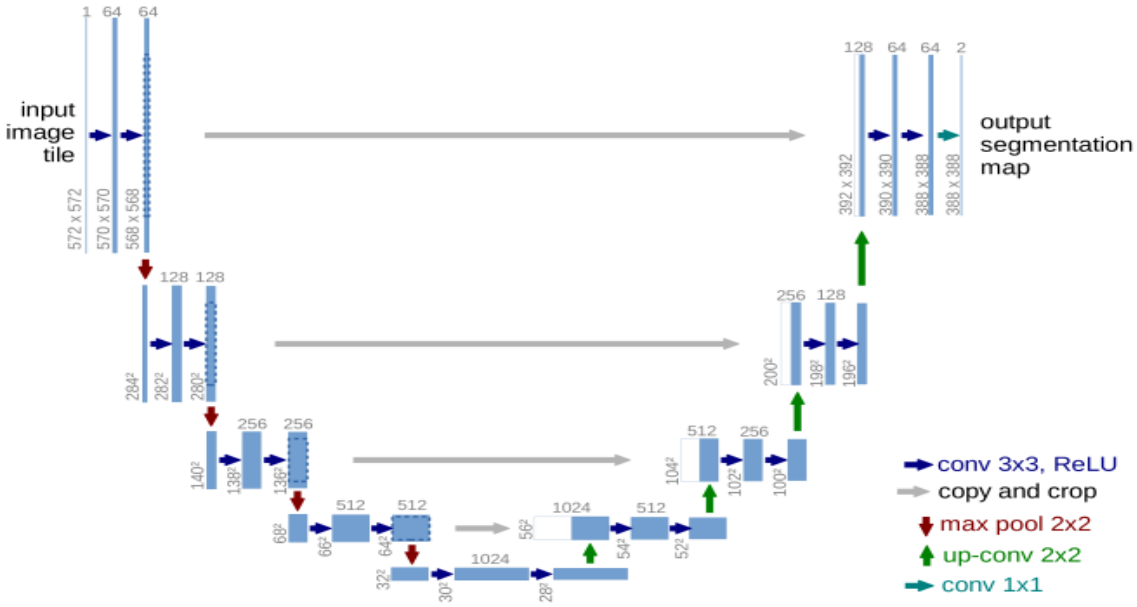


Figure II.12: U-net architecture

Pour permettre une mosaïque transparente de la carte de segmentation en sortie, il est important de sélectionner la taille de la mosaïque en entrée de telle sorte que toutes les opérations de pooling max 2x2 soient appliquées à une couche avec une taille x et y paire [15].

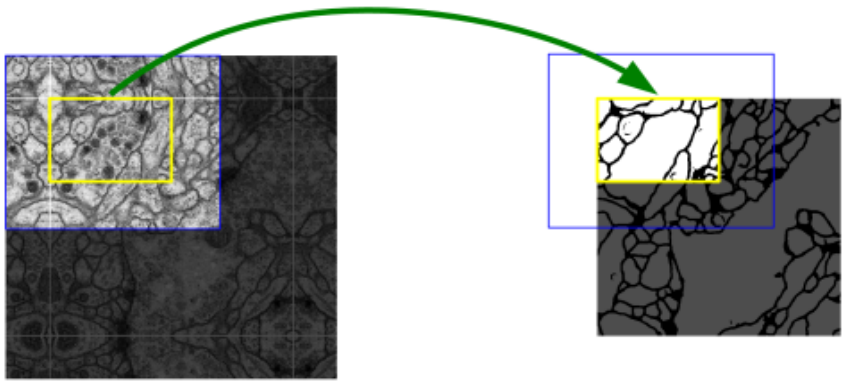


Figure II.13: Stratégie de superposition de tuiles

II.7. Paramètres d'entrainement :

II.7.1. Le classifieur softmax :

La régression logistique produit un nombre décimal compris entre 0 et 1,0. Par exemple, une sortie de régression logistique de 0,8 à partir d'un classificateur d'e-mails suggère une probabilité de 80 % qu'un e-mail soit du spam et 20 % de chances qu'il ne soit pas du spam.

De toute évidence, la somme des probabilités qu'un e-mail soit du spam ou non du spam est de 1,0.

Softmax étend cette idée dans un monde multi-classes. C'est-à-dire que Softmax attribue des probabilités décimales à chaque classe dans un problème multi-classes. Ces probabilités décimales doivent totaliser 1,0. Cette contrainte supplémentaire permet à la formation de converger plus rapidement qu'elle ne le ferait autrement.

Par exemple, en revenant à l'analyse d'image que nous avons vue sur la figure, Softmax pourrait produire les probabilités suivantes d'une image appartenant à une classe particulière :

Class	Probability
apple	0.001
bear	0.04
candy	0.008
dog	0.95
egg	0.001

Figure II.14 : exemple sur l'analyse d'image

Softmax est implémenté via une couche de réseau neuronal juste avant la couche de sortie. La couche Softmax doit avoir le même nombre de nœuds que la couche de sortie.

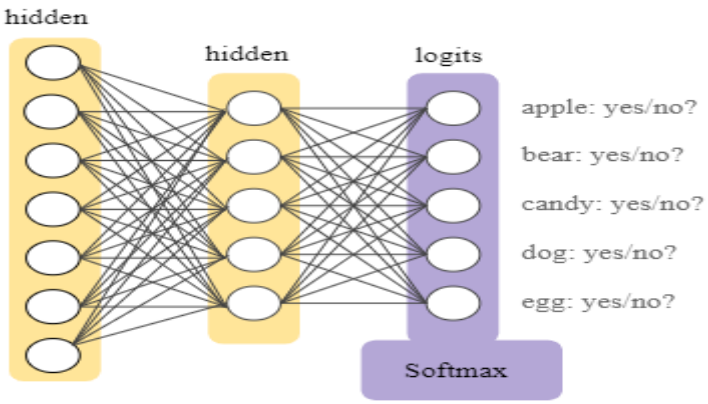


Figure II.15: Une couche Softmax dans un réseau de neurones.

L'équation Softmax est la suivante :

$$p(y = j|x) = \frac{e^{(w_j^T x + b_j)}}{\sum_{k \epsilon K} e^{(w_k^T x + b_k)}}$$

a. Options Softmax :

Considérez les variantes suivantes de Softmax :

- Full Softmax est le Softmax dont nous avons parlé ; c'est-à-dire que Softmax calcule une probabilité pour chaque classe possible.
- L'échantillonnage des candidats signifie que Softmax calcule une probabilité pour toutes les étiquettes positives, mais uniquement pour un échantillon aléatoire d'étiquettes négatives. Par exemple, si nous souhaitons déterminer si une image d'entrée est un beagle ou un limier, nous n'avons pas à fournir de probabilités pour chaque exemple non doggy.

Full Softmax est assez bon marché lorsque le nombre de classes est petit mais devient prohibitif lorsque le nombre de classes grimpe. L'échantillonnage de candidats peut améliorer l'efficacité dans les problèmes ayant un grand nombre de classes [16].

II.7.2. Les fonctions de perte (Loss function) :

Une fonction de perte, ou Loss function, est une fonction qui évalue l'écart entre les prédictions réalisées par le réseau de neurones et les valeurs réelles des observations utilisées pendant l'apprentissage. Plus le résultat de cette fonction est minimisé, plus le réseau de neurones est performant. Sa minimisation, c'est-à-dire réduire au minimum l'écart entre la valeur prédite et la valeur réelle pour une observation donnée, se fait en ajustant les différents poids du réseau de neurones [17].

II.7.3. L'entropie croisée (cross-entropy) :

Lorsque vous travaillez sur un Machine Learning ou un Deep Learning Problème, les fonctions de perte / coût sont utilisées pour optimiser le modèle pendant la formation. L'objectif est presque toujours de minimiser la fonction de perte. Plus la perte est faible, meilleur est le modèle. La perte d'entropie croisée est une fonction de coût la plus importante. Il est utilisé pour optimiser les modèles de classification. La compréhension de l'entropie croisée repose sur la compréhension de la fonction d'activation de Softmax.

Aussi appelée perte logarithmique, la perte de journal ou pertes logistiques. Chaque probabilité de classe prédite est comparée à la sortie souhaitée de classe réelle 0 ou 1 et un score / perte est calculé qui pénalise la probabilité en fonction de sa distance par rapport à la valeur attendue réelle. La pénalité est de nature logarithmique donnant un score élevé pour les grandes différences proches de 1 et un petit score pour les petites différences tendant à 0.

La perte d'entropie croisée est utilisée lors de l'ajustement des poids du modèle pendant l'entraînement. Le but est de minimiser la perte, c'est-à-dire que plus la perte est petite, meilleur est le modèle. Un modèle parfait a une perte d'entropie croisée de 0.

L'entropie croisée est définie comme :

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Figure : Définition mathématique de Cross-Entopy. Notez que le journal est calculé sur la base 2.

Perte d'entropie croisée binaire :

Pour la classification binaire, nous avons une entropie croisée binaire définie comme

$$\begin{aligned} L &= -\sum_{i=1}^{2} t_i \log(p_i) \\ &= -\left[t \log(p) + (1-t) \log(1-p)\right] \end{aligned}$$

where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} class.

Figure II .16: Définition mathématique de la Cross-Entopy binaire.

L'entropie croisée binaire est souvent calculée comme l'entropie croisée moyenne dans tous les exemples de données [18]

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N} \left[t_j \log(p_j) + (1-t_j) \log(1-p_j)\right]\right]$$

for N data points where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} data point.

II.7.4. Epoques (epochs) :

Une époque dans l'apprentissage automatique signifie un passage complet de l'ensemble de données d'entraînement à travers l'algorithme. Ce nombre d'époques est un hyper paramètre important pour l'algorithme. Il spécifie le nombre d'époques ou de passes complètes de l'ensemble de données d'entraînement passant par le processus d'entraînement ou d'apprentissage de l'algorithme. À chaque époque, les paramètres du modèle interne du jeu de données sont mis à jour. Par conséquent, une époque de 1 lot est appelée algorithme d'apprentissage de descente de gradient par lots. Normalement, la taille du lot d'une époque est de 1 ou plus et est toujours une valeur entière dans ce qui est le numéro d'époque [19].

II.7.5. Optimisation de fonction, descentes de gradient :

a. Optimisation de données :

L'optimisation est un des domaines des maths qui intervient le plus en machine Learning, Pour la construction et l'entraînement de réseaux de neurones, l'optimisation est très présente. De manière schématique, En pratique, nos données sont partagées en deux, les données d'entrainements et les données de test. Le réseau de neurones est une machine qui à partir d'une donnée d'entrée, construit une sortie. On va donc entrainer nos réseaux jusqu'à ce que les sorties pour nos données d'entrainements soient aussi proches que possible de ce que l'on veut. Pour cela il faut minimiser l'erreur entre la sortie de notre réseau et la sortie souhaitée. C'est là que l'optimisation intervient ! La plupart du temps se sont les méthodes de descentes de gradient qui sont utilisées pour minimiser cette erreur [20].

b. Méthodes de descentes de gradients :

Pour toutes les méthodes de descentes de gradient le principe est le même, on cherche à s'approcher étape après étape du minimum d'une fonction, Pour se faire, on choisit un point initial et on évalue la valeur de la dérivée de la fonction en ce point, Selon le signe de la dérivée, on se déplace vers la droite ou vers la gauche jusqu'à être assez proche du minimum. Pour l'itération on fixe un pas que l'on appelle Learning rate. C'est ce pas qui détermine à quel point on se décale de la position initiale [21].

c. Taux d'apprentissage :

Le taux d'apprentissage (appelé en anglais Learning rate), souvent noté \alphaα ou parfois \etaη, indique la vitesse à laquelle les coefficients évoluent. Cette quantité peut être fixe ou variable. L'une des méthodes les plus populaires à l'heure actuelle s'appelle Adam, qui a un taux d'apprentissage qui s'Attention, le choix du Learning rate doit être fait de façon intelligente ! Un Learning rate trop grand ne permettra pas de s'approcher assez du minimum, un Learning rate trop petit peut augmenter de façon significative le temps de calcul et n'oubliez pas qu'en deep Learning on veut optimiser des fonctions de dimensions très élevées adapte au fil du temps [22].

d. D'optimisation Adam :

L'algorithme d'estimation adaptative du mouvement, ou Adam en abrégé, est une extension de l'algorithme d'optimisation de la descente de gradient.

Adam est conçu pour accélérer le processus d'optimisation, par exemple réduire le nombre d'évaluations de fonctions nécessaires pour atteindre les optima, ou pour améliorer la capacité de l'algorithme d'optimisation, par exemple pour obtenir un meilleur résultat final.

Ceci est réalisé en calculant une taille de pas pour chaque paramètre d'entrée qui est optimisé. Surtout, chaque taille de pas est automatiquement adaptée en fonction du processus de recherche en fonction des gradients (dérivés partiels) rencontrés pour chaque variable.

Les paramètres impliqués :

- Alpha (α) : Taux d'apprentissage pour l'étape de descente de gradient.
- Beta (β1) : Paramètre du pas de moment (également appelé premier moment en Adam). Généralement 0,9

- Beta (β2) : Paramètre pour l'étape RMSProp (également appelé second moment en Adam). Généralement 0,99
- Epsilon (ϵ) : Paramètre de stabilité numérique. Généralement 10 ^ -8
- M, v : Estimations des premiers et deuxièmes moments, respectivement. Les valeurs initiales des deux sont définies sur 0.
- T : Le paramètre de pas de temps pour les étapes de correction de biais.
- G et f : Gradient et valeurs de fonction à θ [23].

```
m = beta1 * m + (1 - beta1) * g
```

```
m_corrected = m / (1 - np.power(beta1, t))
```

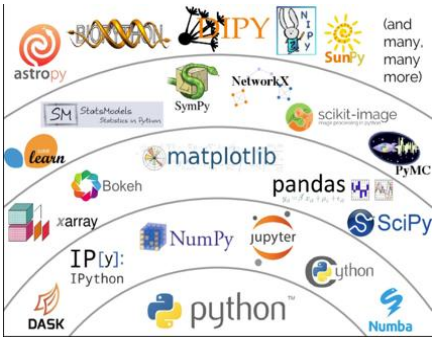
II.8. Langage de programmation (python) :

II.8.1. Définition :

Python est un langage de programmation open source. Python est le langage le plus utilisé pour la Data Science. Pour cause, ce langage est simple, lisible, propre, flexible et compatible avec de nombreuses plateformes. Ses nombreuses bibliothèques, telles que TensorFlow, Scipy et Numpy permettent d'effectuer une large variété de tâches.

Sa syntaxe très simple le rend utilisable par des personnes n'ayant pas forcément de background en ingénierie.

Il permet le prototypage rapide, et le code peut être exécuté n'importe où : Windows, macOS, UNIX, Linux… sa flexibilité permet de prendre en charge le développement de modèles de Machine Learning, le forage de données, la classification et bien d'autres tâches plus rapidement que les autres langages.



Des bibliothèques comme Scrapy et BeautifulSoup permettent d'extraire des données depuis internet, tandis que Seaborn et Matplotlib aident à la Data Visualization. De leur côté, Tensorflow, Keras et Theano permettent le développement de modèles de Deep Learning, et Scikit-Learn aide au développement d'algorithmes de Machine Learning [24].

II.8.2. Les bibliothèques python :

a. NumPy :

NumPy (pour Numerical Python) est un outil parfait pour le calcul scientifique et la réalisation d'opérations de base et avancées avec des tableaux.La bibliothèque offre de nombreuses fonctionnalités pratiques permettant d'effectuer des opérations sur des tableaux

(n-arrays) et des matrices en Python. Elle permet de traiter des tableaux qui stockent des valeurs du même type de données et facilite l'exécution d'opérations mathématiques sur les tableaux (et leur vectorisation). En fait, la vectorisation des opérations mathématiques sur le type de tableau NumPy augmente les performances et accélère le temps d'exécution.

b. SciPy :

Cette bibliothèque utile comprend des modules pour l'algèbre linéaire, l'intégration, l'optimisation et les statistiques. Sa fonctionnalité principale a été construite sur NumPy, donc ses tableaux utilisent cette bibliothèque. SciPy fonctionne parfaitement pour toutes sortes de projets de programmation scientifique (sciences, mathématiques et ingénierie). Il offre des routines numériques efficaces telles que l'optimisation numérique, l'intégration et d'autres dans des sous-modules. La vaste documentation rend le travail avec cette bibliothèque vraiment facile.

c. Pandas :

Pandas est une bibliothèque créée pour aider les développeurs à travailler intuitivement avec des données "étiquetées" et "relationnelles". Elle est basée sur deux structures de données principales : "Série" (unidimensionnelle, comme une liste Python) et "Dataframe" (bidimensionnelle, comme un tableau à plusieurs colonnes). Pandas permet de convertir des structures de données en objets DataFrame, de gérer les données manquantes et d'ajouter/supprimer des colonnes de DataFrame, d'imputer les fichiers manquants et de tracer les données avec un histogramme ou une boîte à moustache. C'est un outil indispensable pour la manipulation et la visualisation des données.

d. Keras :

Keras est une excellente bibliothèque pour la construction de réseaux de neurones et la modélisation. Elle est très simple à utiliser et offre aux développeurs un bon degré d'extensibilité. La bibliothèque tire parti d'autres paquets (Theano ou TensorFlow) comme terminaux. De plus, Microsoft a intégré CNTK (Microsoft Cognitive Toolkit) pour servir d'autre backend.

e. Scikit-Learn :

Il s'agit d'une norme industrielle pour les projets de science des données basés en Python. Scikits est un groupe de paquets de SciPy qui ont été créés pour des fonctionnalités spécifiques – par exemple, le traitement d'images. Scikit-learn utilise les opérations mathématiques de SciPy pour exposer une interface concise aux algorithmes d'apprentissage machine les plus courants.

Les spécialistes des données l'utilisent pour traiter les tâches standard de Machine Learning et d'exploration de données telles que le regroupement, la régression, la sélection de modèles, la réduction de la dimensionnalité et la classification.

f. PyTorch :

PyTorch est un framework qui est parfait pour les data scientists qui veulent effectuer facilement des tâches de Deep Learning. L'outil permet d'effectuer des calculs de tenseurs avec une accélération GPU. Il est également utilisé pour d'autres tâches – par exemple, pour créer des graphiques de calcul dynamiques et calculer automatiquement des gradients. PyTorch est basé sur Torch, qui est une bibliothèque open-source de Deep Learning, implémentée en C, avec un habillage en Lua.

g. TensorFlow :

TensorFlow est un framework Python populaire pour le Machine Learning et le Deep Learning, qui a été développé à Google Brain. C'est le meilleur outil pour des tâches comme l'identification d'objets, la reconnaissance vocale et bien d'autres. Il permet de travailler avec des réseaux neuronaux artificiels qui doivent gérer plusieurs ensembles de données. La bibliothèque comprend plusieurs aides de couches (tflearn, tf-slim, skflow), qui la rendent encore plus fonctionnelle. TensorFlow s'enrichit constamment de nouvelles versions, notamment en corrigeant les éventuelles failles de sécurité ou en améliorant l'intégration de TensorFlow et du GPU.

h. Matplotlib :

Matplotlib est une bibliothèque scientifique de données standard qui aide à générer des visualisations de données telles que des diagrammes et des graphiques bidimensionnels (histogrammes, diagrammes de dispersion, graphiques de coordonnées non cartésiennes). Matplotlib est l'une de ces bibliothèques de tracés qui sont vraiment utiles dans les projets de science des données – elle fournit une API orientée objet pour intégrer des tracés dans les applications.

C'est grâce à cette bibliothèque que Python peut rivaliser avec des outils scientifiques comme MatLab ou Mathematica. Cependant, les développeurs doivent écrire plus de code que d'habitude en utilisant cette bibliothèque pour générer des visualisations avancées. Notez que les bibliothèques de tracés populaires fonctionnent sans problème avec Matplotlib

i. OpenCV :

La bibliothèque OpenCV met à disposition de nombreuses fonctionnalités très diversifiées permettant de créer des programmes en partant des données brutes pour aller jusqu'à la création d'interfaces graphiques basiques [25].

II.9. Google colab :

II.9.1. Définition :

Google est assez agressif dans la recherche sur l'IA. Pendant de nombreuses années, Google a développé un framework d'IA appelé TensorFlow et un outil de développement appelé Colaboratory. Aujourd'hui, TensorFlow est open source et depuis 2017, Google a rendu Colaboratory gratuit pour un usage public. Colaboratory est maintenant connu sous le nom de Google Colab ou simplement Colab.

Une autre fonctionnalité intéressante que Google propose aux développeurs est l'utilisation du GPU. Colab prend en charge le GPU et il est totalement gratuit. Les raisons de le rendre gratuit pour le public pourraient être de faire de son logiciel un standard dans les universitaires pour l'enseignement de l'apprentissage automatique et de la science des données. Il peut également avoir une perspective à long terme de constitution d'une clientèle pour les API Google Cloud qui sont vendues à l'utilisation.

Si vous avez déjà utilisé Jupyter Notebook, vous apprendrez rapidement à utiliser Google Colab. Pour être précis, Colab est un environnement de notebook Jupyter gratuit qui s'exécute entièrement dans le cloud. Plus important encore, il ne nécessite pas de configuration et les blocs-notes crées peuvent être modifiés simultanément par les membres d'une même équipe - tout comme on modifie des documents dans Google Docs. Colab prend en charge de nombreuses bibliothèques d'apprentissage automatique populaires qui peuvent être facilement chargées dans votre bloc-notes [26].

Que propose Colab ?

- Écrire et exécuter du code en Python
- Documentez votre code qui prend en charge les équations mathématiques
- Créer/Télécharger/Partager des blocs-notes
- Importer/enregistrer des blocs-notes depuis/vers Google Drive
- Importer/publier des blocs-notes depuis GitHub
- Importez des ensembles de données externes, par ex. de Kaggle
- Intégrer PyTorch, TensorFlow, Keras, OpenCV
- Service Cloud gratuit avec GPU gratuit

II.9.2. Utiliser la plateforme colab :

Remarque : Colab utilisant implicitement Google Drive pour stocker vos blocs-notes.

Nous ouvrons l'URL suivante dans notre navigateur https://colab.research.google.com/ notre navigateur affichera l'écran suivant (en supposant que nous sommes connecté à notre Google **Drive)**

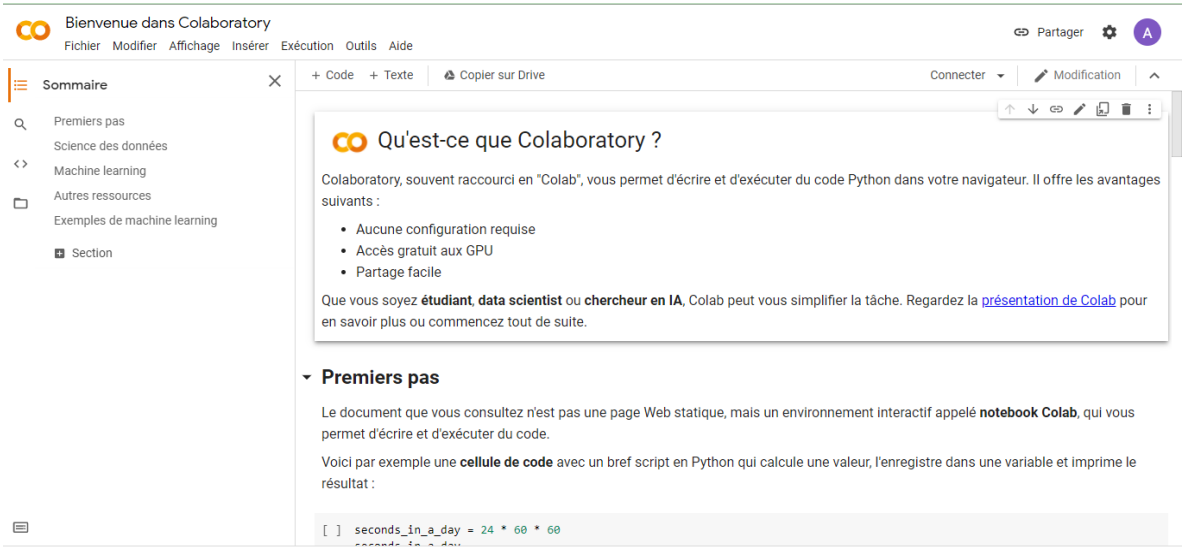


Figure II.17 : la page d'accueil de la plateforme.

- Cliquer sur fichier et choisissez nouveau notebook

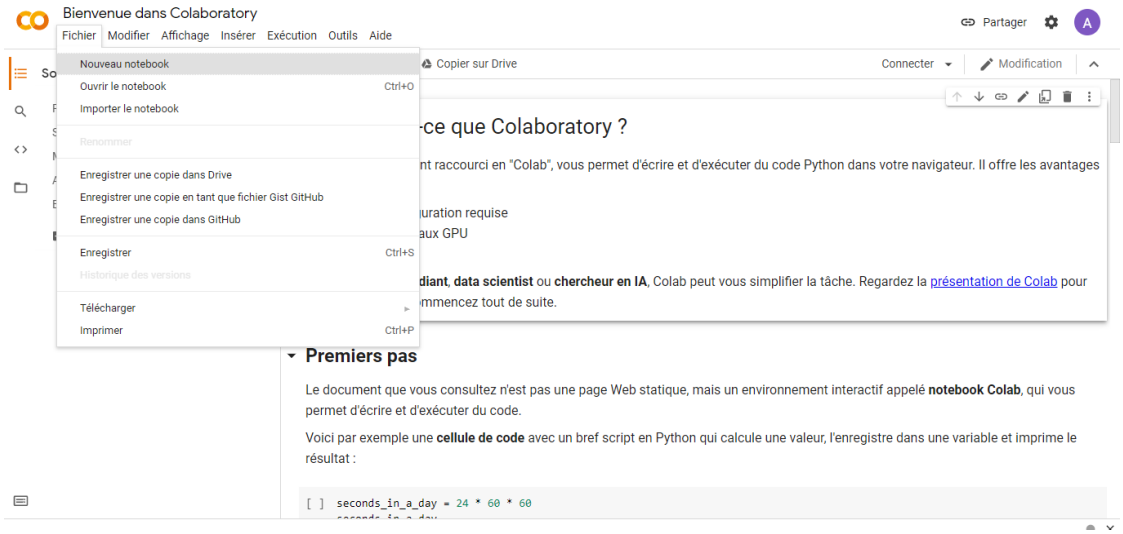


Figure II.18: illustration de comment ouvrir un nouveau notebook dans la plateforme colab .

- Lorsque nous ouvrons un nouveau notebook, nous serons prêts à écrire le code dans la cellule de code.

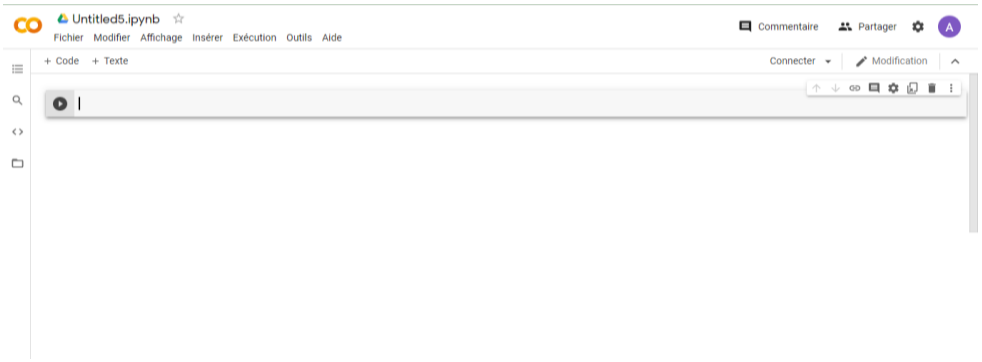


Figure II.19 : illustration de nouveau note book.

- si vous souhaitez ajouter une cellule de code, cliquez sur l'icône +code, ou sur insérer =>cellule de code

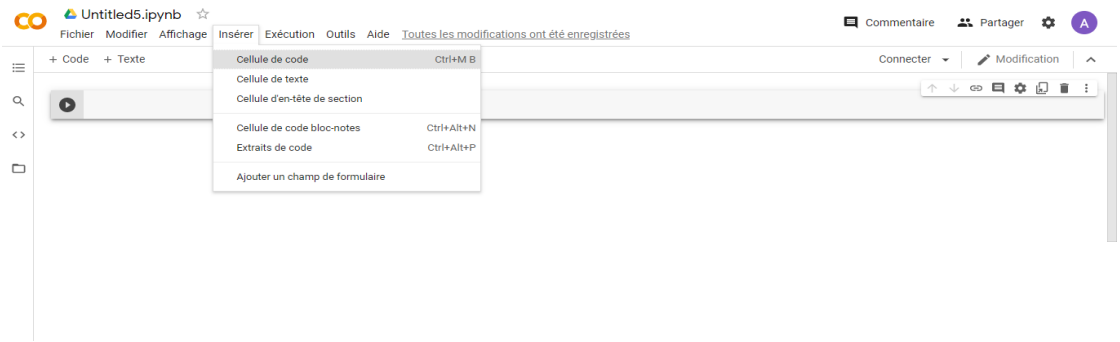


Figure II.20 : illustration pour montrer comment ajouter une cellule de code.

- Pour éxécuter une cellule particuliere , on la sélectionne et on appuie sur les touches Ctrl+ entrer ou bien l'icone correspondante , pour exécuter toutes les cellules execution->tout exécuter

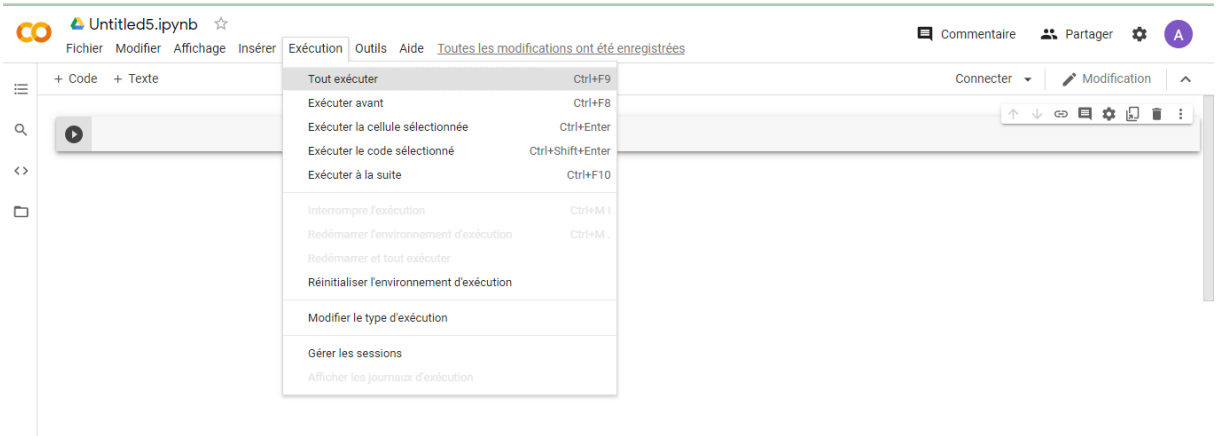


Figure II.21 : illustration pour montrer comment exécuter.

II.9.3. Installer des bibliothèques :

Comme nous savons que les bibliothèques python couramment utilisées sont préinstallées et pour les nouvelles, nous pouvons les installer en utilisant les syntaxes suivantes :

- **! pip install [nom du package]**

Où

- **! apt-get install [nom du package]**

Exemple :

```
!pip install numpy
```

```
!apt-get -qq install -y libfluidsynth1
```

Ensuite, si nous voulons l'importer, nous utilisons la fonction import comme ci-dessus :

```
import numpy as np
```

II.9.4. Importer les fichiers :

Pour importer les fichiers locaux en utilise l'écriture ci-dessus :

- **From google.colab import files**
- **Uploaded=files.upload()**

On clique sur « sélect fichiers « et on importe le fichier souhaité

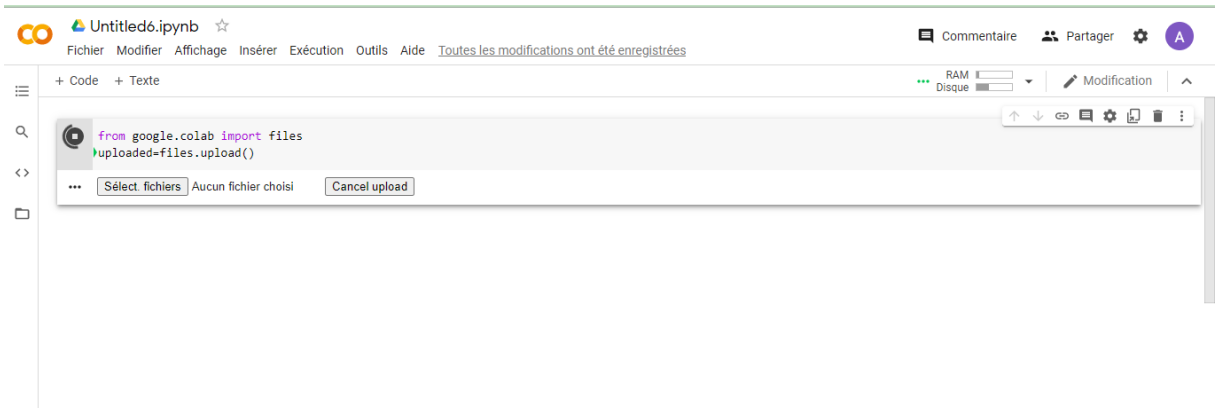


Figure II.22 : illustration comment importer les fichiers.

II.9.5. Utiliser les fichiers depuis Google drive :

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

- **From google. colab import drive**
- **Drive.mount('/content/drive')**

Ensuite, un lien sera généré, il faut cliquer sur le lien, autoriser l'accès puis copier le code qui apparait et le coller dans « enter your autorisation code »

Maintenant, pour voir toutes les données de google drive, on devra exécuter ce qui suit :

- **!ls ‘’/content/drive/Mydrive/ ‘’**

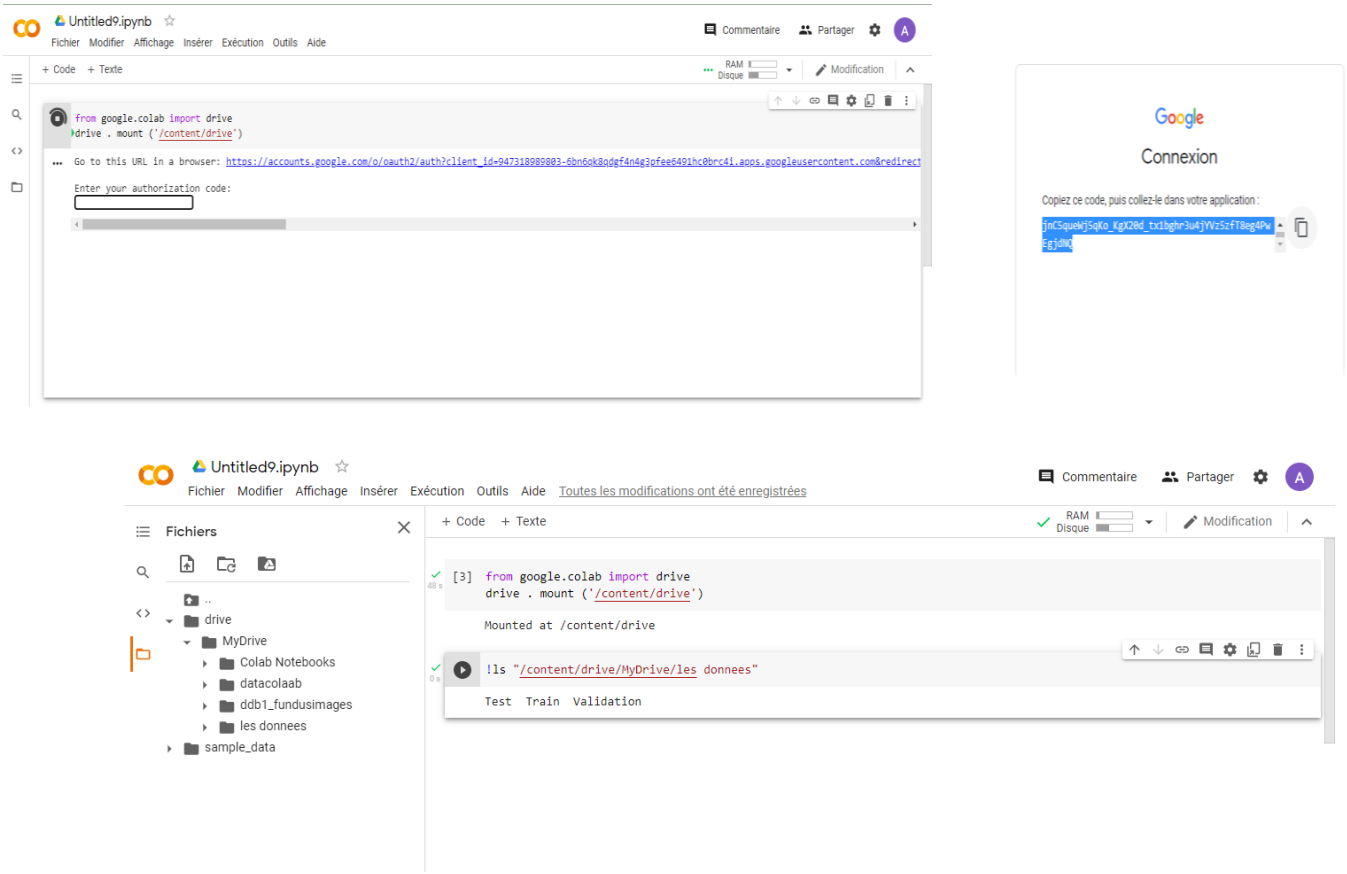


Figure II.23 : visualisation du contenu de notre Google drive.

II.9.6. Entrainer sur GPU et TPU :

GPU (de l’anglais ‘’graphics processing unit’’ ou, en français, (co)processeur graphique) est une unité de calcul (circuit intégré ou puce) présent sur une carte mère, ou encore intégré au même développé par google spécifiquement pour accélérer les systèmes d’intelligence artificielle par réseaux de neurones

Le **TPU** (de l’anglais ‘’tensor processing unit’’ ou, en français, ‘’unité de traitement de tenseur ‘’) est un circuit intégré développé par google spécifiquement pour accélérer les systèmes d’intelligence artificielle par réseaux de neurones

Pour passer en mode GPU ou TPU dans la barre des options choisir ‘’exécution ‘’puis ‘’modifier le type d’exécution ‘’et metre l’option accélérateur matériel en mode **GPU** ou bien **TPU** [27].

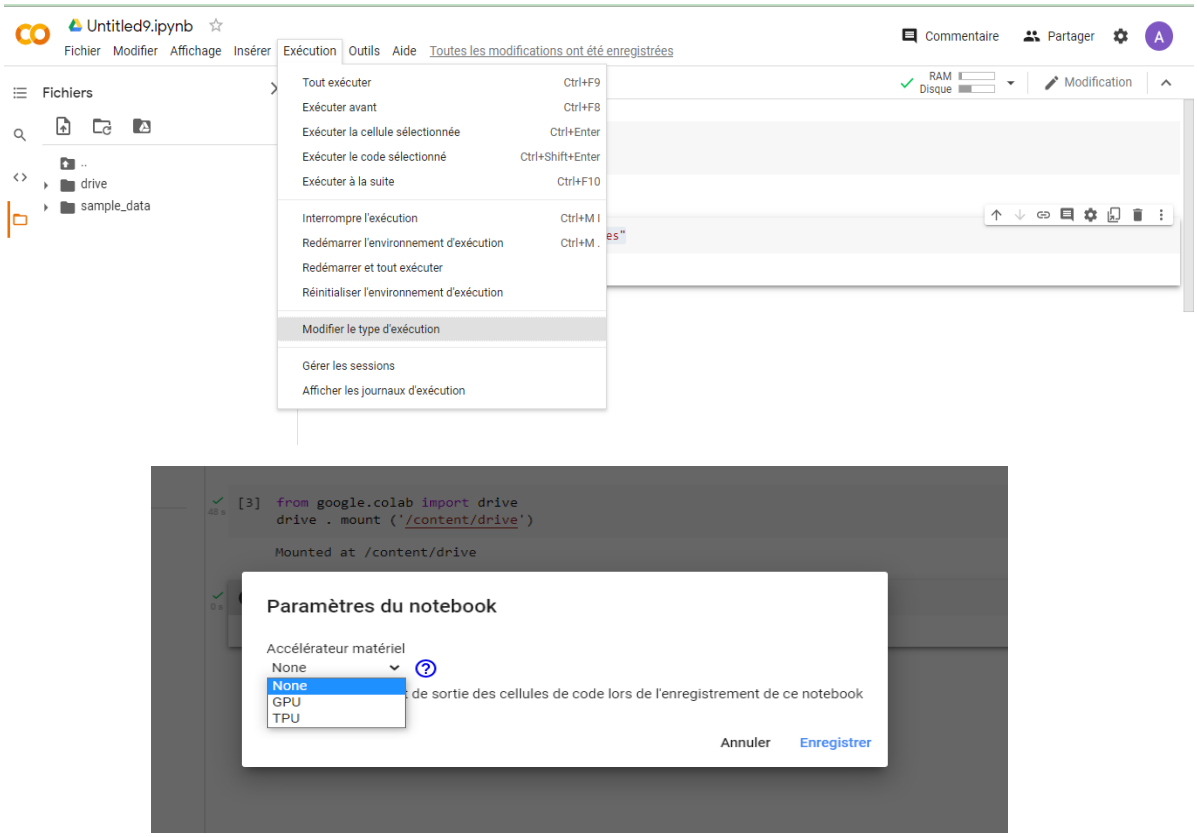


Figure II.24 : illustration pour montrer comment configurer le type d'exécution.

II.9.7. Vérifier les caractéristiques du CPU et de la RAM :

```
!cat/proc/cpuinfo
```

```
!cat/proc/meminfo
```

II.9.8. Vérifier les caractéristiques du GPU :

```
from tensorflow.python.client import device_lib
```

```
device_lib.list_local_devices()
```

II.10. Jupyter notbooks :

Les notebooks Jupyter sont des cahiers électroniques qui, dans le même document, peuvent rassembler du texte, des images, des formules mathématiques et du code informatique exécutable. Ils sont manipulables interactivement dans un navigateur web.

Initialement développés pour les langages de programmation Julia, Python et R (d'où le nom Jupyter), les notebooks Jupyter supportent près de 40 langages différents.

La cellule est l'élément de base d'un notebook Jupyter. Elle peut contenir du texte formaté au format Markdown ou du code informatique qui pourra être exécuté [30].

II.10.1. Installation :

Avec la distribution Miniconda, les notebooks Jupyter s'installent avec la commande :

```
(base) C:\Users\MNMGOLDIT Repair> conda install -y jupyterlab
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.
```

Figure II.25 : installation de jupyter notebook

II.10.2. Lancement de Jupyter et création d'un notebook :

Pour lancer les notebooks Jupyter, utilisez la commande suivante depuis un shell :

```
(base) C:\Users\MNMGOLDIT Repair> jupyter-notebook
[I 2021-09-27 12:52:25.623 LabApp] JupyterLab extension loaded from C:\Users\MNMGOLDIT Repair\anaconda3\lib\site-packages\jupyterlab
[I 2021-09-27 12:52:25.624 LabApp] JupyterLab application directory is C:\Users\MNMGOLDIT Repair\anaconda3\share\jupyter\lab
[I 12:52:25.632 NotebookApp] Serving notebooks from local directory: C:\Users\MNMGOLDIT Repair
[I 12:52:25.632 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 12:52:25.633 NotebookApp] http://localhost:8888/?token=aff9677ba0db480d58f2446267ecc4c6e080481f797c8b81
[I 12:52:25.633 NotebookApp]  or http://127.0.0.1:8888/?token=aff9677ba0db480d58f2446267ecc4c6e080481f797c8b81
[I 12:52:25.633 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:52:25.719 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/MNMGOLDIT%20Repair/AppData/Roaming/jupyter/runtime/nbserver-4452-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=aff9677ba0db480d58f2446267ecc4c6e080481f797c8b81
     or http://127.0.0.1:8888/?token=aff9677ba0db480d58f2446267ecc4c6e080481f797c8b81
```

Figure II.26: lancement de jupyter

Une nouvelle page devrait s'ouvrir dans votre navigateur web :

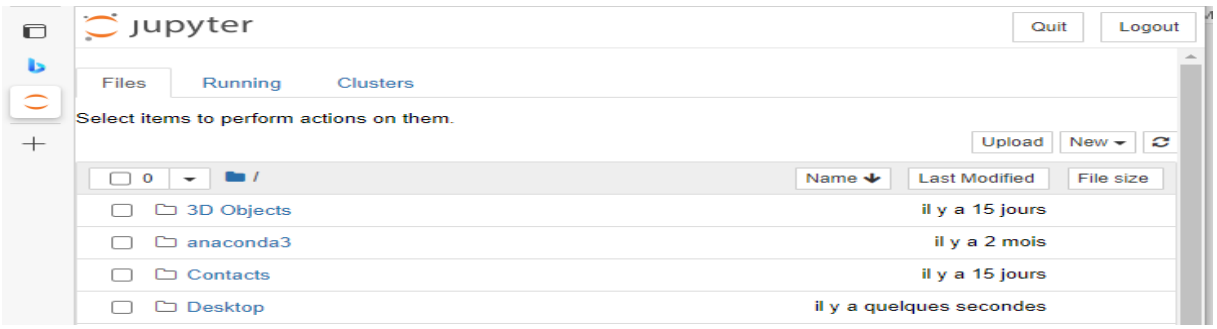


Figure II.27 : interface de jupyter

Pour créer un notebook, nous cliquons sur le bouton à droite New puis on sélectionne Python 3. Nous noterons au passage qu'il est également possible de créer un fichier texte.

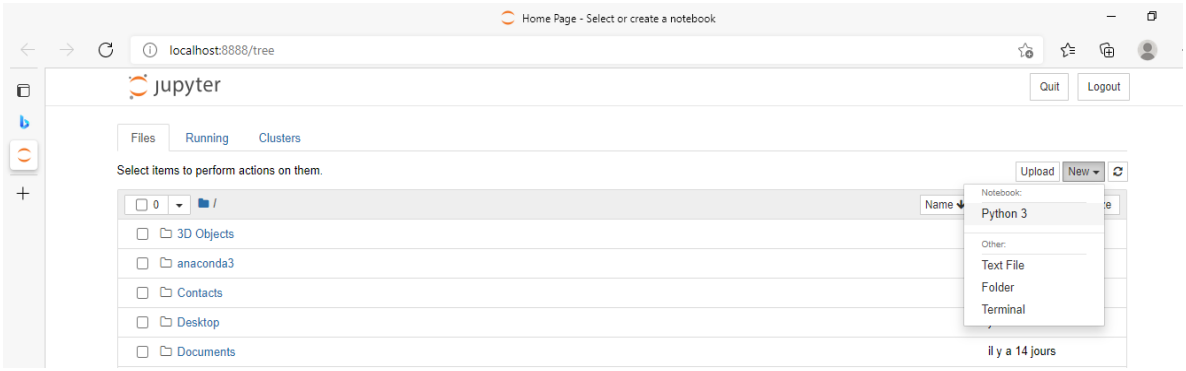


Figure II.28 : Création d'un nouveau notebook.

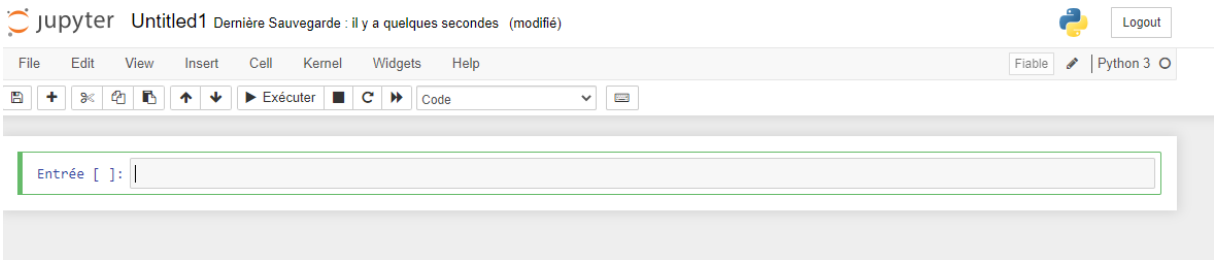


Figure II.29 : Nouveau notebook.

II.11. Conclusion :

Dans ce chapitre, nous présentons l'environnement de collaboration et comment l'utiliser, ainsi que les différentes bibliothèques de Python. 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.

Dans le prochain chapitre, nous présentons la partie pratique les étapes d'installations et à la fin les résultats obtenus et leurs discussions.

Chapitre III :

Implémentation et résultats

III.1. Introduction :

Ce chapitre est la partie application des méthodes théoriques étudiées dans les chapitres précédents. Dans un premier temps, nous préparons l'environnement de travail et installons les bibliothèques nécessaires, puis nous importons la base de données et les réseaux de neurones utilisés, Par la suite on passe à l'implémentation des différents réseaux crées. Et on parle sur les réseaux de neurones modifiés, pour faire la classification des images et la détection de la rétinopathie diabétique.

Enfin, nous explorons ensemble des étapes d'implémentation et discutons des résultats obtenus.

III.2. Les étapes de classification des images de fond d'œil :

Nous étudions les étapes de travail :

Au début nous préparons l'environnement de travail, ensuite nous divisons et importons la base de données, après nous appliquons l'augmentation de données et le transfert Learning. À la fin, nous faisons l'apprentissage

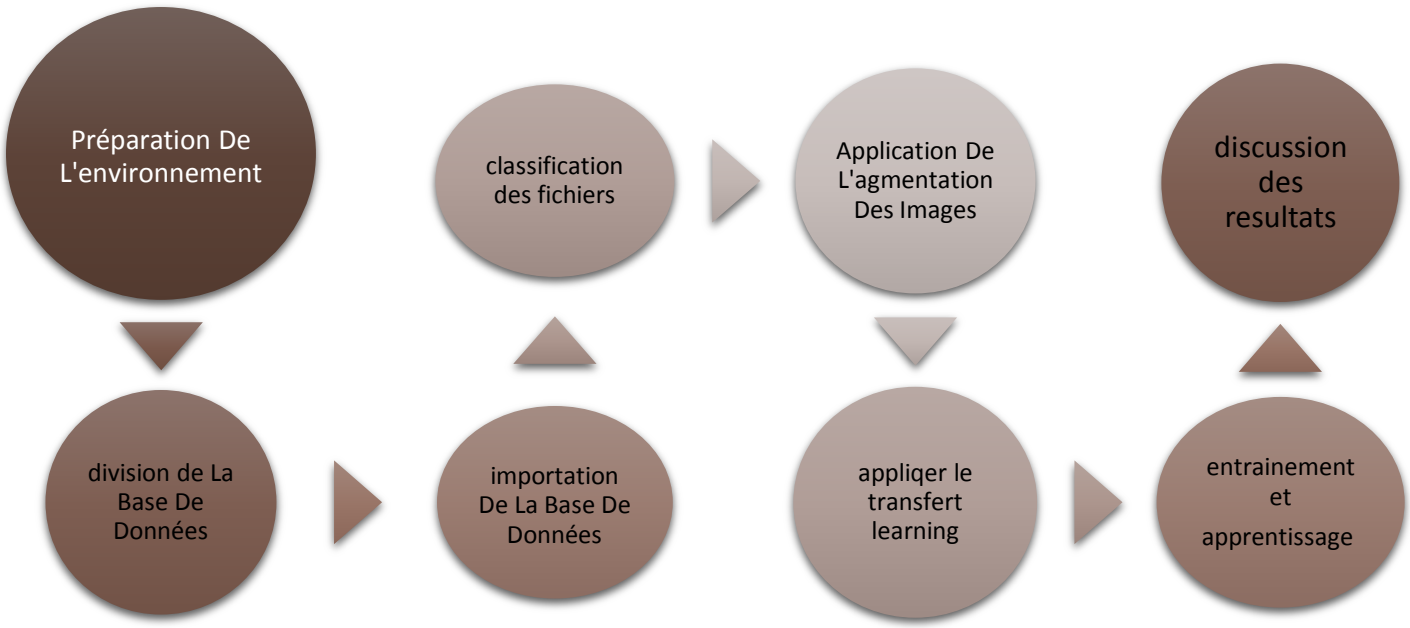


Figure III.1 : Les étapes de classification des images de fond d'œil.

III.2.1. Préparer l'environnement :

Nous allons d'abord importer la bibliothèque keras avec le backend tensorflow pour préparer l'environnement de travail, et on va utiliser le modèle vgg16 pour la classification.

```
# Import Keras with tensorflow backend
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras import optimizers
from tensorflow.keras.applications import VGG16
```

Figure III.2 : code d'importation de keras avec tensorflow.

III.2.2. Importation des bibliothèques :

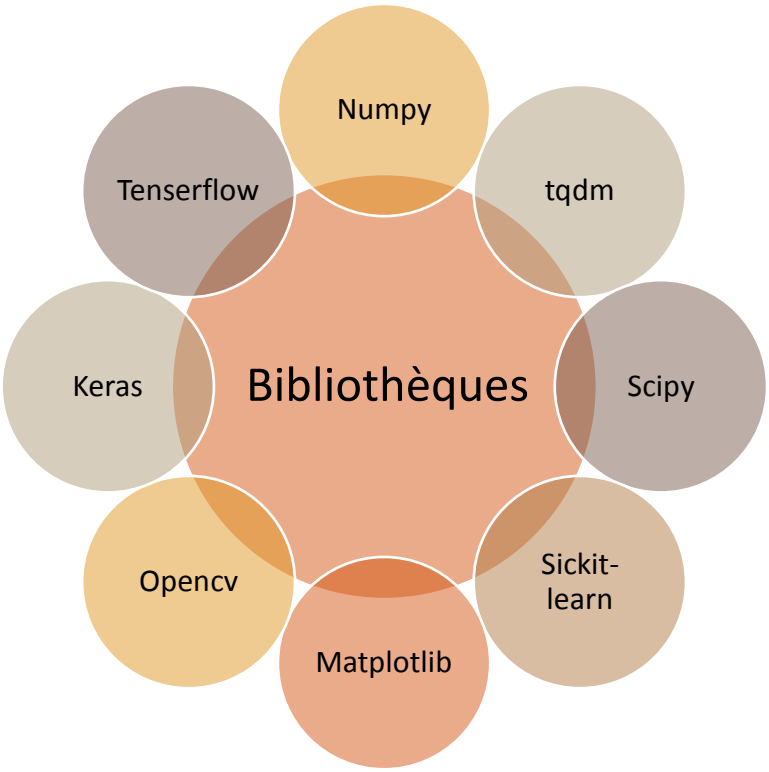


Figure III.3 : les bibliothèques installées.

Dans cette partie nous importerons toutes les bibliothèques nécessaires, et la Matrice de confusion et rapport de classification.

```
# Import OpenCV
import cv2

# Utility
import os
import numpy as np
import itertools
import random
from collections import Counter
from glob import iglob
# Ignore warning
import warnings
warnings.filterwarnings('ignore')

# Confusion Matrix & classification report
from sklearn.metrics import confusion_matrix, classification_report

# Plot
import matplotlib.pyplot as plt
%matplotlib inline
```

Figure III.4 : importation des bibliothèques.

III.2.3. Diviser de base de données :

Pour diviser la base de données, il existe deux techniques, l'une utilisant la programmation et l'autre manuelle.

Dans notre base de données, il y a 5000 images avec la méthode manuelle utilisée, nous diviserons les images en 20% pour le test et 80% pour train et validation.

a) La méthode manuelle :

Train +val = 80%

- 0-pas RD = 800 images
- 1-légère = 800 images
- 2-modérée = 800 images
- 3-grave = 800 images
- 4- RD proliférant = 800 images

Test = 20%

b) La méthode de programmation :

A l'aide de jupyter notebook on divise la base de données :

```
Entrée [8]: import pandas as pd
            import os
            import shutil

Entrée [2]: #rethinopathie0
            file_path = "archive/train.csv"

Entrée [3]: images_path = "archive/gaussian_filtered_images"

Entrée [4]: df = pd.read_csv(file_path)

Entrée [5]: print(df.shape)

            (3662, 2)
```

```
Entrée [6]: df.head()
```

Out[6]:

	id_code	diagnosis
0	000c1434d8d7	2
1	001639a390f0	4
2	0024cdab0c1e	1
3	002c21358ce6	0
4	005b95c28852	0

```
Entrée [7]: target_fold = "rethinopathie0/rethino0"

Entrée [9]: if not os.path.exists(target_fold):
                os.mkdir(target_fold)
                print("covid folder created")

            covid folder created
```

```
Entrée [15]: counter = 0
             for (i,row) in df.iterrows():
                 if row["diagnosis"] == "0":
                     filename = row["filename"]
                     img_p = os.path.join(images_path,filename)
                     img_cp_p = os.path.join(target_fold,filename)
                     shutil.copy2(img_p,img_cp_p)
                     print("img copy",counter)
                     counter+=1
             print(counter)
```

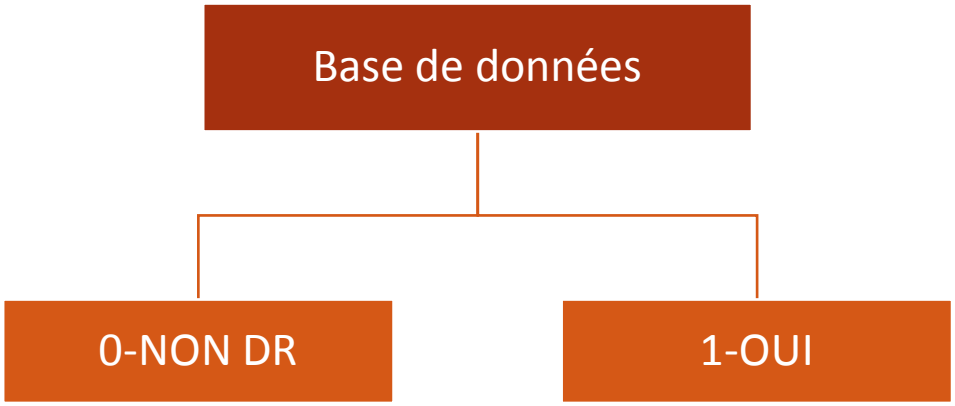
Figure III.5 : programme pour diviser la base de donnée

- Dans notre cas nous avons essayé de diviser la base de données selon trois répartitions (deux classes, trois classes et cinq classes) afin de savoir laquelle donnera le meilleur résultat.

A. Sur deux classes :

La division de la base de donnée sur deux dans drive :

Figure III.6: la division de la base de données sur deux dans le drive



B. Sur trois classes

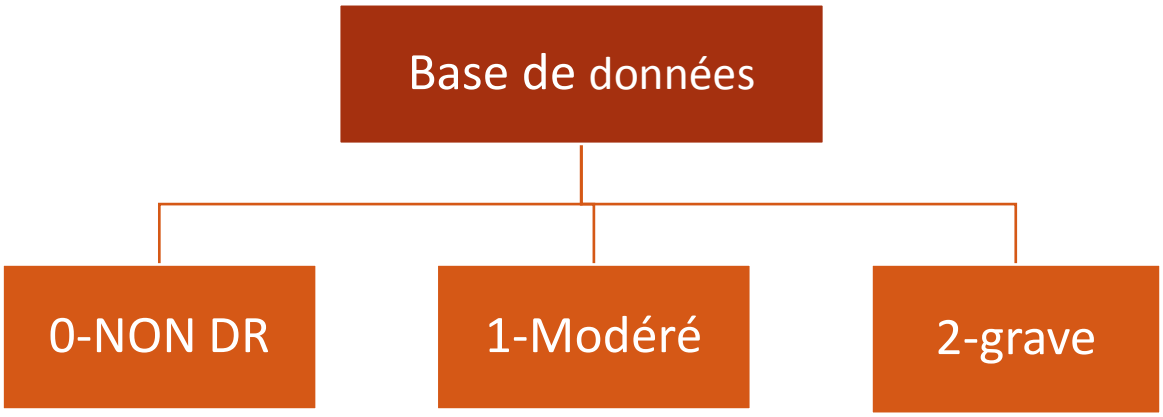


Figure III. 7 : illustration de la division de la base de donnée sur trois classes.

La base de donnée divisé dans drive :

Figure III.8 : la division de la base de donnée sur trois classes

C. Sur cinq classes :

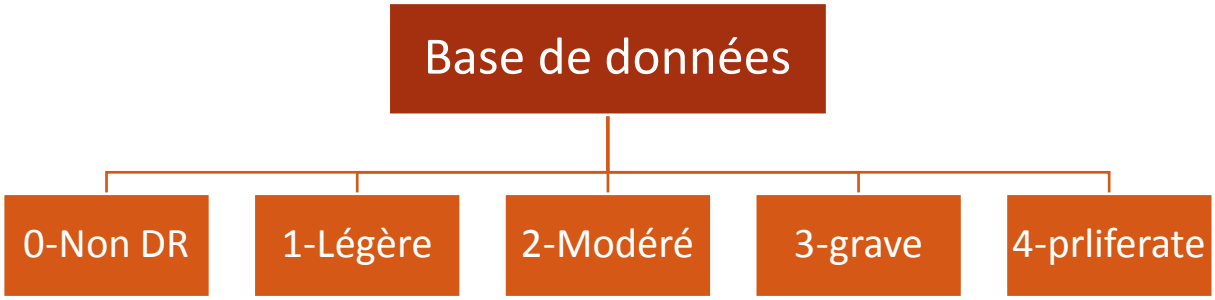


Figure III.9: illustration de la division de la base de donnée sur cinq classes

La base de données divisé dans drive :

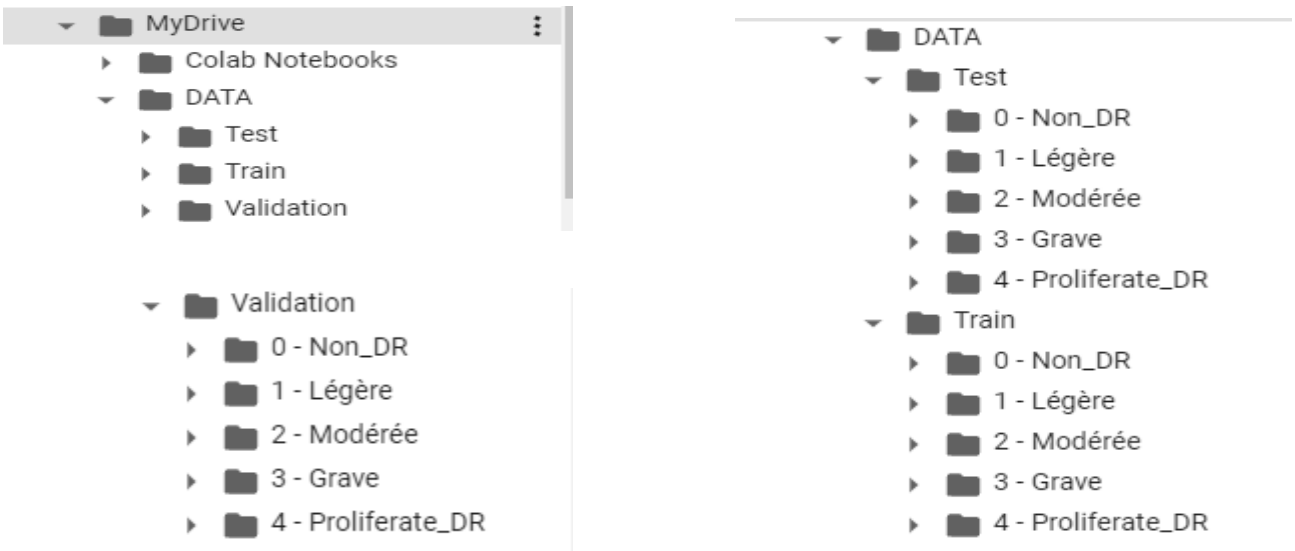


Figure III.10 : la division de la base de donnée sur cinq classes.

✓ **Des exemples :**

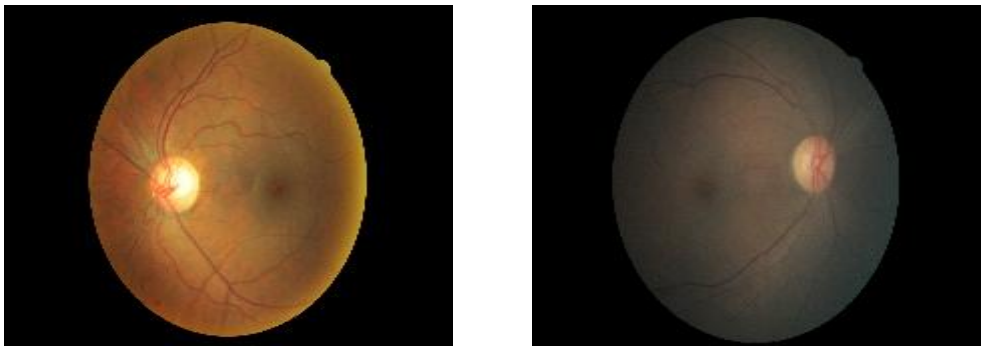


Figure : 0-non_DR

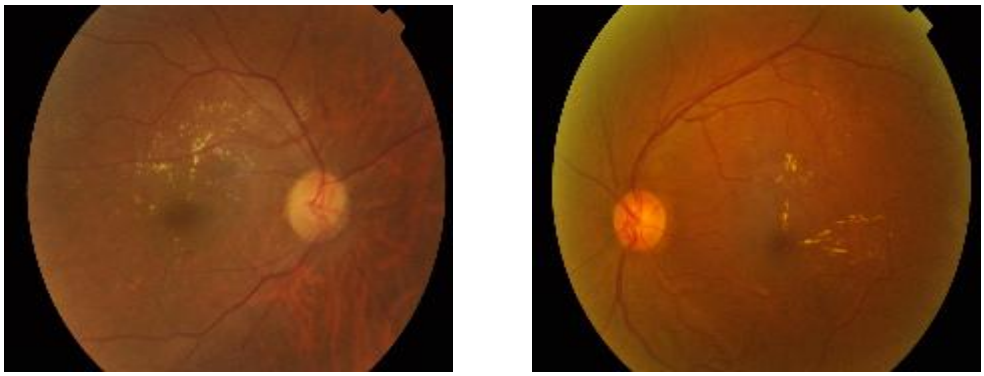


Figure : 1-légère

Figure : 2-modérée

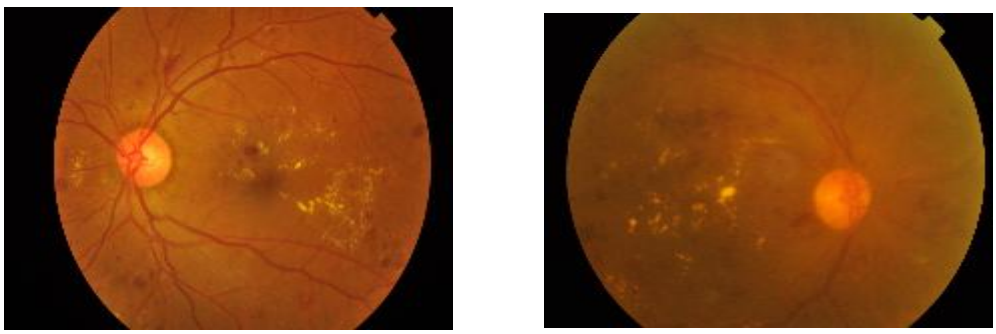


Figure : 3-grave

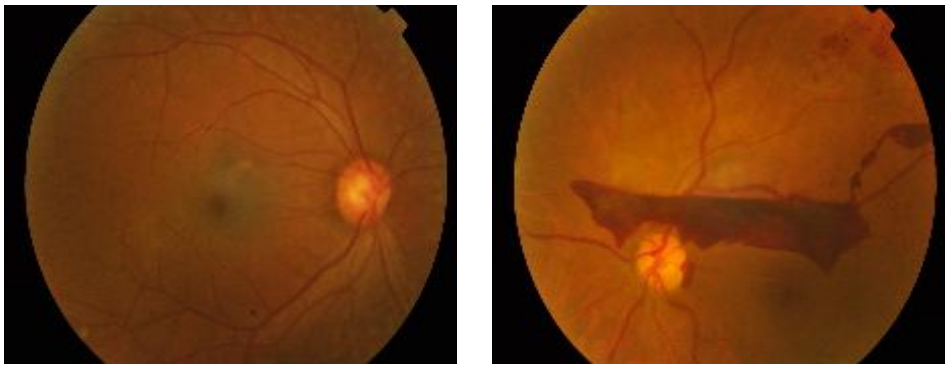


Figure : 4-proliférante_DR

Figure III.11 : des exemples sur la base de donnée.

III.2.4 Importation de la base de donnée :

Après avoir divisé notre base de donnée, nous la stockons dans un dossier dans Google drive et nous l'importons-la dans colab comme nous avons déjà expliqué dans le chapitre2

```
# Définir le chemin du dossier de l'ensemble de données
BASE_DATASET_FOLDER = os.path.join("/content/drive/MyDrive/data")
TRAIN_FOLDER = "Train"
VALIDATION_FOLDER = "Validation"
TEST_FOLDER = "Test"
```

Figure II.12: importation de la base de données.

Remarque :

Au début de notre travail sur le projet nous utilisions une base de données de 89 images.

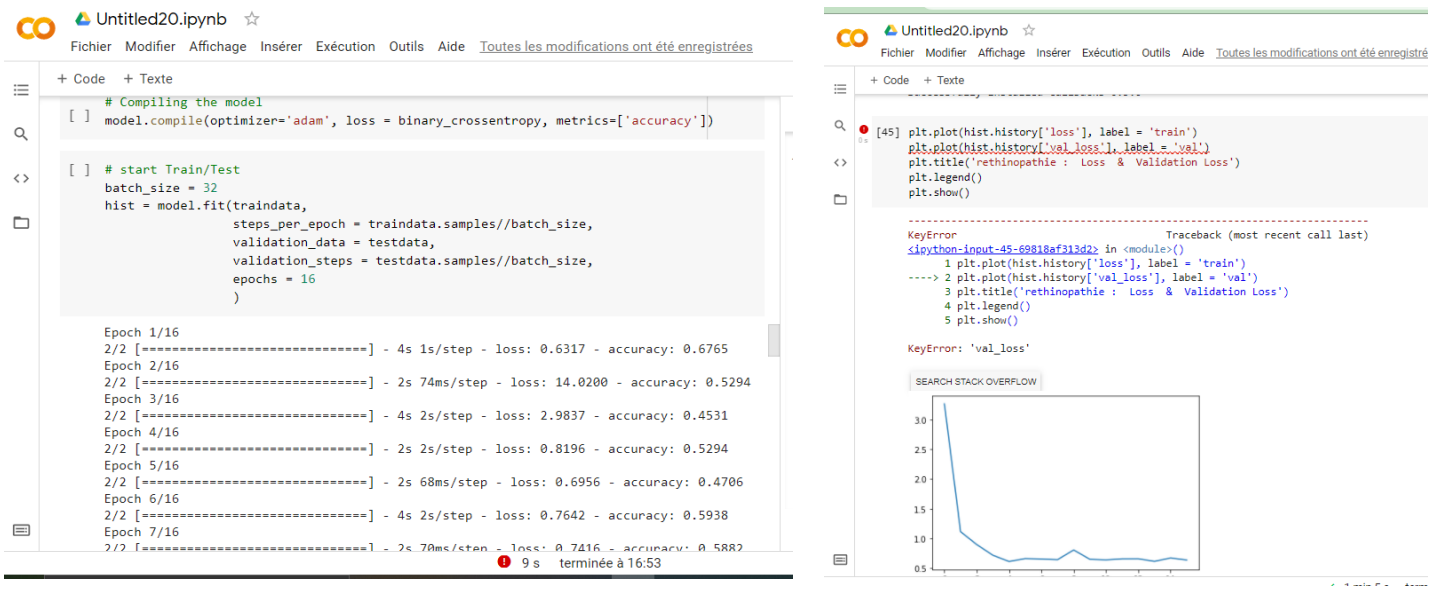


Figure II.13 : l'affichage de l'erreur.

Nous avons constaté que notre base de donnée était trop petite pour un apprentissage correct.

Pour corriger cette erreur, nous avons remplacé l'ancienne base de données par une plus grande contenant 3680 images.

L'erreur a été résolue mais l'apprentissage n'est pas bon car la base de données utilisée était mal divisée.

```
%time
history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
        validation_data=val_generator,
        validation_steps=val_generator.samples//val_generator.batch_size
        )
```

```
29/29 [==============================] - 43s 1s/step - loss: 0.4534 - acc: 0.8232 - val_loss: 0.7960 - val_acc: 0.7795
Epoch 109/120
29/29 [==============================] - 44s 1s/step - loss: 0.4267 - acc: 0.8409 - val_loss: 0.8111 - val_acc: 0.7448
Epoch 110/120
29/29 [==============================] - 44s 1s/step - loss: 0.4412 - acc: 0.8361 - val_loss: 0.6681 - val_acc: 0.7795
Epoch 111/120
29/29 [==============================] - 44s 2s/step - loss: 0.4253 - acc: 0.8325 - val_loss: 0.6801 - val_acc: 0.7760
Epoch 112/120
29/29 [==============================] - 44s 2s/step - loss: 0.4230 - acc: 0.8312 - val_loss: 0.9231 - val_acc: 0.7674
Epoch 113/120
29/29 [==============================] - 45s 2s/step - loss: 0.4155 - acc: 0.8440 - val_loss: 0.7768 - val_acc: 0.7483
Epoch 114/120
29/29 [==============================] - 44s 2s/step - loss: 0.4143 - acc: 0.8392 - val_loss: 0.7222 - val_acc: 0.7795
Epoch 115/120
29/29 [==============================] - 44s 2s/step - loss: 0.3882 - acc: 0.8475 - val_loss: 0.7559 - val_acc: 0.7448
Epoch 116/120
29/29 [==============================] - 45s 2s/step - loss: 0.4135 - acc: 0.8401 - val_loss: 0.7565 - val_acc: 0.7587
Epoch 117/120
29/29 [==============================] - 44s 2s/step - loss: 0.4205 - acc: 0.8422 - val_loss: 0.7608 - val_acc: 0.7674
Epoch 118/120
29/29 [==============================] - 44s 2s/step - loss: 0.3780 - acc: 0.8568 - val_loss: 0.8727 - val_acc: 0.7691
Epoch 119/120
29/29 [==============================] - 44s 2s/step - loss: 0.3911 - acc: 0.8489 - val_loss: 0.8251 - val_acc: 0.7795
Epoch 120/120
29/29 [==============================] - 44s 2s/step - loss: 0.3912 - acc: 0.8436 - val_loss: 0.9743 - val_acc: 0.7552
```

Figure III.14: exécution de l'apprentissage.

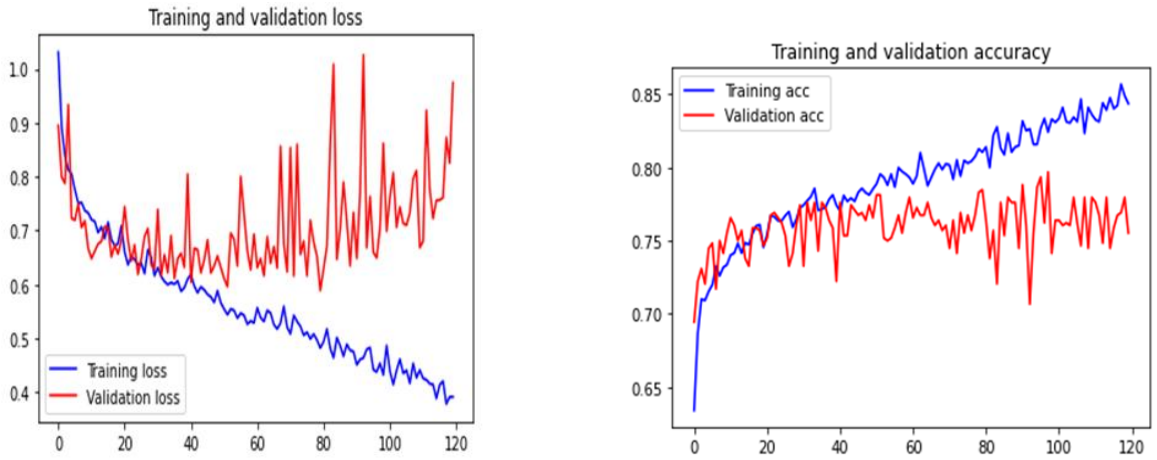


Figure III.15 : les graphes de loss et acc fonction.

Après toute ces recherches, nous avons finalement choisi la bonne base de données.

III.2.5 Classification des fichiers :

Dans un premier temps, nous devons classer notre dossier de base de données en classe numérotés pour faciliter l'apprentissage

```
[ ]  def percentage_value(pct, allvals):
         absolute = int(pct/100.*np.sum(allvals))
         return "{:.1f}%\n({:d})".format(pct, absolute)

     def plot_dataset_description(path, title):
         classes = []
         for filename in iglob(os.path.join(path, "**","*.png")):
             classes.append(os.path.split(os.path.split(filename)[0])[-1])

         classes_cnt = Counter(classes)
         values = list(classes_cnt.values())
         labels = list(classes_cnt.keys())

         plt.figure(figsize=(8,8))
         plt.pie(values, labels=labels, autopct=lambda pct: percentage_value(pct, values),
                 shadow=True, startangle=140)

         plt.title(title)
         plt.show()
```

Figure III.16 : code de classification des classes.

Après avoir classé les dossiers de la base de données, nous voulons maintenant les afficher :

Sur deux classes :

```
[ ]  classes = {v: k for k, v in train_generator.class_indices.items()}
     print(classes)

     {0: 'Non', 1: 'Oui'}
```

Sur trois classes :

```
[ ]  classes = {v: k for k, v in train_generator.class_indices.items()}
     print(classes)

     {0: 'Non', 1: 'greve', 2: 'modere'}
```

Sur cinq classes :

```
[7]  classes = {v: k for k, v in train_generator.class_indices.items()}
     print(classes)

     {0: '0 - Non_DR', 1: '1 - Légère', 2: '2 - Modérée', 3: '3 - Grave', 4: '4 - Proliferate_DR'}
```

Figure III.17: code d'affichage des classes

Maintenant, nous voulons afficher le nombre d'images dans chaque dossier :

```
[ ]  plot_dataset_description(os.path.join(BASE_DATASET_FOLDER, TRAIN_FOLDER), "Train folder description")

     plot_dataset_description(os.path.join(BASE_DATASET_FOLDER, TEST_FOLDER), "Test folder description")

     plot_dataset_description(os.path.join(BASE_DATASET_FOLDER, VALIDATION_FOLDER), "Validation folder description")
```

Sur deux classes :

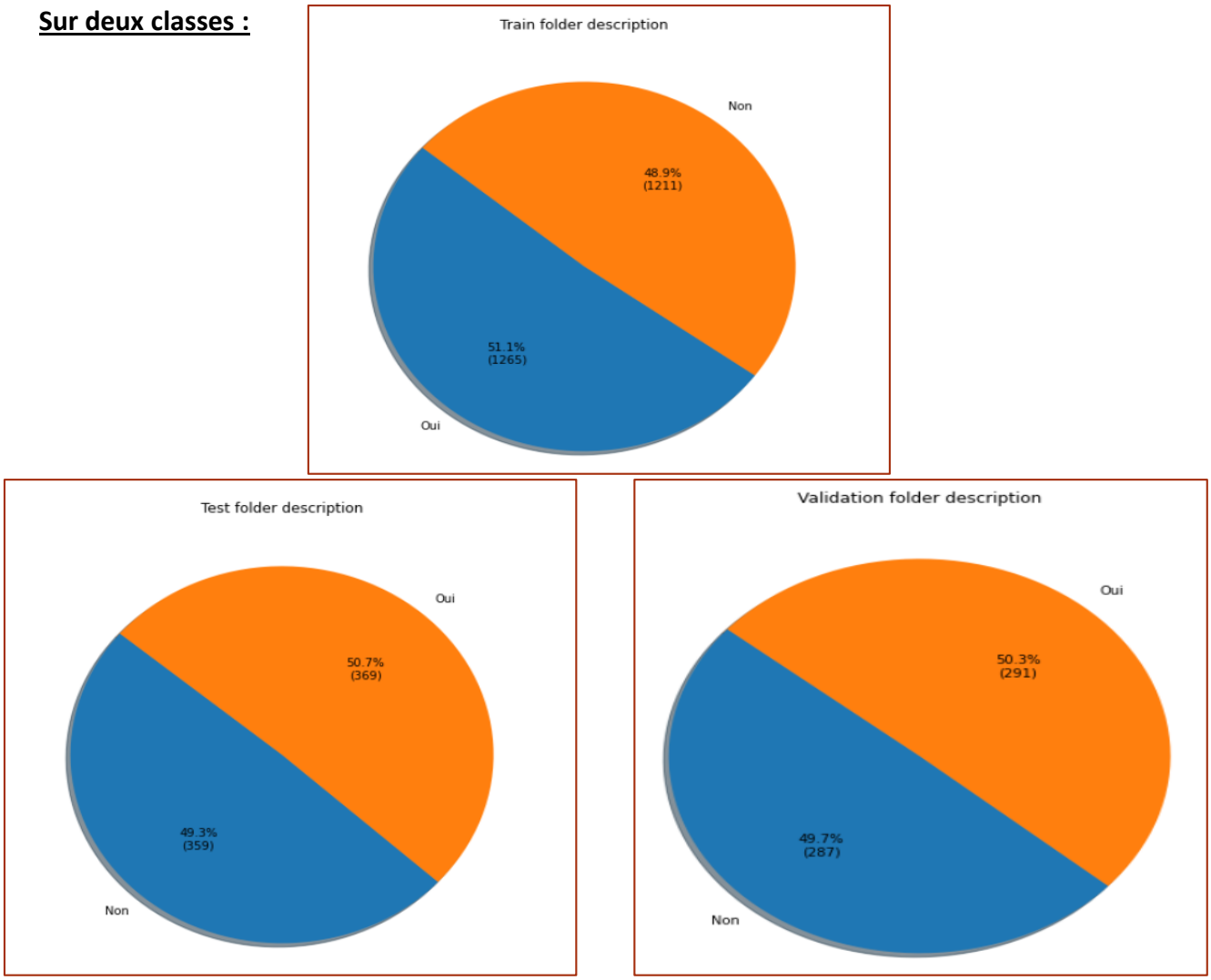
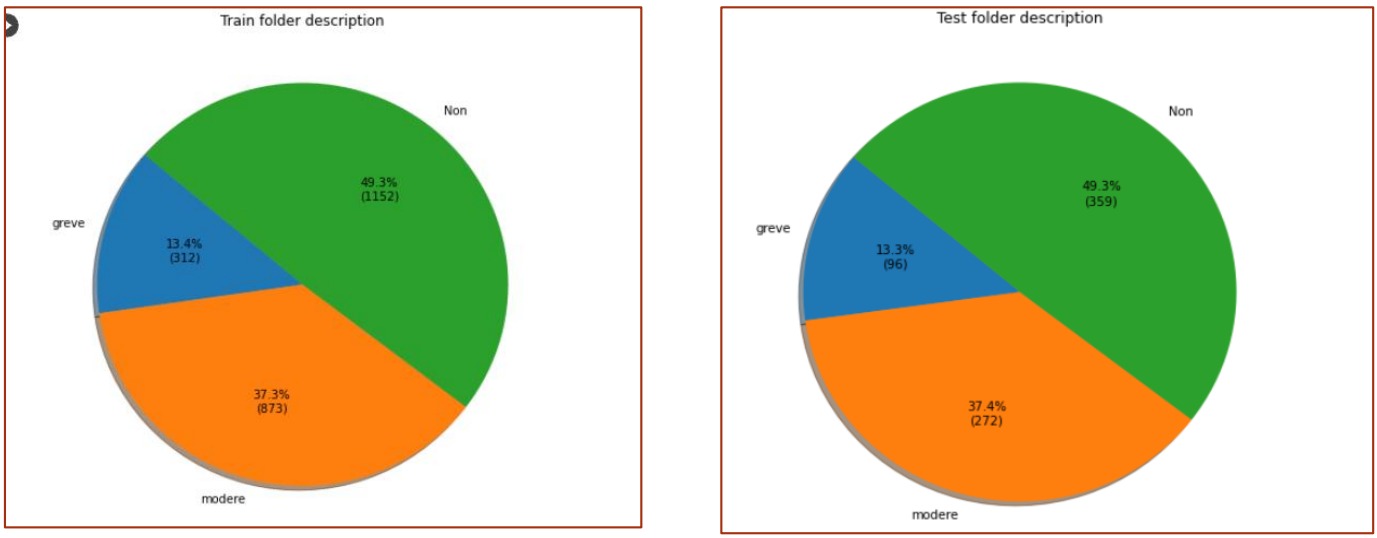


Figure III.18 : description des dossiers de deux classes

Sur trois classes :



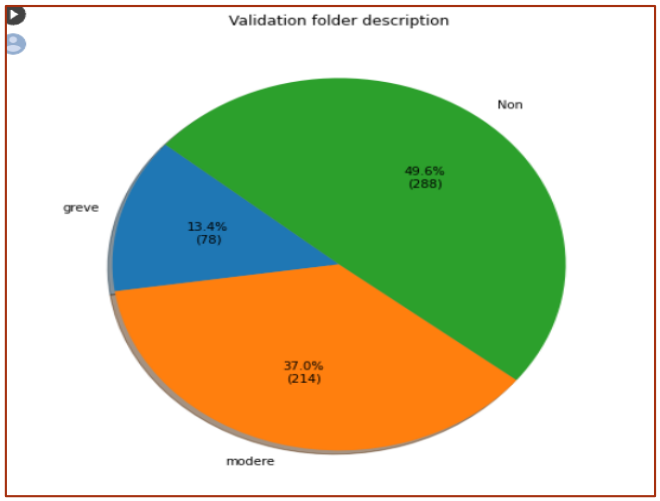
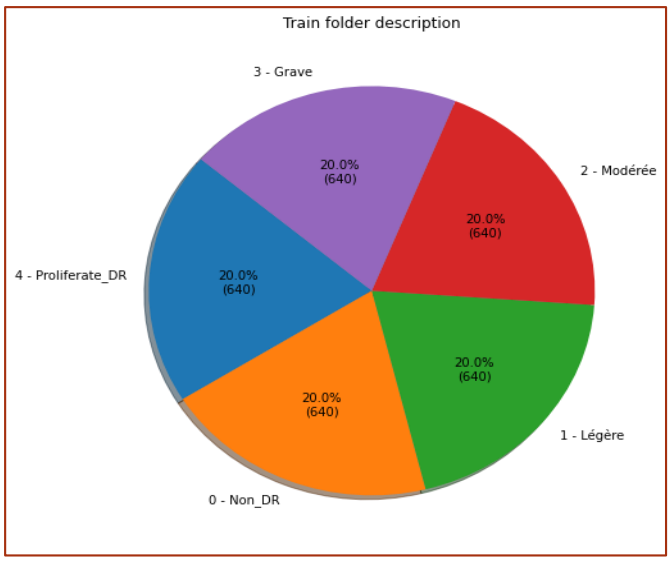
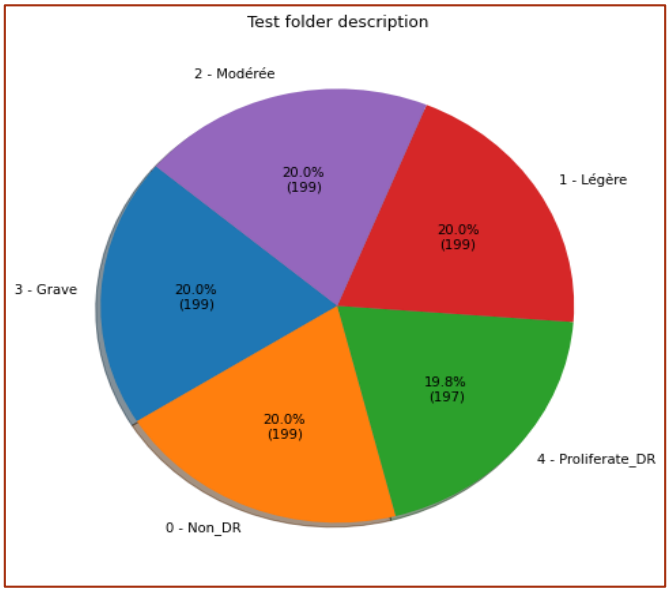


Figure III.19: description des dossiers de trois classes

Sur cinq classes :





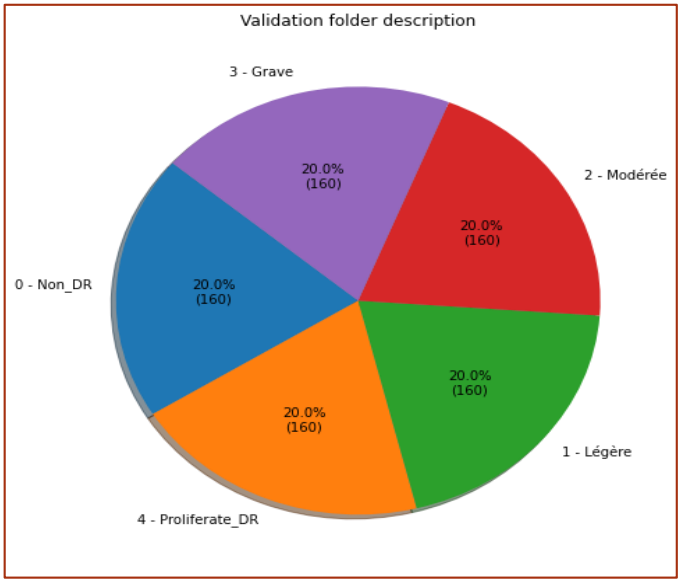


Figure III.20 : description des dossiers de cinq classes

II.2.6 Augmentation de data :

Nous savons que dans le deep Learning, plus la base de données est grande plus l'apprentissage est meilleur et précis.

C'est pourquoi nous avons augmenté la base de données pour obtenir les meilleurs résultats.

Nous avons appliqué toutes sortes d'augmentation de données qui ont été précédemment étudiées :

- La rotation
- Redimensionner
- Flip vertical
- Flip horizontal
- Décalage de hauteur
- Modifier la luminosité

En plus on a utilisé la fonction « **categorical** » pour la base de données divisée sur 5 et 3 mais par contre on utilise la fonction « **binary »** pour la base de donnée diviser sur 2.

```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
        os.path.join(BASE_DATASET_FOLDER, TEST_FOLDER),
        target_size=IMAGE_SIZE,
        batch_size=VAL_BATCH_SIZE,
        class_mode='categorical',
        shuffle=False)


val_datagen = ImageDataGenerator(rescale=1./255)
val_generator = val_datagen.flow_from_directory(
        os.path.join(BASE_DATASET_FOLDER, VALIDATION_FOLDER),
        target_size=IMAGE_SIZE,
        class_mode='categorical',
        shuffle=False)
```

```
train_datagen = ImageDataGenerator(
    rotation_range=15,
        shear_range=0.05,
        rescale=1./255,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest')


train_generator = train_datagen.flow_from_directory(
        os.path.join(BASE_DATASET_FOLDER, TRAIN_FOLDER),
        target_size=IMAGE_SIZE,
        batch_size=TRAIN_BATCH_SIZE,
        class_mode='categorical',
        shuffle=True)
```

Figure III.21 : Programme d'augmentation de data

Sur deux classes :

```
Found 2477 images belonging to 2 classes.
Found 730 images belonging to 2 classes.
Found 580 images belonging to 2 classes.
```

Sur trois classes :

```
Found 2338 images belonging to 3 classes.
Found 720 images belonging to 3 classes.
Found 581 images belonging to 3 classes.
```

Sur cinq classes :

```
Found 3200 images belonging to 5 classes.
Found 998 images belonging to 5 classes.
Found 800 images belonging to 5 classes.
```

II.2.7 Application du transfert Learning :

Tout d'abord, nous devons réduire la taille des images à 224. Il faut travailler dans le domaine RVB car nos images sont en couleur.

```
IMAGE_SIZE = (224, 224)
INPUT_SHAPE = (224, 224, 3)
```

Après nous importons le réseau vgg16 et lui demandons d'enlever les quatre dernières couches, et les changer avec nos propres couches.

```
[ ] vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=INPUT_SHAPE)
    for layer in vgg_model.layers[:-4]:
        layer.trainable = False

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
    58892288/58889256 [==============================] - 1s 0us/step
    58900480/58889256 [==============================] - 1s 0us/step
```

Figure III.22 : modification du programme VGG16

Maintenant, nous devons recréer notre propre réseau :

```
# Create the model
model = Sequential()

# Add the vgg convolutional base model
model.add(vgg_model)

# Add new layers
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(classes), activation='softmax'))
```

Figure III.23 : création de modèle

Nous voulons maintenant afficher un résumé du modèle et vérifier combien de paramètres peuvent être entrainés.

```
# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
=================================================================
Total params: 21,171,013
Trainable params: 13,535,749
Non-trainable params: 7,635,264
_________________________________________________________________
```

Figure III.24: visualisation du résumé du modèle

Maintenant nous allons faire la même tache mais nous changeons le réseau au lieu de VGG16 nous mettons **alexnet** :

Tout d'abord, nous importons les bibliothèques nécessaires :

```
from __future__ import division, print_function, absolute_import

import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.normalization import local_response_normalization
from tflearn.layers.estimator import regression
```

Figure III.25: importation des bibliothèques pour le réseau Alex net

Ensuite l'implémentation de réseau :

```
network = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 1], name='input')
network = conv_2d(network, 96, 11, strides=4, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 256, 5, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 96, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = conv_2d(network, 96, 3, activation='relu')
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network,5, activation='softmax')
network = regression(network, optimizer='adam', learning_rate=LR, loss='categorical_crossentropy', name='targets')
```

Figure III.26 : architecture de réseau alexnet

A la fin on affiche les Résultats de l'exécution :

```
# Training
model.fit({'input': X}, {'targets': Y}, n_epoch=50, validation_set=({'input': test_x}, {'targets': test_y}),
    snapshot_step=500, show_metric=True, run_id=MODEL_NAME)
```

```
Training Step: 649  | total loss: 1.69607 | time: 1.426s
| Adam | epoch: 050 | loss: 1.69607 - acc: 0.2115 -- iter: 768/822
Training Step: 650  | total loss: 1.69904 | time: 2.542s
| Adam | epoch: 050 | loss: 1.69904 - acc: 0.2153 | val_loss: 1.64493 - val_acc: 0.1505 -- iter: 822/822
--
```

Figure III.27: résultats de Alex net

Après avoir travaillé avec les deux réseaux, **vgg16** et **alex net** nous allons maintenant implémenter l'architecture du réseau inception _v3 :

```
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression

network = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 1], name='input')


conv1_7_7 = conv_2d(network, 64, 7, strides=2, activation='relu', name='conv1_7_7_s2')
pool1_3_3 = max_pool_2d(conv1_7_7, 3, strides=2)
pool1_3_3 = local_response_normalization(pool1_3_3)
conv2_3_3_reduce = conv_2d(pool1_3_3, 64, 1, activation='relu', name='conv2_3_3_reduce')
conv2_3_3 = conv_2d(conv2_3_3_reduce, 192, 3, activation='relu', name='conv2_3_3')
conv2_3_3 = local_response_normalization(conv2_3_3)
pool2_3_3 = max_pool_2d(conv2_3_3, kernel_size=3, strides=2, name='pool2_3_3_s2')

# 3a
inception_3a_1_1 = conv_2d(pool2_3_3, 64, 1, activation='relu', name='inception_3a_1_1')
inception_3a_3_3_reduce = conv_2d(pool2_3_3, 96, 1, activation='relu', name='inception_3a_3_3_reduce')
inception_3a_3_3 = conv_2d(inception_3a_3_3_reduce, 128, filter_size=3,  activation='relu', name='inception_3a_3_3')
inception_3a_5_5_reduce = conv_2d(pool2_3_3, 16, filter_size=1, activation='relu', name='inception_3a_5_5_reduce')
inception_3a_5_5 = conv_2d(inception_3a_5_5_reduce, 32, filter_size=5, activation='relu', name='inception_3a_5_5')
inception_3a_pool = max_pool_2d(pool2_3_3, kernel_size=3, strides=1, name='inception_3a_pool')
inception_3a_pool_1_1 = conv_2d(inception_3a_pool, 32, filter_size=1, activation='relu', name='inception_3a_pool_1_1')
inception_3a_output = merge([inception_3a_1_1, inception_3a_3_3, inception_3a_5_5, inception_3a_pool_1_1], mode='concat', axis=3)
```

```
# 4d
inception_4d_1_1 = conv_2d(inception_4c_output, 112, filter_size=1, activation='relu', name='inception_4d_1_1')
inception_4d_3_3_reduce = conv_2d(inception_4c_output, 144, filter_size=1, activation='relu', name='inception_4d_3_3_reduce')
inception_4d_3_3 = conv_2d(inception_4d_3_3_reduce, 288, filter_size=3, activation='relu', name='inception_4d_3_3')
inception_4d_5_5_reduce = conv_2d(inception_4c_output, 32, filter_size=1, activation='relu', name='inception_4d_5_5_reduce')
inception_4d_5_5 = conv_2d(inception_4d_5_5_reduce, 64, filter_size=5,  activation='relu', name='inception_4d_5_5')
inception_4d_pool = max_pool_2d(inception_4c_output, kernel_size=3, strides=1,  name='inception_4d_pool')
inception_4d_pool_1_1 = conv_2d(inception_4d_pool, 64, filter_size=1, activation='relu', name='inception_4d_pool_1_1')
inception_4d_output = merge([inception_4d_1_1, inception_4d_3_3, inception_4d_5_5, inception_4d_pool_1_1], mode='concat', axis=3, name='inception_4d_output')

# 4e
inception_4e_1_1 = conv_2d(inception_4d_output, 256, filter_size=1, activation='relu', name='inception_4e_1_1')
inception_4e_3_3_reduce = conv_2d(inception_4d_output, 160, filter_size=1, activation='relu', name='inception_4e_3_3_reduce')
inception_4e_3_3 = conv_2d(inception_4e_3_3_reduce, 320, filter_size=3, activation='relu', name='inception_4e_3_3')
inception_4e_5_5_reduce = conv_2d(inception_4d_output, 32, filter_size=1, activation='relu', name='inception_4e_5_5_reduce')
inception_4e_5_5 = conv_2d(inception_4e_5_5_reduce, 128,  filter_size=5, activation='relu', name='inception_4e_5_5')
inception_4e_pool = max_pool_2d(inception_4d_output, kernel_size=3, strides=1,  name='inception_4e_pool')
inception_4e_pool_1_1 = conv_2d(inception_4e_pool, 128, filter_size=1, activation='relu', name='inception_4e_pool_1_1')
inception_4e_output = merge([inception_4e_1_1, inception_4e_3_3, inception_4e_5_5, inception_4e_pool_1_1], axis=3, mode='concat')
pool4_3_3 = max_pool_2d(inception_4e_output, kernel_size=3, strides=2, name='pool_3_3')
```

Figure III.28 : architecture de réseau inception_V3

Les résultats de l'exécution :

```
Training Step: 50  | total loss: 1.23090 | time: 4.749s
| Adam | epoch: 050 | loss: 1.23090 - acc: 0.5582 | val_loss: 2.02821 - val_acc: 0.3840 -- iter: 28/28
--
```

Figure III.29 : exécution de inception_v3

II.2.8 Entrainement :

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

On utilise model. Compile () car il prend l'optimiseur la perte (loss) et la métrique (metrics) comme paramètres

Tout d'abord, nous devons choisir les paramètres de keras, nous prenons 80 image de train batch et 15 images de validation batch pour faire une 120 époque.

```
# Keras paramètres
TRAIN_BATCH_SIZE = 80
VAL_BATCH_SIZE = 15
EPOCHS = 120
LEARNING_RATE = 0.0001
MODEL_PATH = os.path.join("rétinophatie_diabithique_detection.h5")
MODEL_WEIGHTS_PATH = os.path.join("model_weights.h5")
```

Figure III.30 : keras paramètres

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(lr=LEARNING_RATE),
              metrics=['acc'])
```

Figure III.31 : compilation du modèle

On exécute avec le **modèle.fit_generator()**

Pour la base de données de 2 classes :

```
model.fit({'input': X}, {'targets': Y}, n_epoch=50, validation_set=({'input': test_x}, {'targets': test_y}),
    snapshot_step=500, show_metric=True, run_id=MODEL_NAME)
```

```
Training Step: 2349  | total loss: 0.10980 | time: 192.675s
| Adam | epoch: 050 | loss: 0.10980 - acc: 0.9726 -- iter: 2944/2949
Training Step: 2350  | total loss: 0.11110 | time: 211.438s
| Adam | epoch: 050 | loss: 0.11110 - acc: 0.9722 | val_loss: 0.60419 - val_acc: 0.8314 -- iter: 2949/2949
--
```

Pour la base de données de 3 classes :

```
model.fit({'input': X}, {'targets': Y}, n_epoch=50, validation_set=({'input': test_x}, {'targets': test_y}),
    snapshot_step=500, show_metric=True, run_id=MODEL_NAME)
```

```
Training Step: 2349  | total loss: 0.20218 | time: 637.480s
| Adam | epoch: 050 | loss: 0.20218 - acc: 0.9292 -- iter: 2944/2949
Training Step: 2350  | total loss: 0.20543 | time: 667.611s
| Adam | epoch: 050 | loss: 0.20543 - acc: 0.9238 | val_loss: 0.53602 - val_acc: 0.8200 -- iter: 2949/2949
--
```

Pour la base de données de 5 classes : On choisit les époques 120, le pas dans les époques = 29 et on lance l'entrainement :

```
%time
history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
        validation_data=val_generator,
        validation_steps=val_generator.samples//val_generator.batch_size
        )
```

```
Epoch 108/120
29/29 [==============================] - 43s 1s/step - loss: 0.4178 - acc: 0.8340 - val_loss: 0.9049 - val_acc: 0.7240
Epoch 109/120
29/29 [==============================] - 44s 1s/step - loss: 0.4053 - acc: 0.8468 - val_loss: 0.8131 - val_acc: 0.7535
Epoch 110/120
29/29 [==============================] - 44s 1s/step - loss: 0.4313 - acc: 0.8442 - val_loss: 0.7401 - val_acc: 0.7639
Epoch 111/120
29/29 [==============================] - 44s 1s/step - loss: 0.4189 - acc: 0.8411 - val_loss: 0.7714 - val_acc: 0.7726
Epoch 112/120
29/29 [==============================] - 44s 1s/step - loss: 0.4671 - acc: 0.8185 - val_loss: 0.6754 - val_acc: 0.7500
Epoch 113/120
29/29 [==============================] - 44s 1s/step - loss: 0.4648 - acc: 0.8300 - val_loss: 0.7543 - val_acc: 0.7604
Epoch 114/120
29/29 [==============================] - 45s 2s/step - loss: 0.4130 - acc: 0.8422 - val_loss: 0.7470 - val_acc: 0.7622
Epoch 115/120
29/29 [==============================] - 44s 1s/step - loss: 0.4172 - acc: 0.8455 - val_loss: 0.8255 - val_acc: 0.7517
Epoch 116/120
29/29 [==============================] - 44s 1s/step - loss: 0.4206 - acc: 0.8389 - val_loss: 0.7419 - val_acc: 0.7830
Epoch 117/120
29/29 [==============================] - 44s 1s/step - loss: 0.4349 - acc: 0.8433 - val_loss: 0.6761 - val_acc: 0.7639
Epoch 118/120
29/29 [==============================] - 44s 2s/step - loss: 0.4127 - acc: 0.8429 - val_loss: 0.7222 - val_acc: 0.7639
Epoch 119/120
29/29 [==============================] - 43s 1s/step - loss: 0.3982 - acc: 0.8530 - val_loss: 0.7398 - val_acc: 0.7500
Epoch 120/120
29/29 [==============================] - 43s 1s/step - loss: 0.3844 - acc: 0.8526 - val_loss: 0.6740 - val_acc: 0.7604
```

Figure III.32 : visualisation de l'entrainement

- **On note que plus les classes de base de données sont petites, meilleurs sont les résultats.**

Ensuite, nous devons le sauvegarder :

```
model.save(MODEL_PATH)
model.save_weights(MODEL_WEIGHTS_PATH)
```

Figure III.33 : modèle sauvegarder

III.2.9 Résultats et discussion :

a. Loss et acc fonction :

Une fois l'apprentissage terminé, nous voulons savoir s'il a été fait correctement, il est donc nécessaire d'afficher les fonctions de loss et de accuracy pour pouvoir observer les résultats.

```
[21] acc = history.history['acc']
     val_acc = history.history['val_acc']
     loss = history.history['loss']
     val_loss = history.history['val_loss']

     epochs = range(len(acc))

     plt.plot(epochs, acc, 'b', label='Training acc')
     plt.plot(epochs, val_acc, 'r', label='Validation acc')
     plt.title('Training and validation accuracy')
     plt.legend()

     plt.figure()

     plt.plot(epochs, loss, 'b', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
     plt.title('Training and validation loss')
     plt.legend()

     plt.show()
```

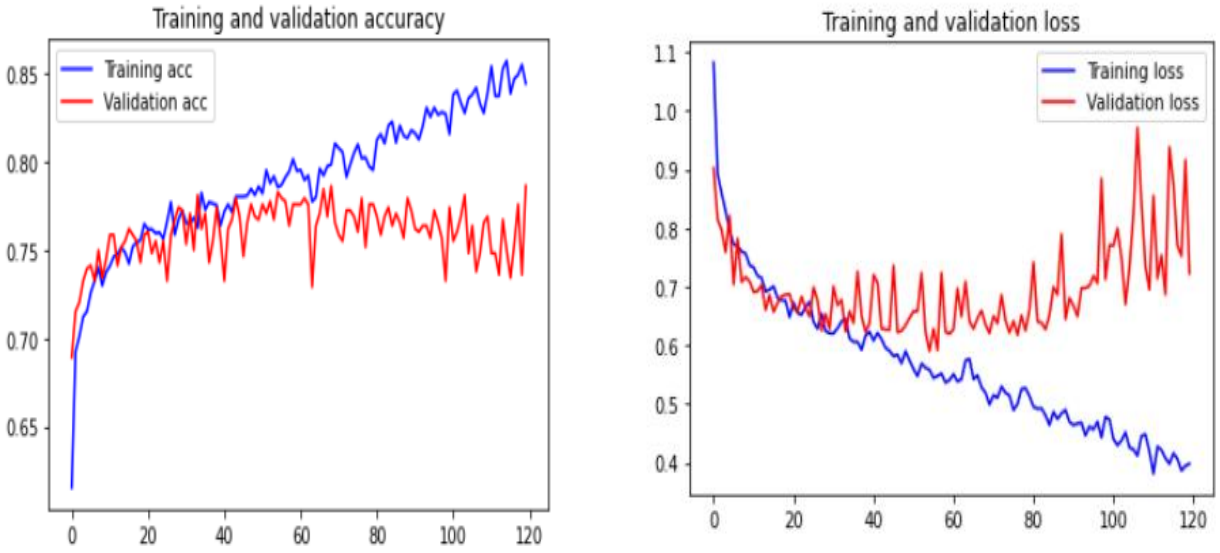


Figure III.34: acc et loss function

Ensuite, nous voulons connaître la durée de chaque époque :

```
[22] %%time
     loss, accuracy = model.evaluate_generator(test_generator,steps=test_generator.samples//test_generator.batch_size)

     CPU times: user 9.05 s, sys: 476 ms, total: 9.52 s
     Wall time: 2min 36s
```

b. test :

Pour tester si ce réseau apprend correctement, nous lui présentons des images de notre base de données et lui demandons de les tester.

```
[27] %%time
     Y_pred = model.predict_generator(test_generator,verbose=1, steps=test_generator.samples//test_generator.batch_size)

     48/48 [==============================] - 6s 127ms/step
     CPU times: user 7.87 s, sys: 338 ms, total: 8.21 s
     Wall time: 10.5 s
```

Ensuite, nous lui disons d'aller dans le dossier test et de prendre toutes les photos et de les redimensionner a 224.

```
[28] def load_image(filename):
         img = cv2.imread(os.path.join(BASE_DATASET_FOLDER, TEST_FOLDER, filename))
         img = cv2.resize(img, (IMAGE_SIZE[0], IMAGE_SIZE[1]) )
         img = img /255

         return img


     def predict(image):
         probabilities = model.predict(np.asarray([img]))[0]
         class_idx = np.argmax(probabilities)

         return {classes[class_idx]: probabilities[class_idx]}
```

A la fin, nous lui demandons d'afficher les images de test et de nous donner sa source et s'il les reconnait et le pourcentage de reconnaissance de celles-ci

```
[29] for idx, filename in enumerate(random.sample(test_generator.filenames, 100)):
         print("SOURCE: class: %s, file: %s" % (os.path.split(filename)[0], filename))

         img = load_image(filename)
         prediction = predict(img)
         print("PREDICTED: class: %s, confidence: %f" % (list(prediction.keys())[0], list(prediction.values())[0]))
         plt.imshow(img)
         plt.figure(idx)
         plt.show()
```

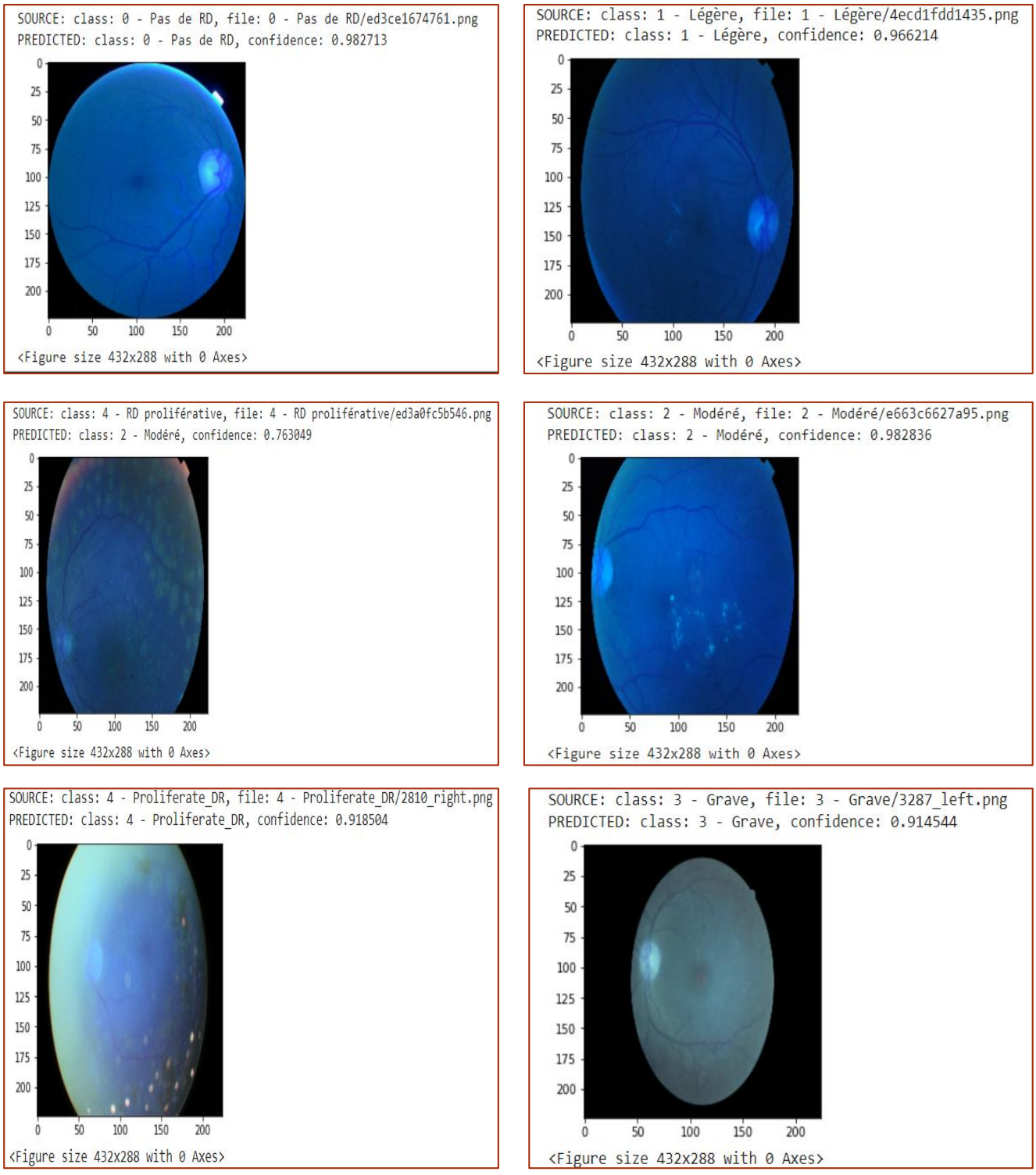


Figure III.35 : visualisation des images de test.

c. La matrice :

```
[68] y_pred = np.argmax(Y_pred, axis=1)
```

```
[81] cnf_matrix = confusion_matrix(test_generator.classes, y_pred)
```

```
[83] plot_confusion_matrix(cnf_matrix, list(classes.values()))
```

```
    plt.ylabel('True label', fontsize=16)
    plt.xlabel('Predicted label', fontsize=16)
```

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=(12,12))

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=18)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, fontsize=8)
    plt.yticks(tick_marks, classes, fontsize=12)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

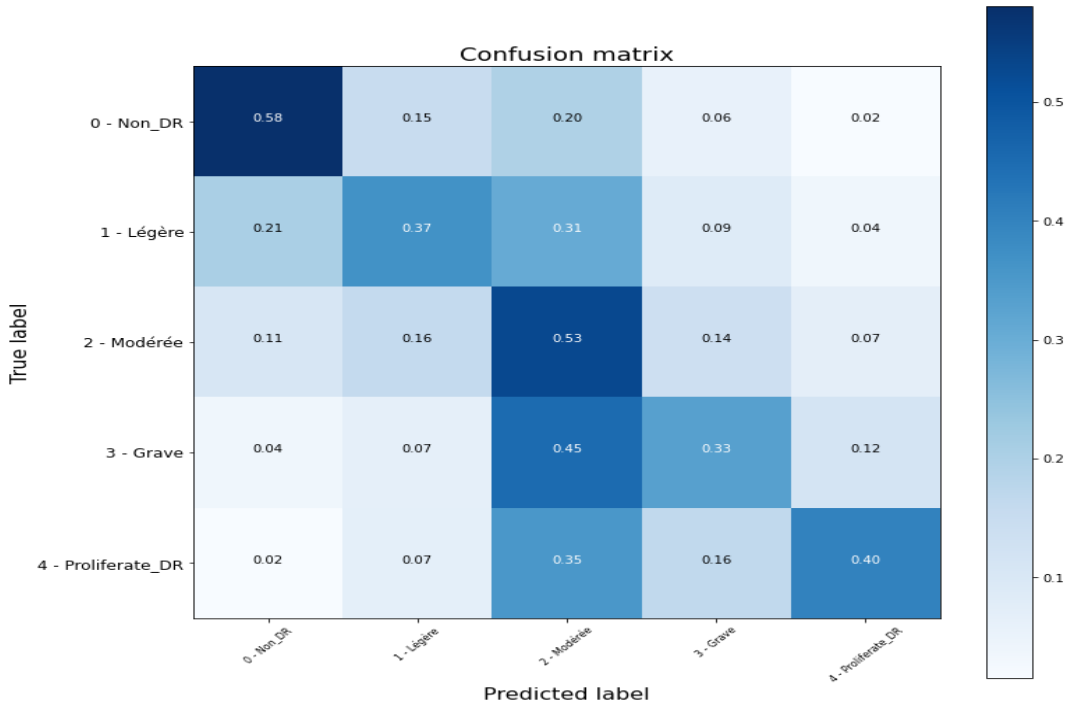


Figure III.36 : visualisation de la matrice

Comparaison :

- Comparaison entre les divisions de la base de donnée

division de la base de données	Sur deux classes	Sur trois classes	Sur cinq classes
La précision de la classification	Faible	Moyenne	Haute
l'apprentissage	Haute	Moyenne	Faible

Figure III.37 : comparaison entre les divisions de la base de donnée

- La comparaison de TPU et GPU et CPU dans une époque (la durée d'exécution) :

	COLAB	JUPYTER
TPU	22 min	/
CPU	35 min	5min
GPU	49 seconde	/

Figure III.38 : comparaison entre TPU et GPU et CPU

Discussion :

D'après les résultats obtenus on peut constater que la performance diffère d'un réseau à l'autre, la meilleure est celle de VGG-16

Dans le VGG16 les résultats de loss fonction étaient petits et de acc fonctions étaient grande cela indique qu'ils sont de bons résultats, d'autre part dans Alex net et l'inception_v3 loss fonction était 1, cela signifie que notre réseau n'a pas bien appris.

En VGG16 quand on augmente le nombre d'époques les résultats sera meilleur mais par contre dans Alex net et inception_V3 ne changent pas.

Notre base de donnée se compose de plusieurs classes, la meilleure architecture est celle de VGG16.

Les images de la rétinopathie diabétique ont besoin d'un réseau qui lit bien les pixels et découpe les images de la bonne façon pour trouver la maladie et le modèle VGG16 peut faire ces choses.

III.3 Implémentation du masque U-Net pour la détection :

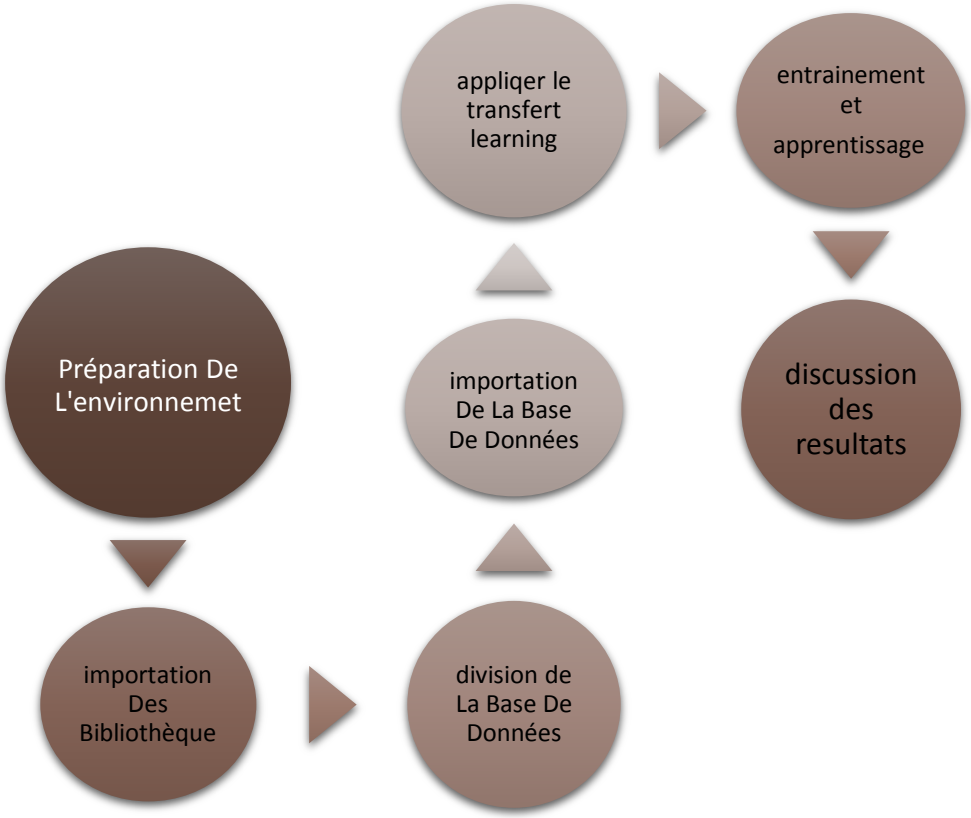


Figure III.39 : Les étapes de détection des images de fond d'œil.

III.3.1 préparation de l'environnement et importation des bibliothèques :

Nous allons d'abord importer la bibliothèque keras avec le backend tensorflow pour préparer l'environnement de travail et toutes les bibliothèques nécessaires, et on va utiliser le modèle U-Net pour la détection.

```
import os
import cv2
import numpy as np
from keras.layers.core import Lambda
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, BatchNormalization
from keras.layers.merge import concatenate
from keras.models import Model, load_model
from keras.callbacks import LearningRateScheduler
from keras import backend as K
import tensorflow as tf

from keras.models import Model, load_model
from keras.layers import Input
from keras.layers.core import 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 matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

III.3.2: division de la base de données :

Pour fait la détection de la rétinopathie diabétique, on a choisi la base de données « diaretdb1_v_1_1 » de la plateforme « Kaggle », cette base de données est composée d'environ 89 images du fond d'œil et 89 masque de chaque image au format PNG et de 1500*1152.

Avec la méthode manuelle utilisée, nous diviserons les images en 20% pour le test et 80% pour train.

Pour les images

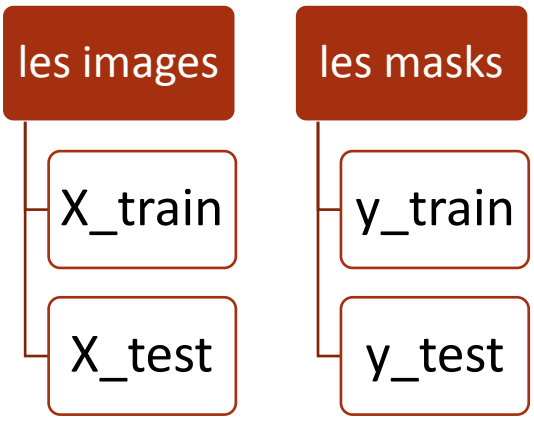


Figure III.40 : Illustration de la division de la base de donnée.

Figure III.41: la division de la base de données dans le drive.

Des exemples :

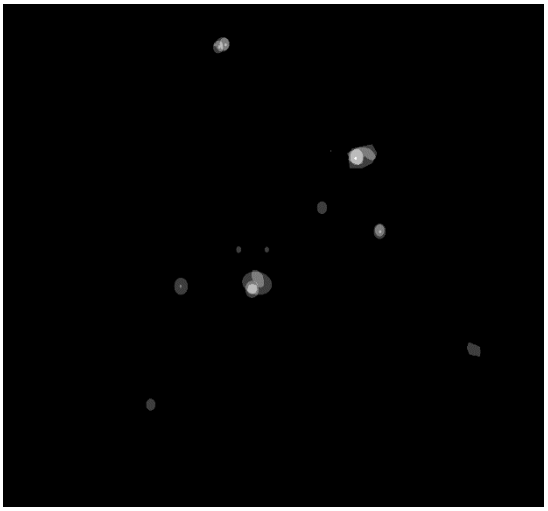
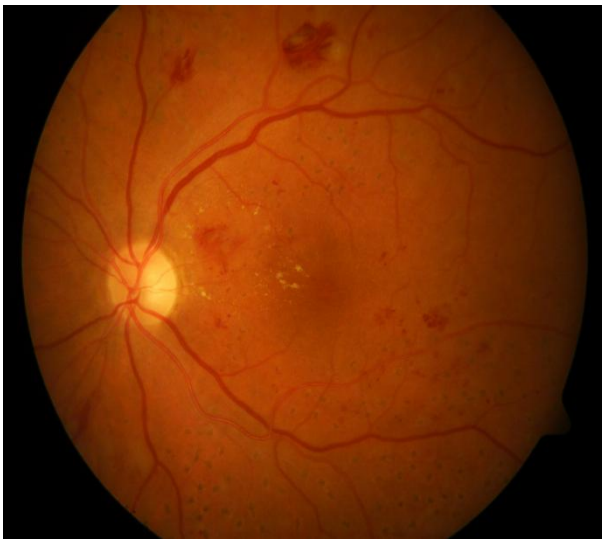
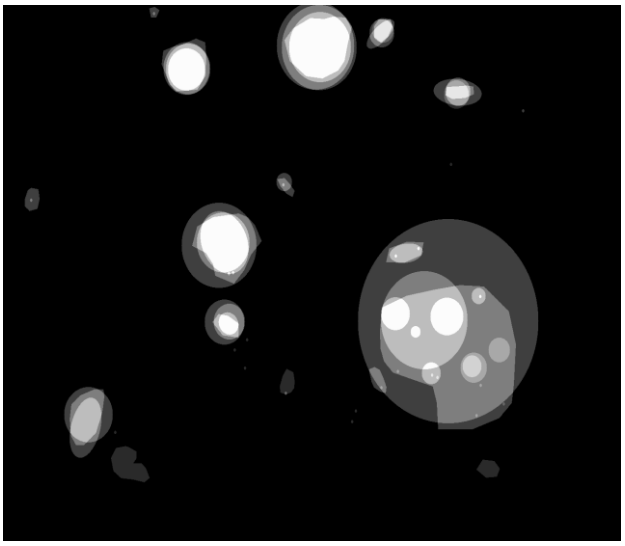





Image **masque**

Figure III.42 : des exemples sur la base de donnée

III.3.3: Importation de la base de données :

```
image_path = sorted(os.listdir("/content/drive/MyDrive/X_train"))
label_path = sorted(os.listdir("/content/drive/MyDrive/y_train"))
```

❖ Redimensionne et visualisation des images :

Dans cette partie nous avons redimensionne la taille de tous les images ont 128*128 et afficher quelques images et leur masque.

```
[4]  def load_images(inputdir, inputpath, imagesize):
         imglist = []

         for i in range(len(inputpath)):
             img = cv2.imread(inputdir+inputpath[i], cv2.IMREAD_COLOR)
             img = cv2.resize(img, (imagesize, imagesize), interpolation = cv2.INTER_AREA)
             img = img[::-1]
             imglist.append(img)

         return imglist
```

```
[5]  def segment(label, img_size):
         labels = []

         for i in range(len(label)):
             tmp = label[i].flatten()
             for j in range(len(tmp)):
                 if tmp[j] > 10:
                     tmp[j] = 200

             labels.append(tmp.reshape(img_size,img_size,3))

         return labels
```

```
IMAGE_SIZE = 128

image = load_images("/content/drive/MyDrive/X_train/", image_path, IMAGE_SIZE)
before_label = load_images("/content/drive/MyDrive/y_train/", label_path, IMAGE_SIZE)

after_label = segment(before_label, IMAGE_SIZE)

image /= np.max(image)
after_label /= np.max(after_label)
```

```
[13] num = 40


     plt.figure(figsize=(14, 7))

     ax = plt.subplot(1, 2, 1)
     plt.imshow(np.squeeze(image[num]))

     ax = plt.subplot(1, 2, 2)
     plt.imshow(np.squeeze(after_label[num]))
```

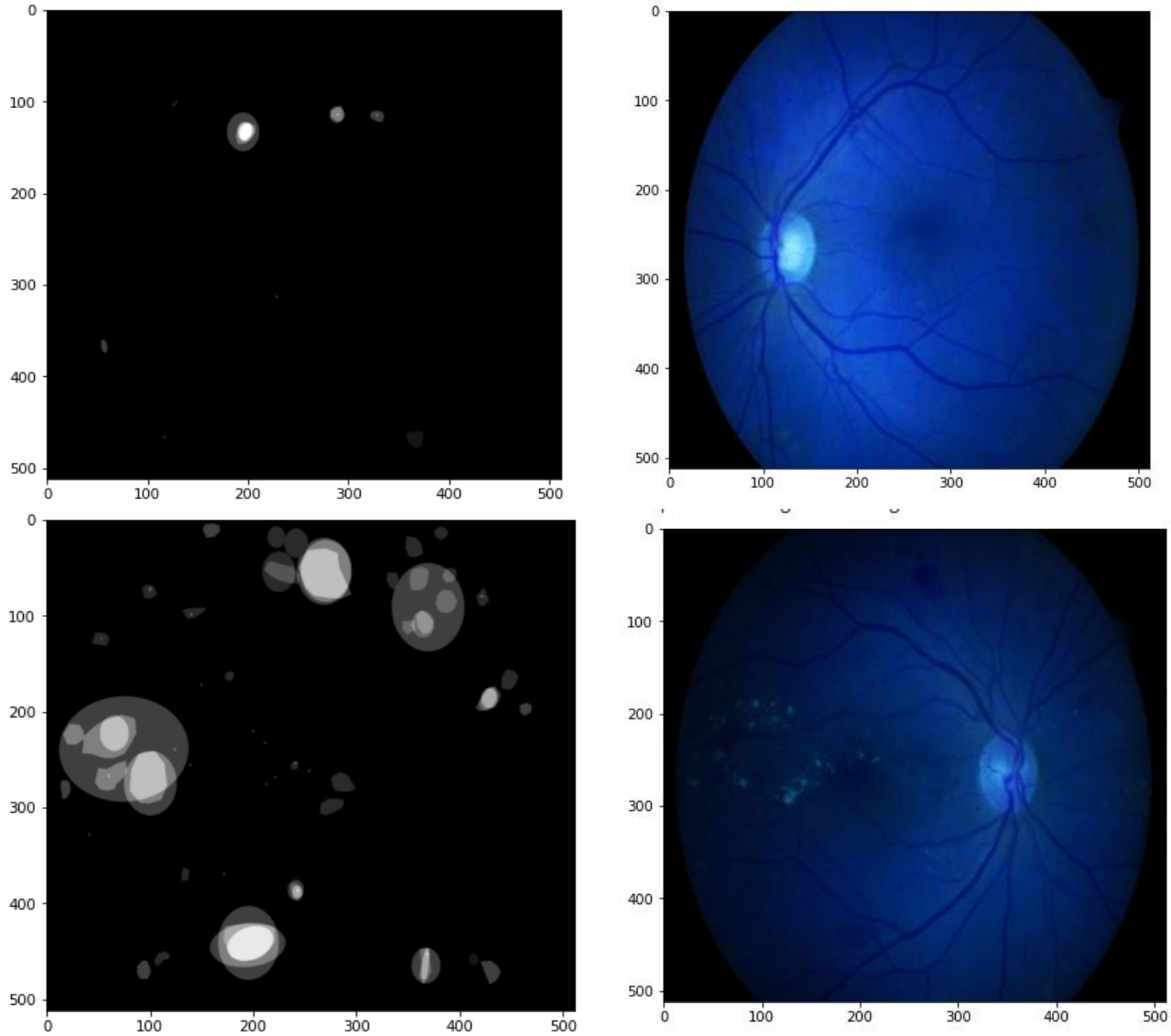


Figure III.43 : visualisation d'images et leur masque.

III.3.4 : applique le transfer learning :

Tous d'abord, on implémentons le réseau U-net :

```
inputs = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
s = Lambda(lambda x: x / 255) (inputs)

c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (s)
c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (p1)
c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(32, (3, 3), activation='relu', padding='same') (p2)
c3 = Conv2D(32, (3, 3), activation='relu', padding='same') (c3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(64, (3, 3), activation='relu', padding='same') (p3)
c4 = Conv2D(64, (3, 3), activation='relu', padding='same') (c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(128, (3, 3), activation='relu', padding='same') (p4)
c5 = Conv2D(128, (3, 3), activation='relu', padding='same') (c5)

u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(64, (3, 3), activation='relu', padding='same') (u6)
c6 = Conv2D(64, (3, 3), activation='relu', padding='same') (c6)
```

```
u7 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(32, (3, 3), activation='relu', padding='same') (u7)
c7 = Conv2D(32, (3, 3), activation='relu', padding='same') (c7)

u8 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(16, (3, 3), activation='relu', padding='same') (u8)
c8 = Conv2D(16, (3, 3), activation='relu', padding='same') (c8)

u9 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(8, (3, 3), activation='relu', padding='same') (u9)
c9 = Conv2D(8, (3, 3), activation='relu', padding='same') (c9)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])
```

Nous voulons maintenant afficher un résumé du modèle et déterminer les paramètres d'entrainement.

conv2d_46 (Conv2D)	(None, 8, 8, 128)	73856	max_pooling2d_11[0][0]
conv2d_47 (Conv2D)	(None, 8, 8, 128)	147584	conv2d_46[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 16, 16, 64)	32832	conv2d_47[0][0]
concatenate_8 (Concatenate)	(None, 16, 16, 128)	0	conv2d_transpose_4[0][0] conv2d_45[0][0]
conv2d_48 (Conv2D)	(None, 16, 16, 64)	73792	concatenate_8[0][0]
conv2d_49 (Conv2D)	(None, 16, 16, 64)	36928	conv2d_48[0][0]
conv2d_transpose_5 (Conv2DTrans	(None, 32, 32, 32)	8224	conv2d_49[0][0]
concatenate_9 (Concatenate)	(None, 32, 32, 64)	0	conv2d_transpose_5[0][0] conv2d_43[0][0]
conv2d_50 (Conv2D)	(None, 32, 32, 32)	18464	concatenate_9[0][0]
conv2d_51 (Conv2D)	(None, 32, 32, 32)	9248	conv2d_50[0][0]
conv2d_transpose_6 (Conv2DTrans	(None, 64, 64, 16)	2064	conv2d_51[0][0]
concatenate_10 (Concatenate)	(None, 64, 64, 32)	0	conv2d_transpose_6[0][0] conv2d_41[0][0]
conv2d_52 (Conv2D)	(None, 64, 64, 16)	4624	concatenate_10[0][0]
conv2d_53 (Conv2D)	(None, 64, 64, 16)	2320	conv2d_52[0][0]

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 128, 128, 3)	0	
lambda_1 (Lambda)	(None, 128, 128, 3)	0	input_3[0][0]
conv2d_38 (Conv2D)	(None, 128, 128, 8)	224	lambda_1[0][0]
conv2d_39 (Conv2D)	(None, 128, 128, 8)	584	conv2d_38[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 64, 64, 8)	0	conv2d_39[0][0]
conv2d_40 (Conv2D)	(None, 64, 64, 16)	1168	max_pooling2d_8[0][0]
conv2d_41 (Conv2D)	(None, 64, 64, 16)	2320	conv2d_40[0][0]
max_pooling2d_9 (MaxPooling2D)	(None, 32, 32, 16)	0	conv2d_41[0][0]
conv2d_42 (Conv2D)	(None, 32, 32, 32)	4640	max_pooling2d_9[0][0]
conv2d_43 (Conv2D)	(None, 32, 32, 32)	9248	conv2d_42[0][0]
max_pooling2d_10 (MaxPooling2D)	(None, 16, 16, 32)	0	conv2d_43[0][0]
conv2d_44 (Conv2D)	(None, 16, 16, 64)	18496	max_pooling2d_10[0][0]
conv2d_45 (Conv2D)	(None, 16, 16, 64)	36928	conv2d_44[0][0]
max_pooling2d_11 (MaxPooling2D)	(None, 8, 8, 64)	0	conv2d_45[0][0]

conv2d_transpose_7 (Conv2DTrans	(None, 128, 128, 8)	520	conv2d_53[0][0]
concatenate_11 (Concatenate)	(None, 128, 128, 16)	0	conv2d_transpose_7[0][0] conv2d_39[0][0]
conv2d_54 (Conv2D)	(None, 128, 128, 8)	1160	concatenate_11[0][0]
conv2d_55 (Conv2D)	(None, 128, 128, 8)	584	conv2d_54[0][0]
conv2d_56 (Conv2D)	(None, 128, 128, 1)	9	conv2d_55[0][0]

Total params: 485,817
Trainable params: 485,817
Non-trainable params: 0

III.3.5 Entrainement et l'apprentissage :

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

On utilise model. Compile () car il prend l'optimiseur la perte (loss) et la métrique (metrics) comme paramètres

Tout d'abord, nous devons choisir les paramètres de keras, nous prenons 12 image de train batch pour faire une 30 époque.

```
X_train, X_val, T_train, T_val = train_test_split(image, after_label, test_size=0.2)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[dice_coef])

initial_learningrate=2e-3

def lr_decay(epoch):
    if epoch < 10:
        return initial_learningrate
    else:
        return initial_learningrate * 0.99 ** epoch

training = model.fit(X_train, T_train,epochs=30, batch_size=12, shuffle=True,
                     validation_data=(X_val, T_val), verbose=1,callbacks=[LearningRateScheduler(lr_decay,verbose=1)])
```

Ensuite, nous devons le sauvegarder :

```
model.save("Unet_model.h5")
```

III.3.6 Résultats et discussion :

Nous importons le dossier de test pour tester les résultats :

```
test_path = os.listdir("/content/drive/MyDrive/data89m/X_test")
test_image = load_images("/content/drive/MyDrive/data89m/X_test/", test_path, IMAGE_SIZE)

test_image /= np.max(test_image)
```

```
results = model.predict(test_image,verbose=1)
```

A la fin, nous lui demandons d'afficher quelque images de test :

```
n = 5

plt.figure(figsize=(14, 7))

ax = plt.subplot(1, 2, 1)
plt.imshow(np.squeeze(test_image[n]))

ax = plt.subplot(1, 2, 2)
plt.imshow(np.squeeze(results[n]))
```

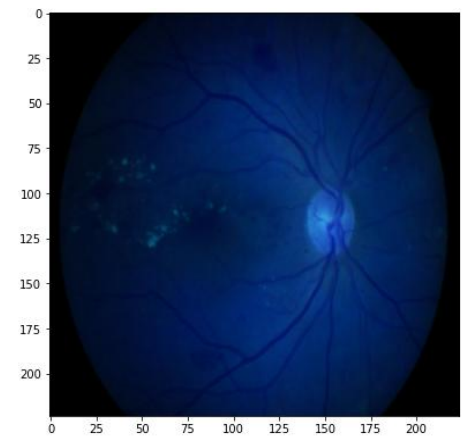
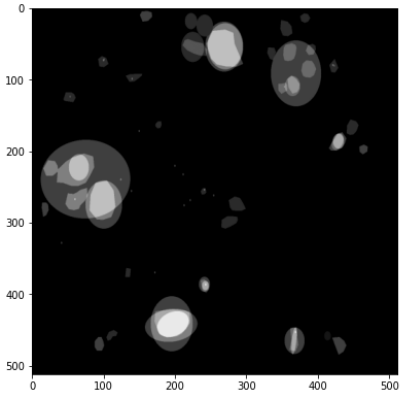
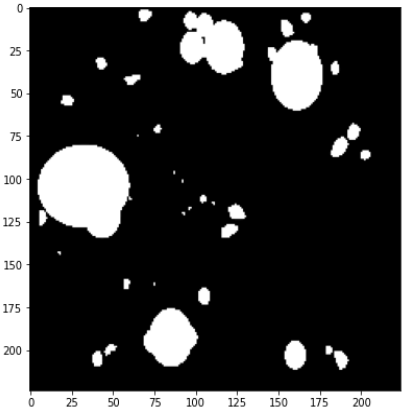
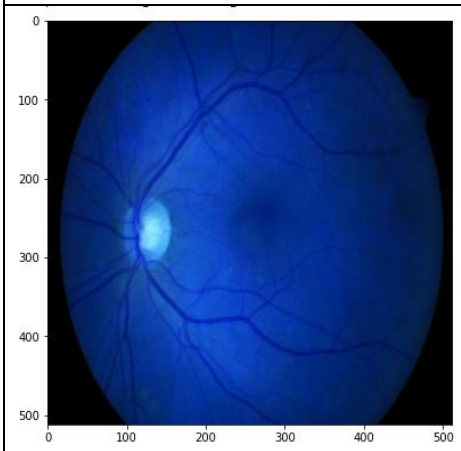
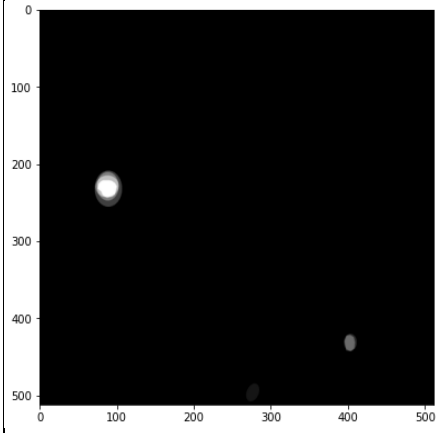
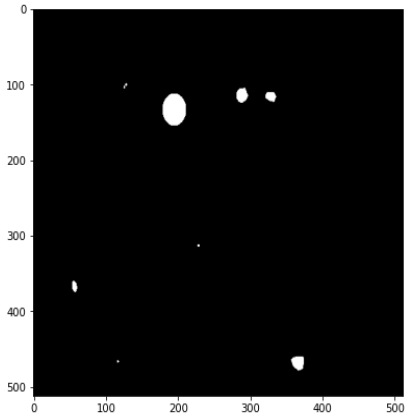
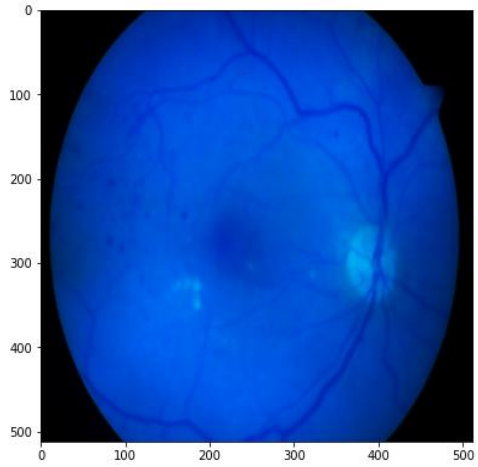
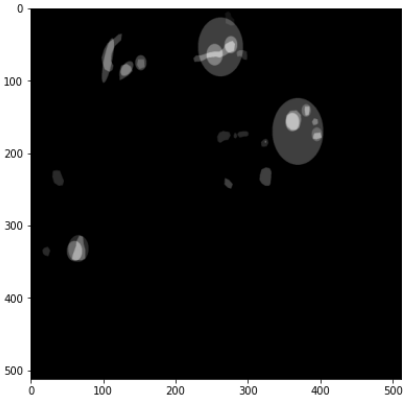
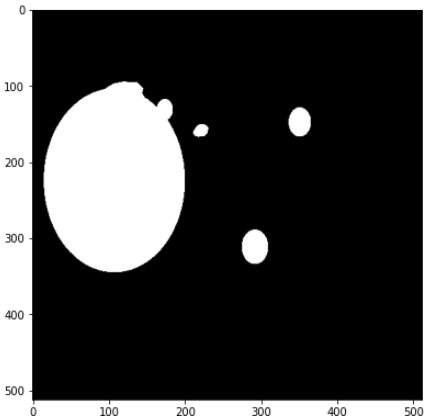
Image	Masque	Résultat
		
		
		

Figure III.44 : résultat de détection de RD par U-Net.

Discussion :

D'après les résultats obtenus la RD est détecté, cependant que les contours des zones touchées ne sont pas précis.

III.4 Conclusion :

Dans ce chapitre nous avons présenté tous les étapes nécessaires de notre projet commençant par l'importation des bibliothèques et la préparation de l'environnement de travail ensuite nous avons créé un nouveau réseau pour la classification et la détection et le test des images on a conclu.

Nous pourrons obtenir des résultats plus précise avec une base de données plus grande.

Les résultats obtenus avec VGG16 pour la classification sont meilleurs qu'eux d'Alex net et de inception V3.

Pour la détection, U-net donne des résultats mais a besoins d'une plus grande base de données aussi.

Finalement on a remarqué que le travaille avec le GPU et le TPU permet de gagner beaucoup de temps d'exécution.

Conclusion générale

Le principal objectif de ce mémoire était de développer un algorithme pour l'aide au diagnostic en ophtalmologie ; il s'agit de dépistage de la RD dans les images du fond d'œil ; en utilisant des outils de traitement des images rétiniennes.

La première partie de cette étude a présenté les termes médicaux permettant une familiarisation avec le domaine d'ophtalmologie. Les différents éléments constituant le fond d'œil, les pathologies pouvant affecter la rétine, plus particulièrement la RD.

La recherche présentée dans ce mémoire a montré des résultats encourageants qui indiquent que le système d'analyse des images du fond d'œil proposé a réussi à extraire tout d'abord les pathologies rétiniennes ainsi de classer les grades de la RDNP. Comme le système détecte les stades de la RDNP avec une bonne sensibilité et une spécificité, il peut être utilisé pour aider les ophtalmologistes dans le diagnostic afin de leurs fournir une seconde opinion et peut également fonctionner comme un outil pour le dépistage de masse de la RD.

Les résultats obtenus pour la classification par VGG16 et de détection par U-Net permettent d'envisager que leur amélioration (par l'utilisation de fonction de transfert différentes, des bases de données étiquetées plus grandes) peut réellement conduire à une reconnaissance et une détection automatiques et précises de la RD.

Bibliographie

[1] https://www.provisu.ch/fr/dossiers/oeil-et-vision.html

[2] le diabète en quelques mots : définition et symptomes,2021 Medtronic France SAS

[3] Dr Anne-Christine Della Valle, Diabète : symptômes, causes, diagnostic, traitements, journal des femmes,12 novembre 2020

[4] lise loumé, Rétinopathie diabétique : comment le diabète peut rendre aveugle sciencesetavenir,11mai 2016

[5] Rétinopathie diabétique : symptômes et traitement 25 aout 2017

[6] démytifier le machine Learning, les réseaux de neurones artificiels, juripredis, 2018

[7] Natural solutions, les réseaux de neurones convolutifs

[8] afshine amidi ,cs 230-apprentissage profond , stanford.edu

[9] shervine amidi, cs 230-apprentissage profond, stanford.edu

[10] rohith gandhi ,yolo-object detection alogorithms towards data science

[11] jason brownlee , a gentle introduction to transfert learning for deep learning , deep learning for computer vision , 20 décembre 2017

[12] VGG-16|CNN model , geeksforgeeks , 27 février 2020

[13]vihan kurama, a review of popular deep learning architecture : ResNet,InceptionV3,and squeeZenet,blog.paperSpace,2020

[14] jerry wei , alexNet :the architecture thet challenged CNNs , towardsdatascience ,jul3,3019

[15] u-Net :concolutional networks for biomedical image segmentation,arxiv-vanity,2015

[16] multi-class neural networks : softmax , machine learning , 17 mars 2020

[17] les fonctions de perte , intelligence artificielle vulgarisé

[18] fonction de perte d'entropie croisée , icho.pro

[19] ajay sarangam , epoch in machine learning : a simple introduction 2021 , jigsawacademy

[20] ilyes talbi , optimisation de fonctions descentes de gradient : applications aux réseaux de neurones , 6avril

[21] Afshine amidi et shervine amidi , pense-bete d'apprentissage profond , CS229-machine learning

[22] optimisation de la descente de gradient code ADAM à partir de zéro , Brutalk

[23] comprendre la descente de dradient et l'optimisation Adam, Ichi.pro

[24] bastientl, python : tout savoir sur le principal langage big data et machine Learning, lebigdata,17juin 2021

[25] 15 bibliothèques python pour la science des données que vous devez connaitre, monocoachdata

[26] Google colab-yourfirst colab notebook,tutorials point

[27] michael aramini, quelle est la différence entre un GPU, TPU et CPU ? quora,2019

[28] 18 jupyter et ses note books , python.SDV.univ-paris_diderot.fr