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 الالكترونيك

Mémoire de Master

Filière Électronique

Spécialité Électronique des Systèmes Embarqués

Présenté par

ABABSA Mohamed Tahar

&

BOUKRID Nesrine

Implémentation de U-net pour la segmentation de l'arbre rétinien, et implantation d'un CNN sur FPGA.

Encadré par : Mme. BOUGHERIRA HAMIDA

Année Universitaire 2023-2024

Dédicaces

Ce Je dédie ce modeste travail à notre chère mère, symbole de l'amour et de la tendresse, qui m'a toujours soutenu et encouragé pendant toute ma scolarité. À celui qui a sacrifié pour que mes études puissent se dérouler dans de meilleures conditions morales et financières, mon cher père, symbole de sagesse et de sacrifice.

- *À mon binôme Tahar*
- *À mes chers parents Abdelkrim et Hanifa*
- *À mes frères Younes, Hakim, Amine*
- *À Youcef, Amir, Rayane, Iyad, et Dania*
- *À mes belles-sœurs Imane et Yasmina*
- *À nos amis Soumia, Mounia, Narimane, Yasmina, et Ines*
- *À nos collègues du département de l'électronique avec qui j'ai passé des moments exceptionnels*
- *À toutes nos connaissances et personnes que j'aime*

Nesrine

Dédicaces

Je réserve cette page pour exprimer ma profonde gratitude envers tous ceux qui ont contribué à la réalisation de ce travail.

En premier lieu, mes chers parents méritent toute ma reconnaissance pour leur patience, leur encouragement et leur soutien indéfectible. À mes frères ANIS, YOUCEF, YACINE, et FARES, je suis reconnaissant pour leur amour et leur soutien constant.

Ma promotrice, Mme BOUGHERIRA HAMIDA, mérite mes remerciements pour ses conseils précieux et son accompagnement tout au long de ce projet.

De même, je remercie chaleureusement notre chef de spécialité, Mme NACEUR DJAMILA, pour son encouragement et son soutien dans ma formation en Electronique des Systèmes Embarqués.

Je tiens à exprimer ma reconnaissance envers le doctorant et mon co-promoteur, Boukrid Abdelhakim, ainsi que mon binôme, Nesrine, pour leur contribution essentielle à ce projet.

Enfin, je remercie sincèrement les membres du jury pour leur examen attentif de ce travail, ainsi que toutes les personnes qui ont participé de près ou de loin à sa réalisation.

Tahar

Résumé

Ce projet explore l'intégration de l'intelligence artificielle dans l'imagerie médicale, en mettant en œuvre l'architecture U-Net pour la segmentation précise de l'arbre rétinien. L'utilisation de réseaux de neurones convolutifs permet d'identifier avec exactitude les structures anatomiques critiques, notamment dans la détection avancée de la rétinopathie diabétique. Parallèlement, un CNN est implanté sur une carte FPGA Pynq Z2 pour traiter les données MNIST, les images étant transmises via UART. Cette combinaison exploite le parallélisme matériel du FPGA pour accélérer le traitement, démontrant des performances optimisées et une efficacité énergétique accrue pour des applications en temps réel.

Mots clés: Imagerie médicale, intelligence artificielle, U-Net, réseaux de neurones convolutifs, segmentation de l'arbre rétinien, rétinopathie diabétique, implémentation FPGA, applications en temps réel, performances optimisées, efficacité énergétique

ملخص

يستكشف هذا المشروع دمج الذكاء الاصطناعي في التصوير الطبي من خلال تنفيذ بنية U-Net لتجزئة شجرة الشبكية بدقة. يسمح استخدام الشبكات العصبية التلافيفية بتحديد الهياكل التشريحية بدقة، خاصة في الكشف المتقدم عن اعتلال الشبكية السكري. في الوقت نفسه، يتم تنفيذ شبكة عصبية تلافيفية (CNN) على لوحة FPGA Pynq Z2 لمعالجة بيانات MNIST، حيث يتم نقل الصور عبر UART. يستفيد هذا المزيج من التوازي المادي لـ FPGA لتسريع المعالجة، مما يُظهر أداءً محسّنًا وكفاءة طاقة متزايدة للتطبيقات في الوقت الفعلي.

كلمات المفتاحية: التصوير الطبي، الذكاء الاصطناعي، U-Net، الشبكات العصبية التلافيفية، تجزئة شجرة الشبكية، اعتلال الشبكية السكري، تنفيذFPGA ، التطبيقات في الوقت الفعلي، الأداء المحسن، كفاءة الطاقة

Abstract

Medical imaging has been significantly enhanced by artificial intelligence (AI), particularly in the precise detection of retinal diseases such as diabetic retinopathy. This project focuses on implementing the U-Net architecture for accurate retinal vessel segmentation using convolutional neural networks (CNNs). Concurrently, a CNN is deployed on an FPGA-based Pynq Z2 board to efficiently process MNIST data, with images transmitted via UART. This combination leverages the hardware parallelism of the FPGA to accelerate processing, demonstrating optimized performance and increased energy efficiency for real-time applications.

Keywords: U-net; Segmentation ; Retinal vessel ;Convolutional neural network (CNN) ; Pynq Z2 board ; MNIST data ; UART ; Hardware parallelism ; FPGA ; Real-time processing ; Optimized performance ; Energy efficiency

Table des matières

Chapitre 01 : Généralités

Chapitre 03 : Segmentation de l'arbre rétinien par U-net

Chapitre 04 : Implémentation sur FPGA

Liste des abréviations

API : Interface de programmation d'applications (Application Programming Interface)

ARM : Advanced RISC Machines

AXI : Interface extensible avancée (Advanced eXtensible Interface)

BSD : Distribution de logiciels Berkeley (Berkeley Software Distribution)

CNN : Réseau de neurones convolutifs (Convolutional Neural Network)

CV : Vision par ordinateur (Computer Vision)

DMLA : Dégénérescence maculaire liée à l'âge

DRIVE : Digital Retinal Images for Vessel Extraction

FPGA : Matrice de portes programmable en champ (Field-Programmable Gate Array)

GPU : Unité de traitement graphique (Graphics Processing Unit)

IA : Intelligence artificielle

IDE : Environnement de développement intégré (Integrated Development Environment)

IoT : Internet des objets (Internet of Things)

IoU : Intersection sur Union (Intersection over Union)

JPEG : Groupe d'experts en photographie (Joint Photographic Experts Group)

Jupyter : Julia, Python et R

LED : Diode électroluminescente (Light Emitting Diode)

MLP : Perceptron multicouche (Multi-Layer Perceptron)

MNIST : Institut national des normes et de la technologie modifié (Modified National Institute of Standards and Technology)

PC : Ordinateur personnel (Personal Computer)

PL : Logique programmable (Programmable Logic)

PNG : Graphiques portables en réseau (Portable Network Graphics)

PS : Système de traitement (Processing System)

RAM : Mémoire vive (Random Access Memory)

ReLU : Unité linéaire rectifiée (Rectified Linear Unit)

SD : Sécurisé numérique (Secure Digital)

SoC : Système sur puce (System on Chip)

SSD : Disque à état solide (Solid State Drive)

SSH : Shell sécurisé (Secure Shell)

SVM : Machine à vecteurs de support (Support Vector Machine)

TFlearn : Apprentissage de TensorFlow (TensorFlow Learn)

TPU : Unité de traitement Tensor (Tensor Processing Unit)

UART : Récepteur-transmetteur asynchrone universel (Universal Asynchronous Receiver-Transmitter)

U-Net : Réseau en U (U-shaped Network)

VHDL : Langage de description matériel VHSIC (VHSIC Hardware Description Language)

Vitis HLS : Synthèse de haut niveau Vitis (Vitis High Level Synthesis)

Vivado : Suite de conception Vivado (Vivado Design Suite)

Liste des figures

Liste des tableaux

Introduction Générale

Introduction Générale

L'imagerie médicale et l'intelligence artificielle (IA) ont considérablement transformé le domaine de l'ophtalmologie en permettant des avancées significatives dans le diagnostic et la gestion des maladies rétiniennes, notamment la rétinopathie diabétique. Ces pathologies nécessitent des outils de diagnostic précis pour détecter les changements précoces et orienter les traitements appropriés.

Ce projet se concentre sur l'implémentation de l'architecture U-Net pour la segmentation de l'arbre rétinien, une méthode reconnue pour sa capacité à segmenter de manière précise les structures anatomiques complexes à partir d'images médicales. L'U-Net sera utilisé pour analyser et identifier les vaisseaux sanguins ainsi que d'autres composants critiques de la rétine, facilitant une évaluation détaillée et objective.

En intégrant des techniques avancées de Deep Learning, utilisant des bibliothèques tels que TensorFlow et Keras, nous développerons des modèles optimisés pour améliorer la détection précoce et la caractérisation des anomalies rétiniennes. Cette approche permettra non seulement de renforcer la précision diagnostique mais aussi d'explorer des applications pratiques dans la surveillance et la gestion des patients atteints de rétinopathie diabétique.

Ce mémoire explore ainsi la convergence entre la biologie et la technologie, en exploitant les capacités avancées des réseaux neuronaux convolutifs pour transformer les pratiques cliniques en ophtalmologie. En mettant l'accent sur le développement de solutions logicielles robustes et efficaces, nous visons à fournir des outils innovants qui peuvent être facilement intégrés dans les environnements cliniques pour améliorer les soins aux patients.

En conclusion, ce projet représente une contribution significative à la recherche sur l'IA appliquée à l'imagerie médicale, ouvrant de nouvelles perspectives pour des diagnostics plus précis et des traitements personnalisés dans le domaine complexe de la rétinopathie diabétique.

Chapitre 01 :

Généralités

I.1. Introduction

Le but de ce projet est de démontrer l'intersection complexe entre la biologie et la technologie en explorant l'anatomie de l'œil humain, en particulier la rétine, et son importance dans la conversion des rayons lumineux en signaux nerveux. Parallèlement, nous appliquerons la vision par ordinateur, utilisant des réseaux de neurones convolutifs (CNN) via Keras/TensorFlow, pour identifier et classifier le contenu visuel des images. Ce processus impliquera la collecte et le prétraitement des images pour améliorer leur qualité et standardiser leurs dimensions, améliorant ainsi la précision de la segmentation, L'objectif final est de développer des applications pratiques telles que la détection de pathologies rétiniennes, fusionnant ainsi les avancées en ophtalmologie avec les innovations en intelligence artificielle.

I.2. L’anatomie de l'œil

Comprend le globe oculaire et ses annexes (muscles extra-oculaires, nerfs, paupière, système lacrymal, orbite). [1]

I.2.1. Les annexes du globe oculaire

Incluent les paupières, l’appareil lacrymal, et les muscles extra-oculaires, essentiels à la protection et au mouvement de l'œil.

I.2.2. Le globe oculaire

Il se divise en quatre parties principales : couche protectrice, couche vascularisée, contenu de la cavité interne, et couche visuelle.

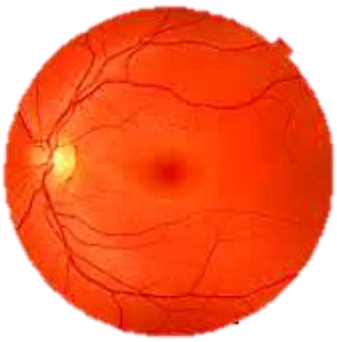


Figure 1.1 : *L’anatomie de l'œil*

I.3. La rétine

I.3.1. Définition de la rétine

La rétine est une fine membrane tapissant le globe oculaire. Par analogie avec un appareil photo, elle est en quelque sorte la « pellicule » de l'œil, chargée de capter les rayons lumineux pour les transmettre au système nerveux central. D'une épaisseur comprise entre 0,1 et 0,4 mm, elle est constituée de 10 couches : quatre de cellules photo réceptrices à l'extérieur et six de cellules nerveuses à l'intérieur. [2]

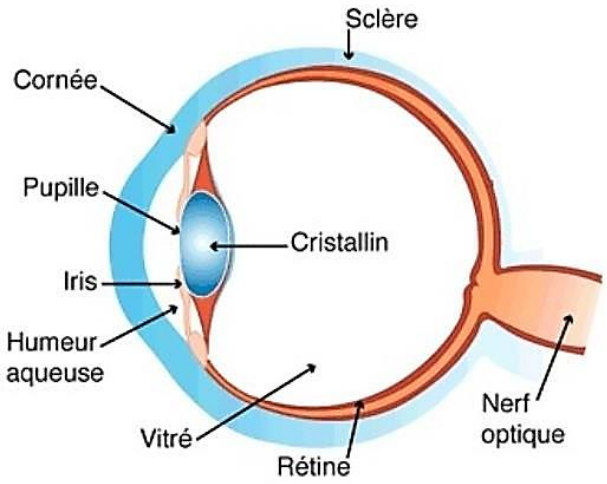


Figure 1.2 : *la rétine*

I.4. Anatomie de la rétine

La rétine, tissu sensible à l'arrière de l'œil, se divise en deux zones principales : la rétine centrale et la rétine périphérique.

I.4.1. La rétine centrale

Comprenant la macula et la fovéa, est spécialisée dans la vision détaillée et colorée grâce à ses 7 millions de cônes.

I.4.2. La rétine périphérique

Dominée par les bâtonnets au nombre de 110 à 130 millions, permet une vision efficace dans des conditions de faible luminosité ou de nuit, mais ne distingue pas les couleurs. Ce système permet à l'œil humain de s'adapter à divers environnements lumineux et de percevoir le monde avec précision et efficacité.

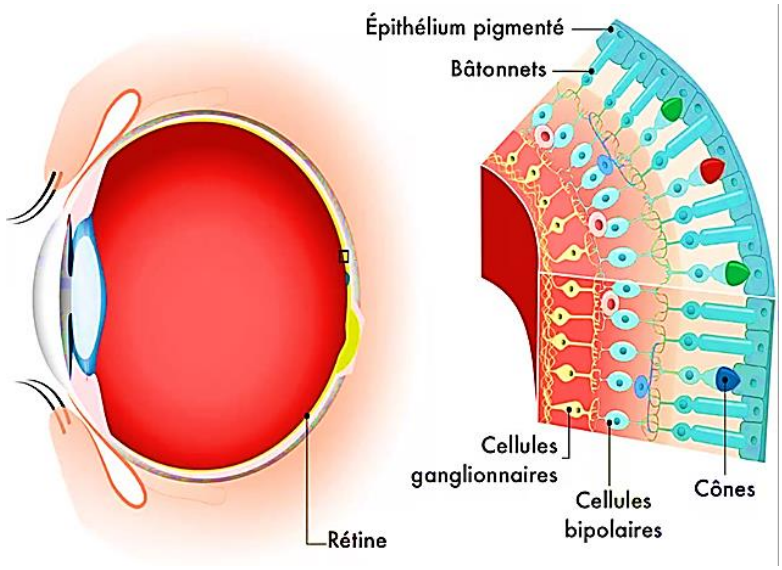


Figure.1.3 : *la rétine [1]*

La rétine est constituée de couches de cellules photo réceptrices chargées de capter la lumière et de cellules nerveuses chargées de transmettre l'information au cerveau.

L'œil humain peut discriminer deux millions de nuances de couleur et 750 niveaux de luminosité. Cette information est convertie en signal électrique par les neurotransmetteurs et transmise aux cellules ganglionnaires, qui la relayent au cerveau via le nerf optique. Ainsi, le cerveau interprète les données visuelles pour la perception.

I.5. Les différentes couches de la rétine

La rétine est une structure essentielle de l'œil, est composée de plusieurs couches distinctes, chacune remplissant des fonctions spécifiques dans le processus de vision [3].

- **Couche pigmentée externe (adhérente à la choroïde)**

Située à l'extrémité externe de la rétine, cette couche assure l'ancrage de la rétine à la choroïde, la couche vasculaire externe de l'œil.

- **Couche des cônes et des bâtonnets**

Cette couche est formée par les deux segments externes et internes des cellules visuelles, les cônes et les bâtonnets. C'est là que débutent les premiers processus de la vision. Les bâtonnets sont environ 130 millions et les cônes sont environ 65 millions.

- **Limitante externe**

Une membrane qui délimite la couche des grains externes.

- **Couche des grains externes**

Elle est constituée par les corps cellulaires des cônes et des bâtonnets.

- **Plexi forme externe**

Cette couche, d'une épaisseur d'environ 20 µm, est le site de la connexion synaptique entre les cônes, les bâtonnets et les dendrites des cellules de la couche suivante. C'est là que se fait la transition entre la rétine sensorielle et la rétine cérébrale.

- **Couche des grains internes**

D'une épaisseur d'environ 30 µm, cette couche est composée de neurones bipolaires qui transmettent l'influx nerveux des cellules réceptrices aux cellules ganglionnaires, ainsi que des cellules d'association et des cellules de soutien, comme les fibres de Müller et les cellules amarines.

- **Plexi forme interne**

D'une épaisseur de 20 à 30 µm, cette couche est le lieu de la connexion entre les cellules bipolaires et les cellules ganglionnaires.

Couche des cellules ganglionnaires : Composée de grandes cellules nerveuses, cette couche abrite les axones qui formeront le nerf optique. Elle est généralement constituée d'une seule couche, sauf autour de la fovéa, où les noyaux des cellules ganglionnaires s'empilent sur plusieurs rangs.

- **Couche des fibres optiques**

Cette couche regroupe les fibres nerveuses qui convergent vers la papille optique, où elles forment le nerf optique.

- **Limitante externe (formée par la réunion des extrémités internes des fibres des cellules de Müller)**

Cette couche, jouant un rôle de soutien, constitue la frontière interne de la rétine Fovéa, où les noyaux des cellules ganglionnaires s'empilent sur plusieurs rangs.

- **Couche des fibres optiques**

Cette couche regroupe les fibres nerveuses qui convergent vers la papille optique, où elles forment le nerf optique.

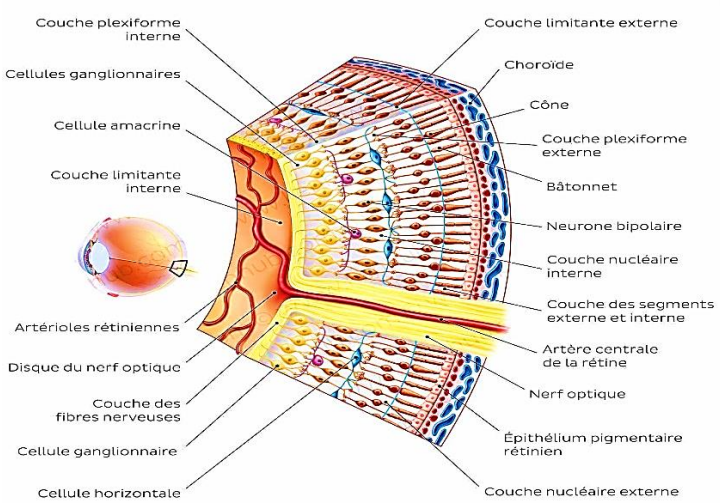


Figure 1.4 : *Les différentes couches de rétine*

I.6. Anatomie Macroscopique de la rétine

L'œil humain perçoit formes et couleurs ; l'ophtalmologie étudie sa complexité. Le défi technologique est de créer des yeux électroniques surpassant les capacités humaines, potentiellement pour remplacer les yeux blessés. Cela nécessite des avancées en robotique, intelligence artificielle et biotechnologie. Ces yeux artificiels pourraient offrir une vision améliorée et nocturne accrue.

Cependant, des questions éthiques concernant la confidentialité et l'identité humaine se posent. [4]

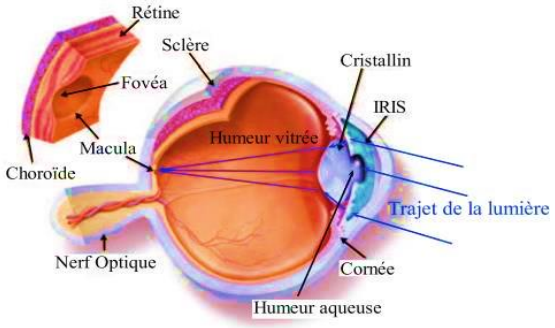


Figure.1.5 : *Anatomie Macroscopique de la rétine*

I.7. Zones particulières de la Rétine

I.7.1. La macula

La macula est la zone centrale de la rétine ou fond d'œil et c'est le point où l'acuité visuelle est à son maximum. Une macula intacte permet une vision centrale claire, la distinction des détails, la lecture et la reconnaissance, par exemple, des visages des personnes. [5]

I.7.2. La fovéa

La fovéa est la zone centrale de la macula, est la zone de la rétine où la vision des détails est la plus précise. Elle est située dans l'axe visuel de l'œil.

I.7.3. La pupille

C’est une région d’émergence du nerf optique, et dépourvue de photorécepteurs, cette partie représente le diaphragme de l’œil, avec la capacité de se contracter ou se dilater selon l’intensité lumineuse.

I.8. L’arbre de rétinien

L'arbre vasculaire rétinien est composé des vaisseaux sanguins (artères et veines) qui irriguent la rétine. Son étude est importante pour détecter plusieurs pathologies oculaires.

Certaines caractéristiques de l'arbre vasculaire rétinien permettent de diagnostiquer des maladies comme la rétinopathie hypertensive. En phase active, les vaisseaux prennent un aspect de manchon blanc jaunâtre à bords flous et la lumière vasculaire apparaît rouge avec une constriction variable

Des méthodes de traitement d'images permettent de segmenter automatiquement l'arbre vasculaire rétinien à partir de photographies du fond d'œil. Cela permet de localiser précisément les vaisseaux, d'en mesurer les diamètres et d'identifier les bifurcations et croisements.

En résumé, l'analyse de l'arbre vasculaire rétinien, que ce soit cliniquement ou par traitement d'images, est un outil précieux pour le diagnostic et le suivi de nombreuses pathologies oculaires et systémiques [6]

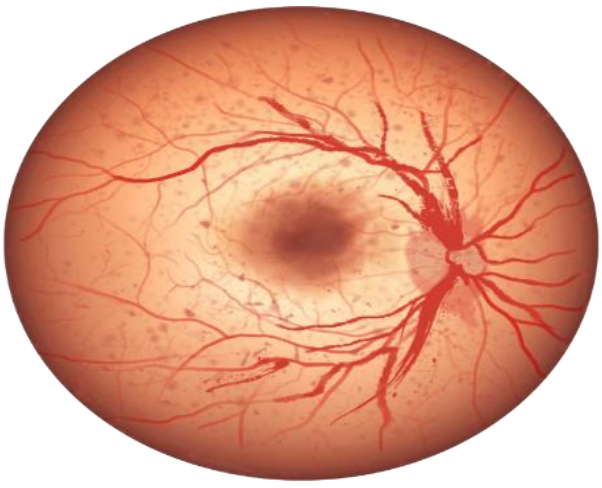


Figure 1.6 : *L’arbre de rétinien*

I.9. Pathologie

C’est des maladies et affections rétiniennes sont celles qui affectent la rétine, la couche sensible à la lumière située au fond de l’œil. L’importance de la rétine dans la vision réside dans sa capacité à convertir la lumière en signaux neuronaux transmis au cerveau. [7] Voici quelques-unes des maladies rétiniennes les plus courantes :

I.10. Pathologies de la rétine

Plusieurs pathologies peuvent affecter la rétine et entraîner une altération de la vue :

I.10.1.1. La DMLA (Dégénérescence maculaire liée à l'âge)

Un vieillissement prématuré de la rétine qui se traduit par une perte de vision centrale.

I.10.1.2. La rétinopathie diabétique

Il s'agit d'une complication fréquente du diabète sucré qui compromet le bon fonctionnement du globule rouge. Elle est causée par la détérioration des vaisseaux sanguins de la rétine. La dilatation de ces vaisseaux modifiés peut provoquer une fuite de liquide (plasma, lipides et/ou sang) et même s'obstruer, ce qui laisse une partie de la rétine sans circulation sanguine. Chacune de ces manifestations du diabète peut causer des dommages graduels aux structures du globe oculaire, ce qui peut entraîner une perte significative de la vision et même, en l'absence de traitement approprié, la cécité. Bien que très avancée, la rétinopathie diabétique ne provoque pas toujours une altération de la vision. Des examens ophtalmologiques réguliers sont donc recommandés aux patients diabétiques.

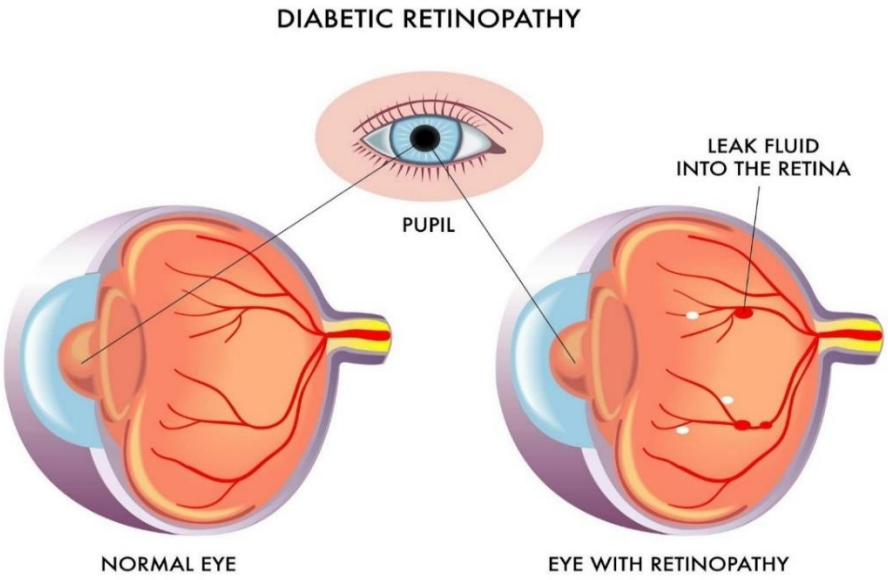


Figure 1.7 : *Complications oculaires du diabète*

I.10.1.3. Le décollement de la rétine

Qui est en fait un dédoublement intra rétinien et constitue une urgence médicale.

Il existe différentes méthodes de traitement au laser qui peuvent être utilisées en fonction du problème, de sa gravité et de sa localisation sur la rétine. La rétine et ses vaisseaux sanguins sont directement touchés par le laser.

I.11. Photocoagulation

Dans le domaine de l'ophtalmologie, la photocoagulation est une méthode qui permet de créer des cicatrices dans la rétine en utilisant une brûlure thérapeutique à l'aide d'un faisceau laser. La cicatrice qui en découle permet d'empêcher la progression de certaines affections rétiniennes. Il s'agit de faire appel à l'action cicatrisante et coagulante d'un laser dont le rayon est dirigé spécifiquement vers la rétine.[8] Cette méthode à trois objectifs différents :

- Au moment de la cicatrisation, il est important de créer un accouplement des différentes couches de la rétine pour prévenir, par exemple, l'extension d'une déchirure rétinienne.
- Réunir des vaisseaux qui échappent ou qui présentent des anomalies. Il est important de détruire avec précision les zones malades de la rétine qui, sans traitement, pourraient causer une perte de vision.
- L'opération est réalisée avec un verre de contact et un microscope laser. Parmi les autres Les maladies les plus utilisées sont :
    - **Les fractures de la rétine :** Il est essentiel de les traiter dès que possible. Au contraire, il est possible que du liquide s'accumule sous les déchirures. Le processus de photo-coagulation interrompt la progression de ces déchirures pour prévenir les décollements de la rétine.

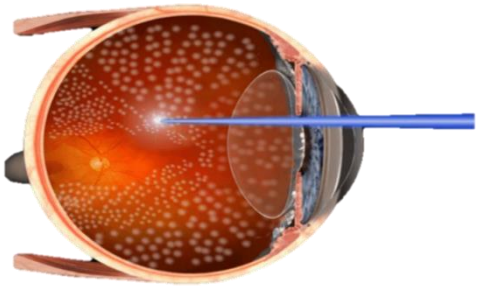


Figure 1.8 : *laser de correction*

I.11.1. Les rétinopathies vasculaires

La rétinopathie diabétique, les occlusions des veines ou les pathologies exsudatives comme la vitré rétinopathie exsudative familiale sont des exemples. La photocoagulation permet de séparer les vaisseaux sanguins ou les tissus anormaux, de diminuer en partie la croissance de nouveaux vaisseaux sanguins anormaux et de faciliter l'absorption du liquide.

Dans la rétinopathie diabétique l'objectif principal est de détruire une grande partie de la rétine privée d'oxygène (ischémie) pour éviter la formation de nouveaux vaisseaux sanguins (néovascularisation) et éviter une hémorragie du vitré, qui nécessiterait une intervention chirurgicale (vitrectomie). .

I.11.1.1. Glaucome

La photocoagulation réduit la pression oculaire tout en facilitant l'issue de l'humeur aqueuse dans les cas de glaucome.

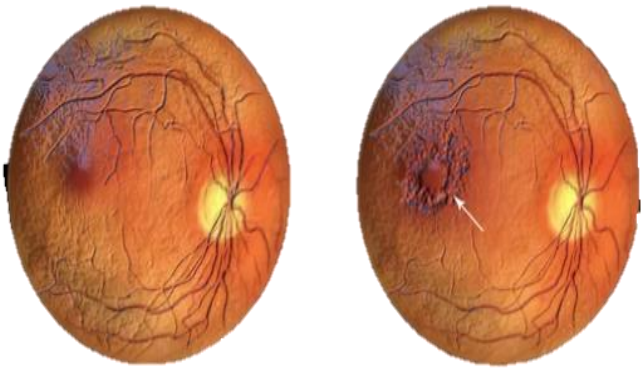
I.11.1.2. Tumeurs

La photocoagulation peut arrêter la croissance des tumeurs rétiniennes ou même les réduire.

I.11.1.3. L'œdème maculaire

Dans le cas de l'œdème maculaire, le traitement au laser est limité à des cas très particuliers, les injections intra vitréennes restant à ce jour le traitement initial.

Diabétique Rétinopathie



Normal rétine Macula œdème

Figure 1.9 : *macula oedem*

I.12. Historique L'intelligence artificielle

Depuis les années 1950, les chercheurs en intelligence artificielle (IA) s'efforcent de créer des systèmes capables de traiter des données visuelles, menant au développement de la vision artificielle. En 2012, une avancée majeure a été réalisée par une équipe de l'université de Toronto, qui a développé Alex Net, un réseau de neurones artificiels. Lors du concours Image Net de cette année-là, Alex Net a réussi à surpasser les performances humaines en reconnaissance d'images, atteignant une précision de 85 %. Cette victoire a marqué un tournant significatif dans le domaine de la vision artificielle et a démontré le potentiel des réseaux de neurones pour des tâches complexes de reconnaissance visuelle. [9]

I.13. Intelligence artificielle

L'intelligence artificielle est une branche de l'informatique dédiée à la création de machines capables de simuler l'intelligence humaine. Son objectif est de développer des programmes exécutant des tâches typiquement humaines. L'apprentissage machine, une sous-discipline de l'IA, comprend l'apprentissage profond, qui permet des analyses et des décisions complexes basées sur de grandes quantités de données.

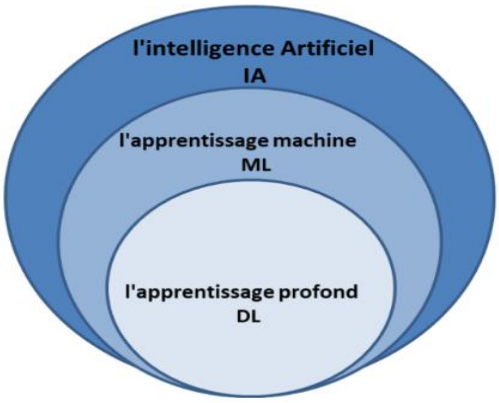


Figure 1.10 : *l'intelligence artificielle.*

I.14. La classification

La classification des images est une technique de vision par ordinateur qui consiste à identifier et catégoriser le contenu visuel d'une image en différentes classes. Utilisant des bibliothèques comme Tensorflow, ce processus débute par la collecte d'un ensemble de données, suivi d'un prétraitement pour améliorer la qualité des données pour l'analyse. Les données sont ensuite divisées en ensembles de formation et de test. Des modèles de réseaux de neurones convolutifs (CNN) sont alimentés avec ces données pour apprendre à reconnaître les caractéristiques visuelles spécifiques. Finalement, le modèle entraîné est capable de classer de nouvelles images, par exemple, identifier des pathologies sur des images de la rétine. [10]

I.15. La segmentation d'image

La segmentation d'images est une méthode de vision par ordinateur qui subdivise une image numérique en groupes de différents pixels (segments d'image) afin que les objets et les tâches associées soient plus faciles à détecter. En fragmentant les informations visuelles complexes d'une image en segments de forme précise, la segmentation d'image permet un traitement d'image plus rapide et plus sophistiqué. Les méthodes de segmentation d'images sont variées, allant de l'analyse heuristique simple et intuitive à l'utilisation d'un apprentissage profond de pointe.[11]

I.16. La détection

La détection d'objets consiste à identifier et à localiser les objets d'intérêt présents dans les images. Contrairement à la classification, qui fournit une étiquette pour l'image entière,

la détection fournit des données sur l'emplacement (généralement sous la forme de cadres englobants) et la catégorie de l'objet.

I.17. La différence entre segmentation et détection et classification [12]

Tableau 1.1 : Différence entre les segmentation et classification.

Technique	Principales caractéristiques	Cas d'utilisation
Segmentation	-Fournir des informations précises sur les limites et les régions des objets. -permet une détection précise des défauts de fabrication -Permet une identification précise des tumeurs en imagerie médicale.	-analyse d'images médicales. -détection de défauts dans les procédés de fabrication.
Détection d'objet	-identifie des objets spécifiques et leurs emplacements -Utilise des cadres de délimitation pour une localisation précise des objets. -Permet la vidéosurveillance et la surveillance de la sécurité. -Aide en agriculture à la surveillance des cultures et à la détection des parasites.	-vidéosurveillance et sécurité. -suivi agricole et des récoltes.
Classification	-attribuer des étiquettes aux images ou aux régions. -fournit une compréhension complète du contenu de l'image -Permet le marquage et l'étiquetage des images.	-marquage et étiquetage d'images. -Reconnaissance faciale. -diagnostic de maladies en imagerie médicale.

I.18. Le traitement d'images

Le traitement d'images par l'IA permet d'identifier différentes classes d'objets et d'optimiser les images en conséquence. Notre logiciel de traitement d'images par IA peut reconnaître les personnes, les visages et déterminer si la photo a été prise en intérieur ou en extérieur. Il est également capable de recadrer automatiquement les images dans des formats prédéfinis tout en gardant la mise au point sur l'objet dominant. Cela assure que les éléments principaux restent au centre de l'attention, améliorant ainsi la qualité et la pertinence des images.

Choisir l'IA pour le traitement d'images permet une identification précise des objets, des personnes et des visages, optimisant ainsi la qualité des photos. L'IA détermine automatiquement le contexte de prise de vue (intérieur ou extérieur) et recadre les images en conservant la mise au point sur l'objet principal. Cela améliore l'efficacité et la pertinence des images, offrant des résultats professionnels de manière rapide et fiable.[13]

I.19. Méthode traitement

Dans le but d'améliorer la qualité des images numériques ou d'extraire des informations à partir d'images numériques, diverses méthodes et algorithmes peuvent être utilisés, qui se divisent en deux catégories : les méthodes classiques et les méthodes intelligentes.[14]

Figure 1.11 : *méthode traitement*

I.19.1. Méthodes Classiques de Segmentation

Les techniques traditionnelles de segmentation d'images peuvent être divisées en plusieurs catégories principales, chacune avec ses méthodes et applications spécifiques dans le **Tableau 1.2** .[14]

Tableau 1.2 : Déférent méthode classique.

Méthode	Description	Exemples	Applications
Seuillage	Sépare les pixels par intensité.	Seuillage global, adaptatif	Images à contraste élevé
Clustering	Partitionne les pixels en groupes homogènes.	K-means, Fuzzy C-means	Images médicales, satellites
Contours Actifs	Utilise des courbes pour détecter les bords des objets.	Contours de Kass, Chan-Vese	Objets avec des bords bien définis (IRM)
Régions	Regroupe les pixels adjacents similaires.	Région croissante, division-fusion	Objets clairement séparés par caractéristiques régionales
Transformée de Watershed	Considère l'image comme une topographie pour segmenter les objets.	-	Gradients marqués, objets adjacents avec intensités similaires

I.19.2. Méthodes moderne de Segmentation

Des techniques avancées d'apprentissage automatique et de traitement d'image sont employées dans les méthodes modernes de segmentation d'images afin d'améliorer la précision et la solidité des résultats de segmentation. Voici une liste synthétique des principales techniques de segmentation contemporaines, incluant leur description, des exemples et leurs applications dans le **Tableau 1.3** :

Tableau 1.3 : différent méthode moderne.

Méthode	Description	Exemples	Applications
Réseaux de Neurones Convolutifs (CNN)	Utilisent des couches de convolution pour extraire des caractéristiques de l'image.	U-Net, SegNet, Fully Convolution Networks (FCN)	Segmentation d'images médicales, reconnaissance d'objets
Segmentation Sémantique	Assigne une étiquette à chaque pixel en fonction de la classe d'objet.	DeepLab, PSPNet U-Net	Voitures autonomes, analyse d'images de scènes urbaines
Segmentation d'Instances	Différencie les instances distinctes d'objets de la même classe.	Mask R-CNN, SOLO v2	Voitures autonomes, analyse de vidéos de surveillance
Segmentation Panoptique	Combine la segmentation sémantique et d'instances en un seul cadre unifié.	Panoptic FPN, UPSNet	Applications nécessitant à la fois segmentation sémantique et d'instances
Segmentation Basée sur GANs	Utilisent des Réseaux Adversaires Génératifs pour améliorer la qualité de la segmentation.	GAN-Segmentation, Pix2Pix	Amélioration de la segmentation dans des images à faible contraste
Apprentissage Auto-supervisé	Utilise des techniques d'auto-supervision pour apprendre sans annotations étendues.	SimCLR, BYOL	Segmentation dans des contextes avec peu de données annotées

I.20. La différence entre segmentation moderne et classique

Voici le **Tableau 1.4** compact résumant les différences entre les méthodes classiques et modernes de segmentation d'images :

Tableau 1.4 : Comparaison entre les deux méthodes.

Critère	Méthodes Classiques	Méthodes Modernes
Base Théorique	Techniques traditionnelles de traitement d'image	Apprentissage automatique et profond
Complexité	Simples à implémenter, rapides	Complexes, nécessitent des ressources importantes
Adaptabilité	Ajustements manuels nécessaires	Apprennent à partir de données annotées, meilleure adaptation
Dépendance aux Données	Peuvent fonctionner avec peu de données	Apprennent à partir de données annotées, meilleure adaptation
Dépendance aux Données	Peuvent fonctionner avec peu de données	Nécessitent de grandes quantités de données pour l'entraînement
Exemples de Méthodes	Seuillage, K-means, Contours actifs, Watershed	U-Net, DeepLab, Mask R-CNN, Vision Transformers, GAN-Segmentation
Exemples de Méthodes	Tâches simples, images avec caractéristiques bien définies	Applications complexes, haute précision requise
Applications	Tâches simples, images avec caractéristiques bien définies	Applications complexes, haute précision requise
Robustesse et Précision	Moins robustes, précision limitée	Très robustes, haute précision
Évolution Technologique	Technologies établies et bien comprises	État de l'art, avancées constantes

En bref, les techniques de segmentation traditionnelles sont souvent plus simples, plus rapides et nécessitent moins de données pour leur fonctionnement, mais elles peuvent présenter des limites en termes de précision et de capacité à gérer des images complexes. Quant aux méthodes contemporaines, elles reposent sur des techniques d'apprentissage profondes et permettent une des performances et de l'adaptabilité, mais elles sont plus complexes et davantage de données d'entraînement.

I.21. L'intelligence artificielle embarquée

I.21.1. Définition

L'intelligence artificielle embarque ou "Edge AI", désigne l'intégration et l'exécution d'algorithmes d'IA directement sur des dispositifs locaux tels que smartphones, caméras de sécurité, ou microcontrôleurs, sans dépendre d'une connexion au cloud. Cette approche permet de traiter les données sur place, réduisant ainsi la latence, économisant la bande passante, et améliorant la confidentialité des données. Elle est particulièrement utile dans les applications nécessitant des réponses rapides et fiables, comme les systèmes de transport autonome et la domotique. En outre, elle favorise une meilleure gestion de l'énergie et une autonomie accrue des appareils en réseau. [15]

Les FPGA (Field-Programmable Gâte Arrays), Arduino, et Raspberry Pi jouent des rôles distincts dans le domaine de l'intelligence artificielle embarquée. Chacun apporte des capacités uniques, adaptées à différents niveaux de complexité de projet et de performance requise. Voici un aperçu détaillé de leur rôle dans l'IA embarquée :

I.21.2. Arduino

- **Accessibilité et simplicité**

Arduino est une plateforme de microcontrôleur très accessible et facile à utiliser, idéale pour les hobbyistes, les éducateurs, et les prototypes simples d'IA.

- **Projets d'IA à petite échelle**

Bien que les capacités de traitement des Arduino soient limitées, ils peuvent être utilisés pour des projets d'IA embarquée simples, comme la reconnaissance de gestes ou la commande vocale de base, souvent en association avec des modules externes.

- **Éducation et initiation à l'IA**

Arduino offre un point de départ excellent pour apprendre les bases de l'IA et de la robotique, grâce à une grande communauté et une multitude de ressources et de projets partagés.[15]

I.21.3. Raspberry Pi

- **Puissance et polyvalence**

Le Raspberry Pi est un mini-ordinateur à part entière qui offre assez de puissance de calcul pour exécuter des algorithmes d'IA plus complexes, comme la reconnaissance faciale ou le traitement du langage naturel.

- **Support de logiciels d'IA**

Le Raspberry Pi peut exécuter divers systèmes d'exploitation et logiciels, y compris des bibliothèques d'IA populaires comme TensorFlow ou PyTorch, le rendant adapté pour des projets d'IA plus avancés.

- **Applications IoT et domestiques**

Grâce à sa capacité à se connecter facilement à Internet et à d'autres appareils, le Raspberry Pi est souvent utilisé dans des projets de domotique, de surveillance de sécurité, et d'autres applications IoT intégrant l'IA.[15]

I.21.4. FPGA (Field-Programmable Gate Array)

- **Flexibilité et performance**

Les FPGA sont des circuits intégrés configurables qui peuvent être programmés pour exécuter des tâches spécifiques, y compris des algorithmes d'IA, avec une efficacité énergétique et des vitesses élevées. Ils sont particulièrement utiles pour des applications nécessitant un traitement en temps réel.

- **Personnalisation du matériel**

Contrairement aux CPU et GPU, les FPGA permettent la configuration du matériel pour optimiser spécifiquement certaines fonctions d'IA, comme la convolution dans les réseaux de neurones, améliorant ainsi les performances.

- **Utilisation dans les applications industrielles**

Les FPGA sont souvent utilisés dans des scénarios industriels pour le contrôle de processus et la vision par ordinateur, grâce à leur capacité à gérer des flux de données en continu et à effectuer des calculs rapides.

- **En résumé :**

Les FPGA offrent une personnalisation de haut niveau et sont idéaux pour des applications industrielles exigeantes, tandis que l'Arduino est plus adapté pour l'éducation et les projets d'IA de petite échelle. Le Raspberry Pi se situe entre les deux, offrant une bonne puissance de calcul et la flexibilité pour gérer des projets d'IA plus complexes et des applications IoT.

I.22. Les réseaux de neurones

- **Définition**

Les réseaux de neurones, couramment appelés réseaux de neurones artificiels, sont des modèles simplifiés des fonctions des neurones dans le cerveau humain, utilisés pour résoudre des problèmes d'apprentissage automatique (Machine Learning). [16]

- **L’historique**

Le concept des réseaux de neurones artificiels a été inventé en 1943 par Warren McCullough et Walter Pitts de l'Université de Chicago. Ils ont présenté leur théorie sur l'activation des neurones comme unité de base de l'activité cérébrale dans un article du journal Brain Theory. En 1957, le Perceptron, le premier algorithme de Machine Learning, a été créé pour effectuer des tâches de reconnaissance de patterns complexes, permettant ultérieurement aux machines d'apprendre à reconnaître des objets sur des images.

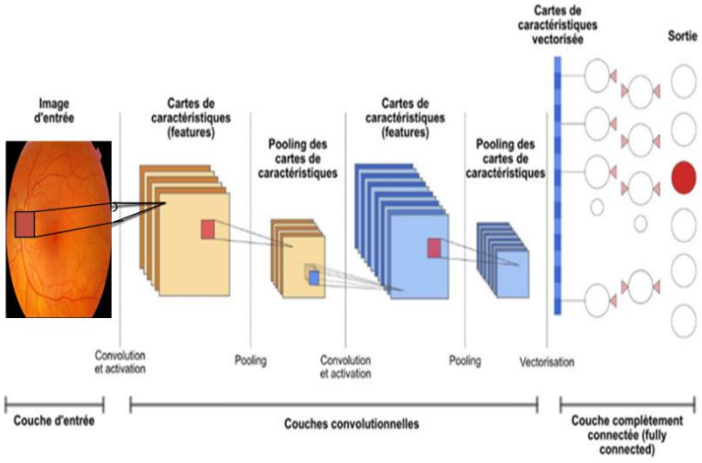


Figure 1.12 : *Réseaux neuronaux classiques pour reconnaissance d'image.*

- **Réseau de neurones convolutifs**

Les réseaux convolutifs, inspirés du cortex visuel des mammifères, sont des réseaux neuronaux multicouches utilisés pour la catégorisation d'informations simples à complexes. Ils prétraitent des petites quantités d'informations à travers des couches convolutionnelles basées sur la convolution mathématique. Chaque couche extrait des caractéristiques de plus en plus complexes d'une image, des contours aux objets entiers. Leur efficacité est optimisée par une phase d'apprentissage avec des données connues. Ces réseaux ont de nombreuses applications, notamment dans la reconnaissance d'images, de vidéos et le traitement du langage naturel. [16]

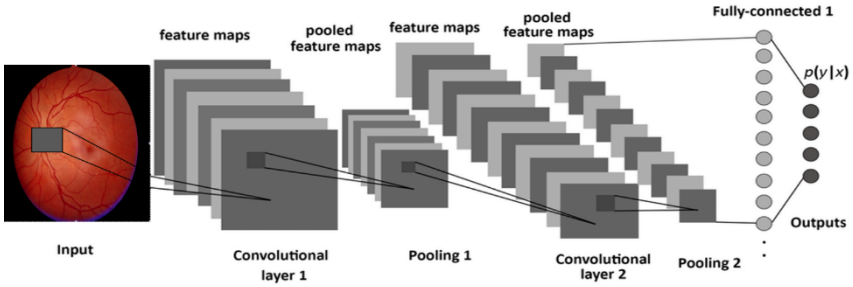


Figure 1.13 : *Réseau de neurones convolutifs*

- **Convolution**

La convolution, un outil mathématique utilisé pour le traitement d'image, permet aux réseaux de neurones convolutionnels de bien fonctionner pour la reconnaissance d'images. Elle agit comme un filtre en balayant l'image avec une fenêtre définie, se déplaçant horizontalement et verticalement pour parcourir toute l'image. [17]

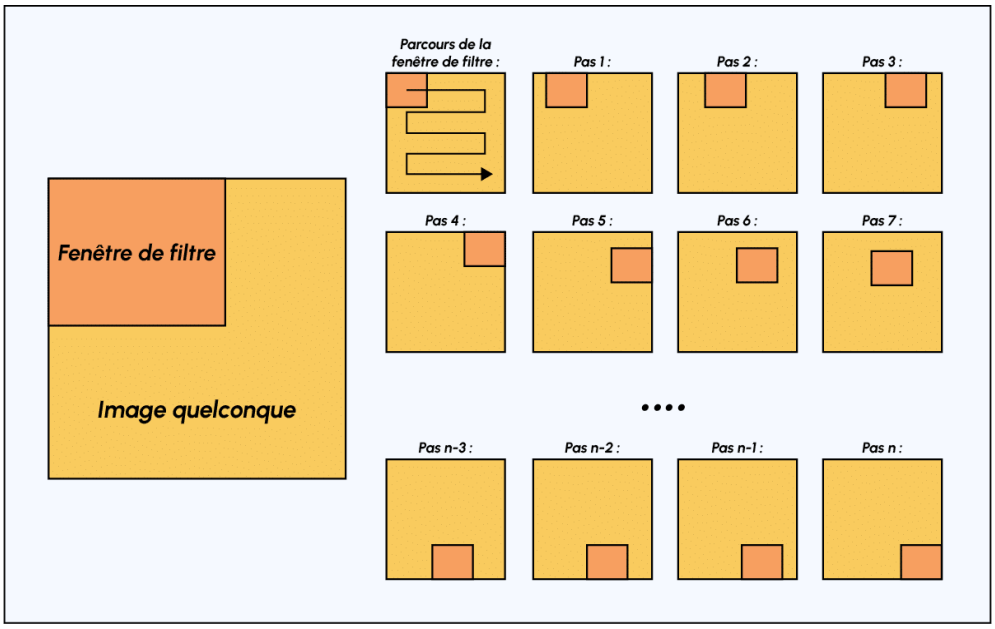


Figure 1.14 : *Schéma du parcours de la fenêtre de filtre sur l'image*

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 représente 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).

- **Pooling**

La couche de pooling (ou couche de sous-échantillonnage) est appliquée après une couche convolutionnelle pour réduire la dimensionnalité. Les types les plus courants sont le max pooling, qui prend la valeur maximale, et l'average pooling, qui calcule la moyenne des valeurs. [18]

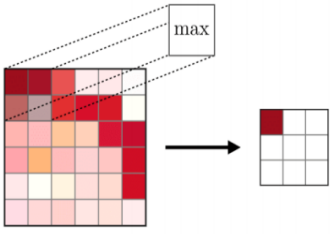
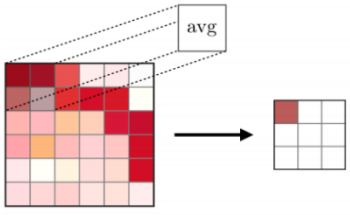
Max pooling	Moyenne pooling
Chaque opération de pooling sélectionne la Valeur maximale de la surface	Chaque opération de pooling sélectionne la Valeur moyenne de la surface
	
• Garde les caractéristiques détectées • Plus communément utilisé	• Sous-échantillonne la feature map • Utilisé dans LeNet

Figure 1.15 : *Max et moyenne pooling.*

- **Normalisation par lot :**

Dans les réseaux de neurones, la normalisation par lots (batch normalisation) est une méthode employée afin de stabiliser et accélérer l'apprentissage. Les activations des couches intermédiaires sont normalisées en les recentrant et en les ajustant à chaque mini-lot de données. Cela diminue le décalage interne de la covariance et améliore la convergence du modèle. Elle comprend des réglages de décalage et d'échelle accessibles, offrant au réseau la possibilité de représenter des distributions de données complexes. [19]

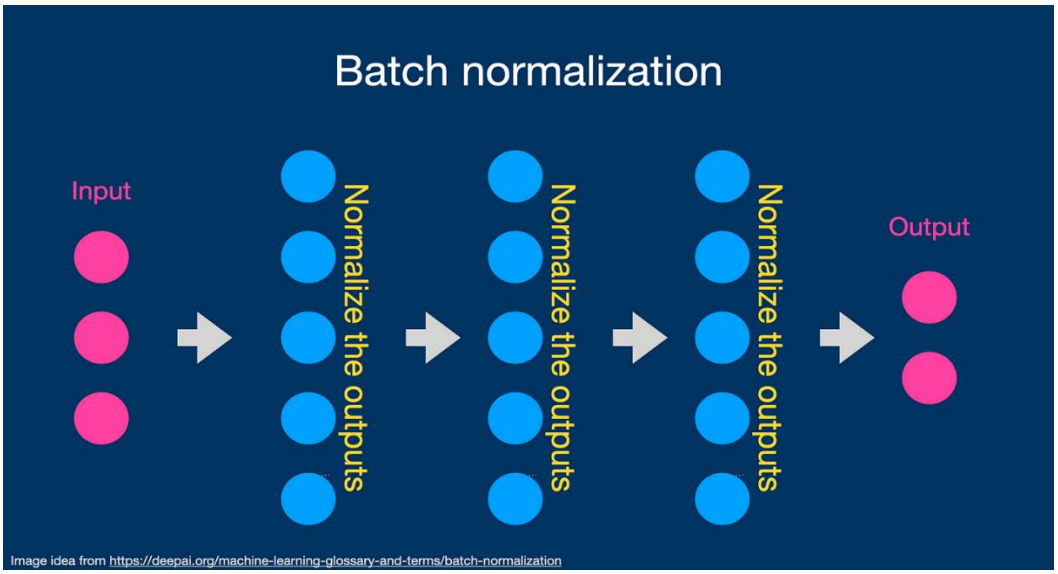


Figure 1.16: *Normalisation par lot*

- **Entièrement connecter**

La couche entièrement connectée (ou FC) est utilisée pour une entrée préalablement aplatie où chaque entrée est reliée à tous les neurones. Il est courant de trouver des canapés d'entièrement connectés à la fin des architectures de CNN, ce qui permet d'optimiser des objectifs tels que les scores de classe. [20]

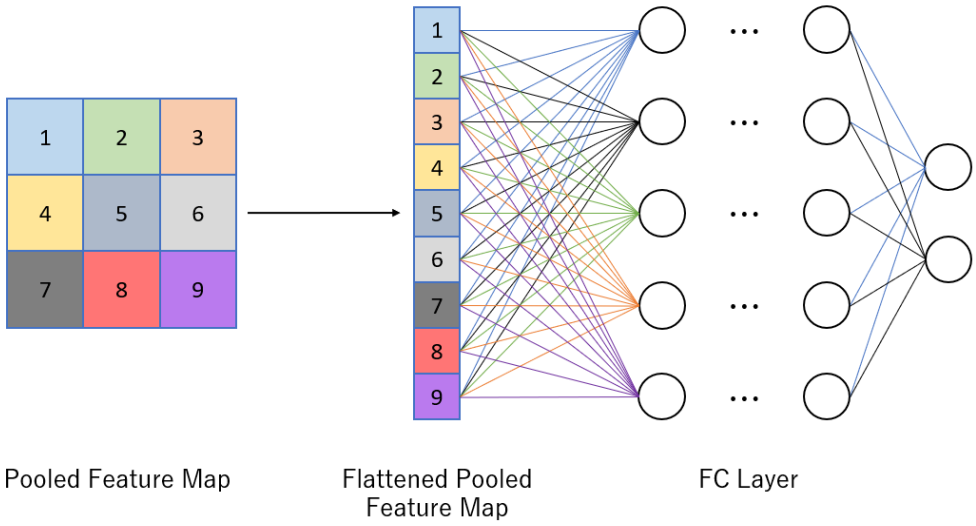


Figure 1.17: *entièrement connectés*

I.23. Architectures CNN populaires

I.23.1. Le modelé 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.[21]

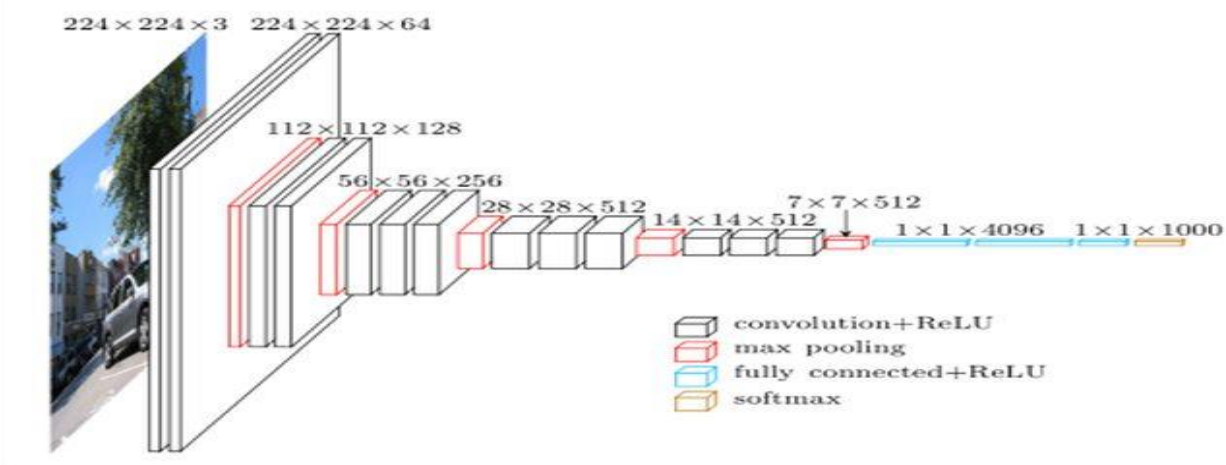


Figure 1.18 : *Architecture VGG*

I.23.2. Le Modèle Alex Net

Alex Net est une architecture de réseau neuronal convolutif classique. Il se compose de convolutions, d'une mise en commun maximale et de couches denses comme éléments de base. Des convolutions groupées sont utilisées pour ajuster le modèle sur deux GPU [21].

Alex net elle est plus profonde avec plus de filtres par couche. Il se compose de huit

couches : cinq couches convolutives (certaines d'entre elles sont suivies de couches de regroupement maximal), deux couches cachées entièrement connectées et une couche de sortie entièrement connectée. La formation de ce réseau est également réalisable sur plusieurs GPU.

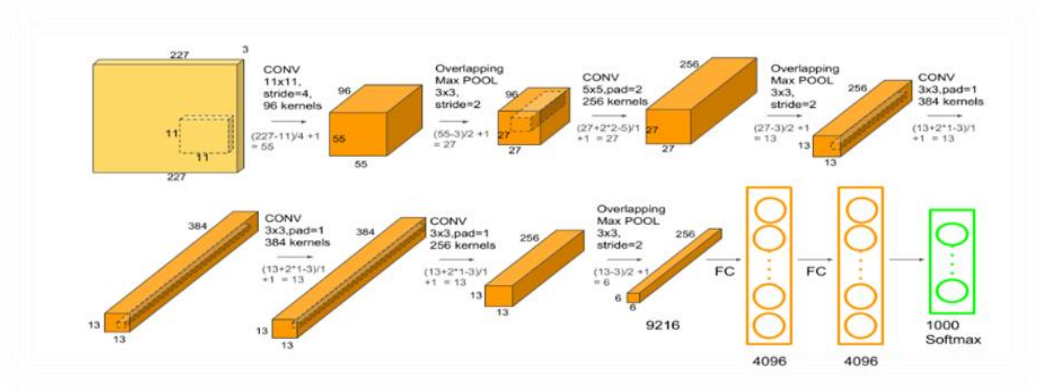


Figure 1.19 : *Architecture Alex Net*

I.23.3. Le Modèle Google Net

Google Net est un réseau de neurones à convolution profonde à 22 couches qui est une variante du réseau Inception, un réseau de neurones à convolution profonde développé par des chercheurs de Google. L'architecture Google Net présentée dans le Image Net Large-Scale Visual Recognition Challenge 2014 (ILS RC14) a résolu des tâches de vision par ordinateur telles que la classification d'images et la détection d'objets. [21]

L'architecture Google Net est très différente des architectures de pointe précédentes telles qu'Alex Net et ZF-Net. Il utilise de nombreux types de méthodes telles que la convolution 1 × 1 et la mise en commun des moyennes globales qui lui permettent de créer une architecture plus profonde

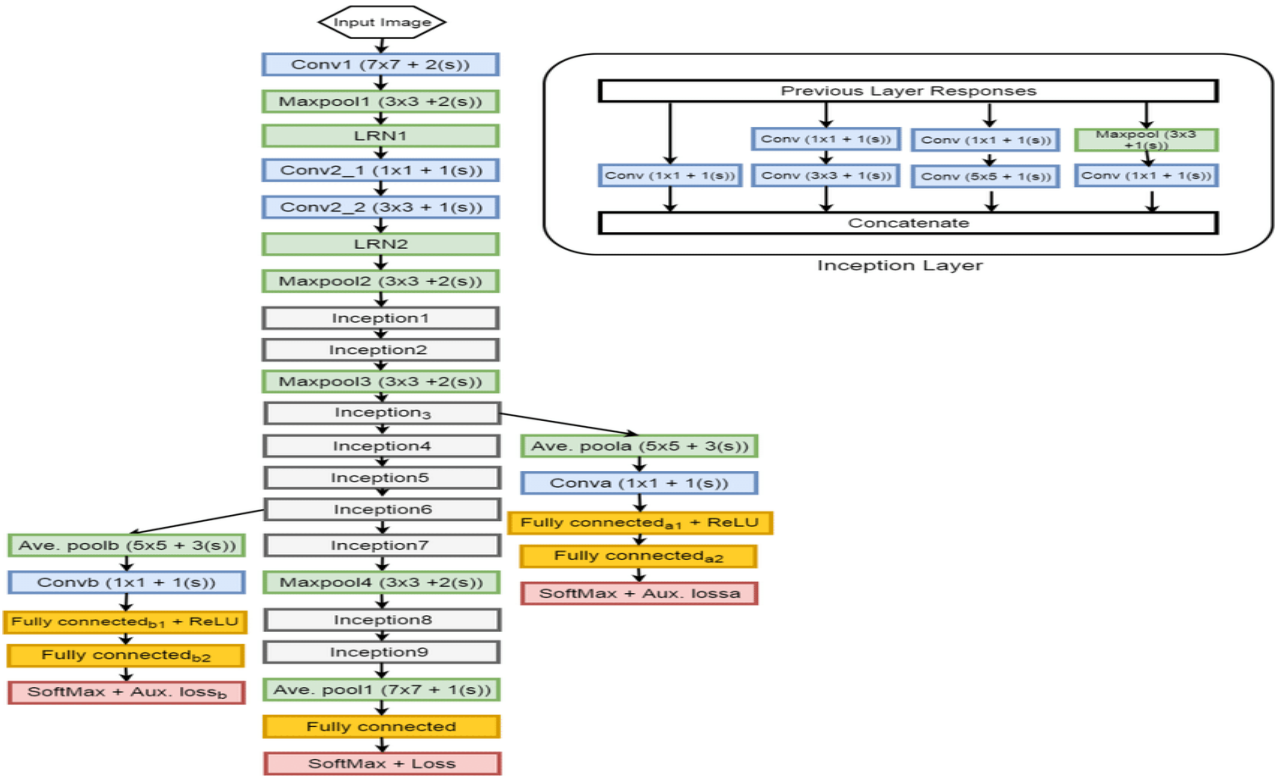


Figure 1.20 : *Architecture du Google Net*

I.24. Conclusion

Ce chapitre met en lumière l'anatomie complexe de l'œil humain et son interrelation avec la technologie, notamment à travers l'étude détaillée de la rétine et de son rôle essentiel dans la transformation des signaux lumineux en impulsions nerveuses. Il explore également l'application des technologies de vision par ordinateur, utilisant des réseaux de neurones convolutifs et Ténor Flow, pour la segmentation et l'analyse d'images. Cette convergence entre biologie et technologie ouvre des perspectives pour le diagnostic de pathologies rétiniennes et autres applications pratiques, démontrant l'intégration profonde entre l'ophtalmologie et l'intelligence artificielle pour relever des défis médicaux et technologiques modernes.

Chapitre 02 :

Conception d'une architecture de Deep Learning pour la segmentation de l’arbre rétinien

II.1. Introduction

Dans ce chapitre, on explore la création d'une architecture de deep Learning pour segmenter les vaisseaux rétiniens, en se basant sur U-Net, une structure performante pour les images biomédicales.

U-Net présente une forme en forme de "U" avec une section destinée à réduire la taille des images et une autre section destinée à la restaurer, ce qui permet une segmentation précise. De plus, nous examinons le traitement préliminaire des images rétiniennes et l'analyse des données afin d'améliorer les performances du modèle.

II.2. Principe de segmentation de l'arbre rétinien

II.2.1. Etapes de conception

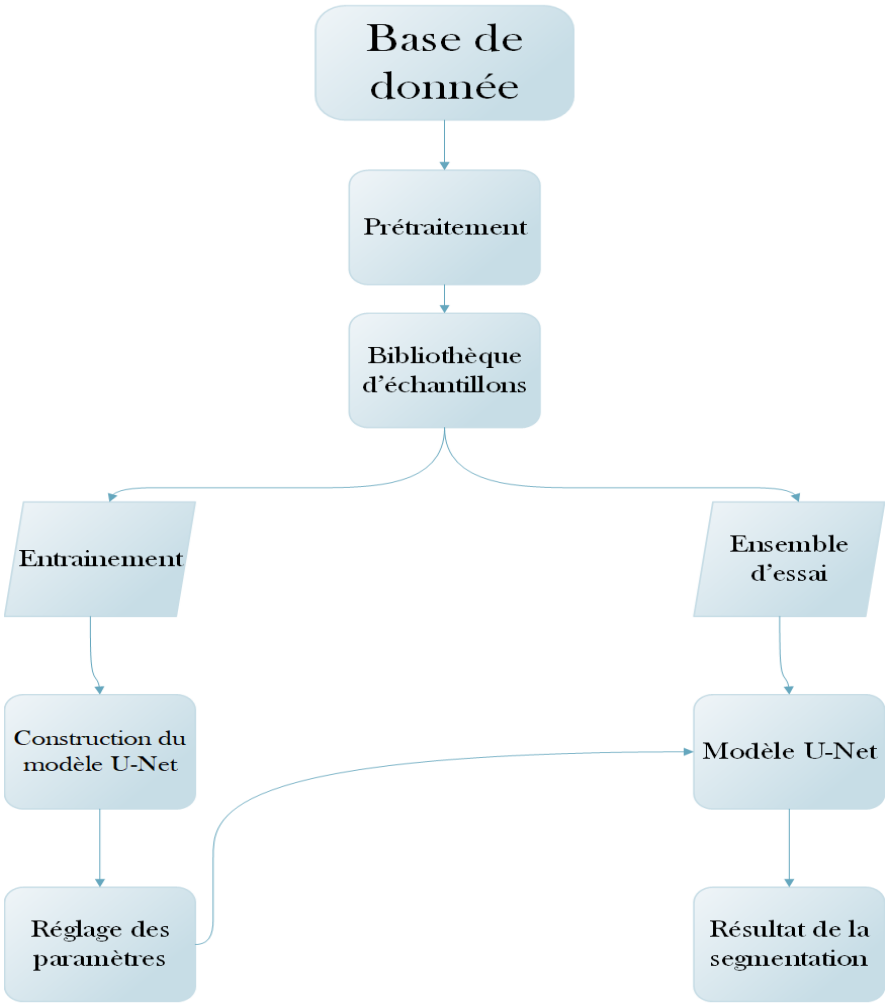


Figure 2.1 : *étape de conception*

II.3. Base de données

Notre ensemble de données se compose de 320 images rétiniennes, chacune accompagnée de son masque correspondant. Les images ont une taille de 256*256 pixels et sont utilisés pour entraîner et tester notre modèle de segmentation.

II.3.1. Description de DRIVE dataset

Le jeu de données DRIVE est largement utilisé dans le domaine de la segmentation rétinienne. Il est constitué d'images rétiniennes numériques en couleur accompagnées de masques binaires annotés. Ces masques sont essentiels pour extraire et segmenter les vaisseaux sanguins de la rétine, en marquant les pixels des vaisseaux comme 1 et les autres pixels comme 0. [22]

II.3.2. Dataset DRIVE comprend

II.3.2.1. Images

Il contient un ensemble d'images rétiniennes numériques en couleur, généralement au format JPEG ou PNG. Chaque image représente une partie de la rétine d'un œil.

II.3.2.2. Masques

Chaque image est associée à un masque binaire qui identifie les pixels appartenant aux vaisseaux sanguins de la rétine. Les pixels des vaisseaux sont marqués comme 1 dans le masque, tandis que les autres pixels sont marqués comme 0.

Utilisation du jeu de données dans notre travail :

Nous avons indiqué avoir utilisé une partie du jeu de données DRIVE pour notre étude :

- Nous avons sélectionné 80 images avec leurs masques correspondants.
- Nous avons appliqué la technique de patchification en découpant chaque image en 4 morceaux de taille 256x256 pixels pour les images et 256x256 pixels pour les masques, ce qui a abouti à un total de 320 images et 320 masques.

Cette approche de patchification est couramment employée pour optimiser l'entraînement des modèles de segmentation, améliorant ainsi la gestion des données et facilitant l'analyse locale des caractéristiques.

Exemple d'image de DRIVE dataset et son masque :

Des exemples :

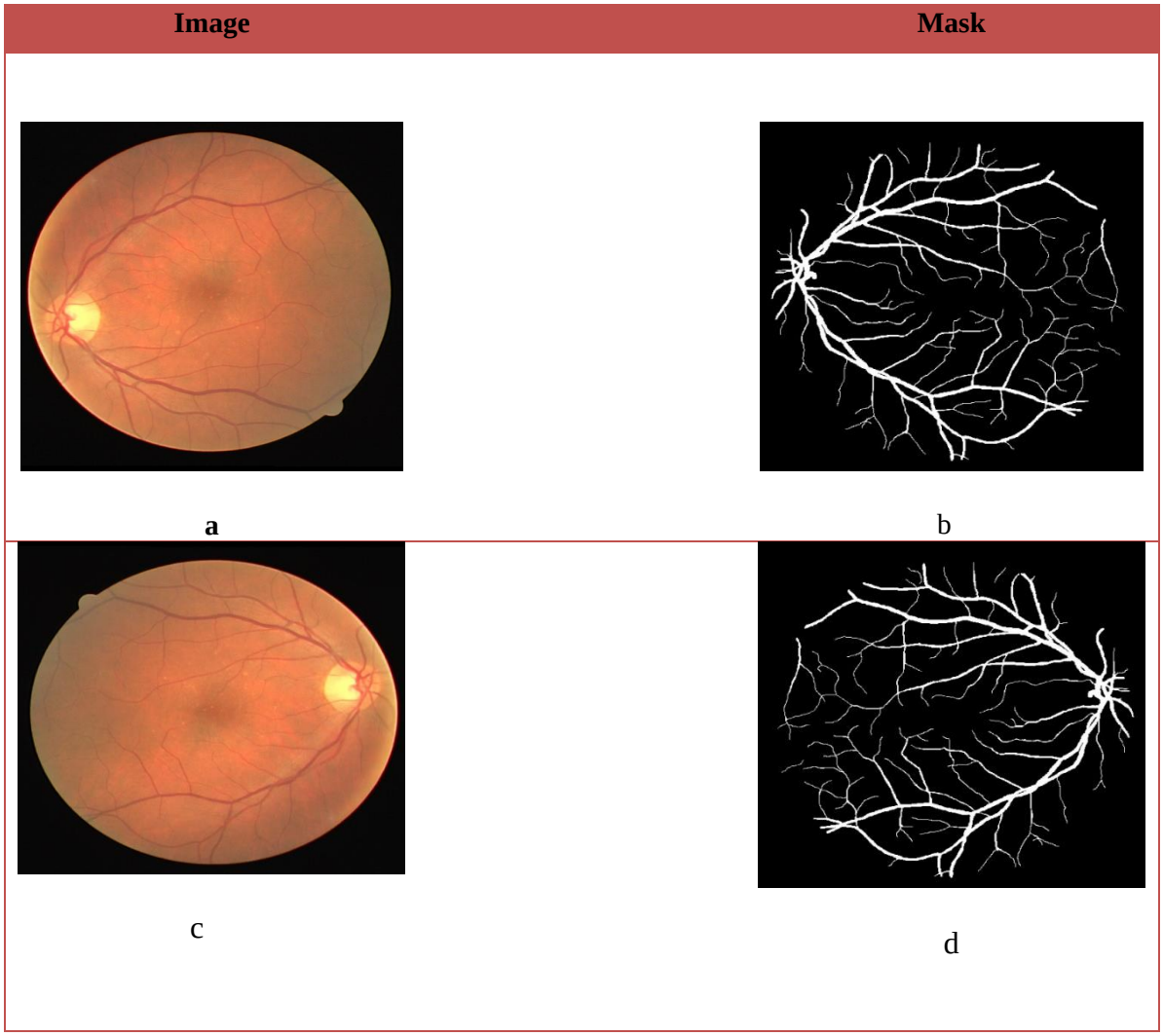


Figure 2.2 : *Deux image de DRIVE dataset et son masque*

II.3.3. Prétraitement des Données

Avant d'entraîner notre modèle, nous avons effectué un prétraitement sur les images rétiniennes pour garantir leur qualité et leur cohérence. Cela inclut la normalisation des intensités, la réduction du bruit et l'ajustement de la luminosité et du contraste.

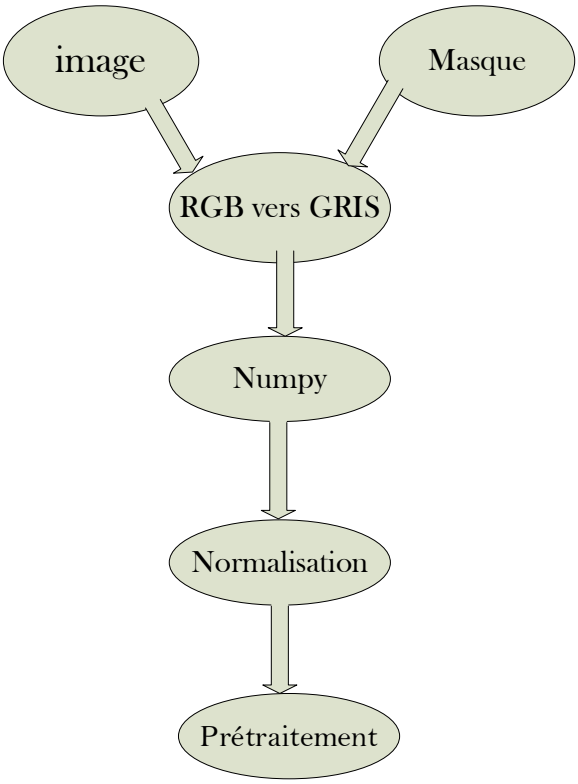


Figure 2.3 : *prétraitement*

II.3.4. Augmentation des Données

Afin d'enrichir notre ensemble de données et d'améliorer la capacité de généralisation de notre modèle, nous avons appliqué des techniques d'augmentation de données telles que la rotation, le décalage, le retournement et le zoom sur les images et leurs masques correspondants [23].

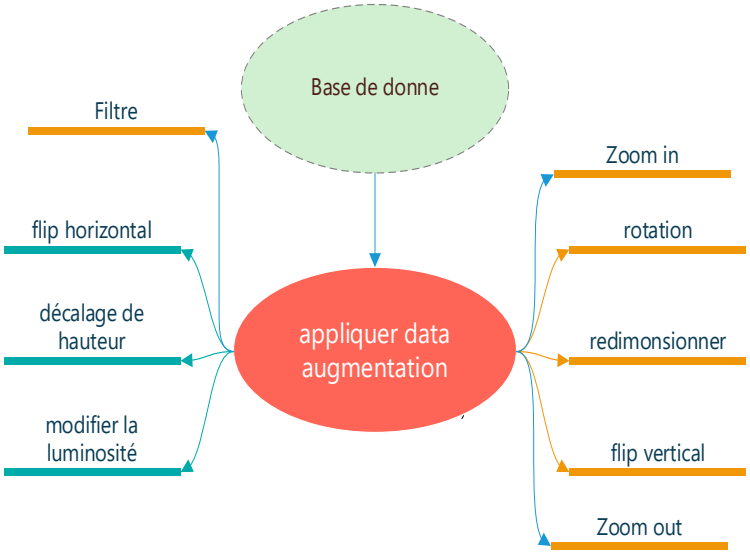


Figure 2.4 : *Augmentation des Données*

II.4. U-Net

L'architecture U-Net, mise au point en 2015 par Olaf Ronneberger, Philip Fischer et Thomas Brox à l'Université de Fribourg, représente une avancée majeure pour la segmentation sémantique des images, en particulier dans le domaine biomédical. Cette structure en forme de "U" comprend une voie de sous-échantillonnage pour capturer le contexte et une voie de sur-échantillonnage qui permet une localisation précise, optimisant ainsi les performances des réseaux de neurones convolutifs pour les tâches de segmentation avec peu d'exemples de formation. U-Net reste une référence dans le domaine, combinant efficacité et précision, même avec des jeux de données limités. [24]

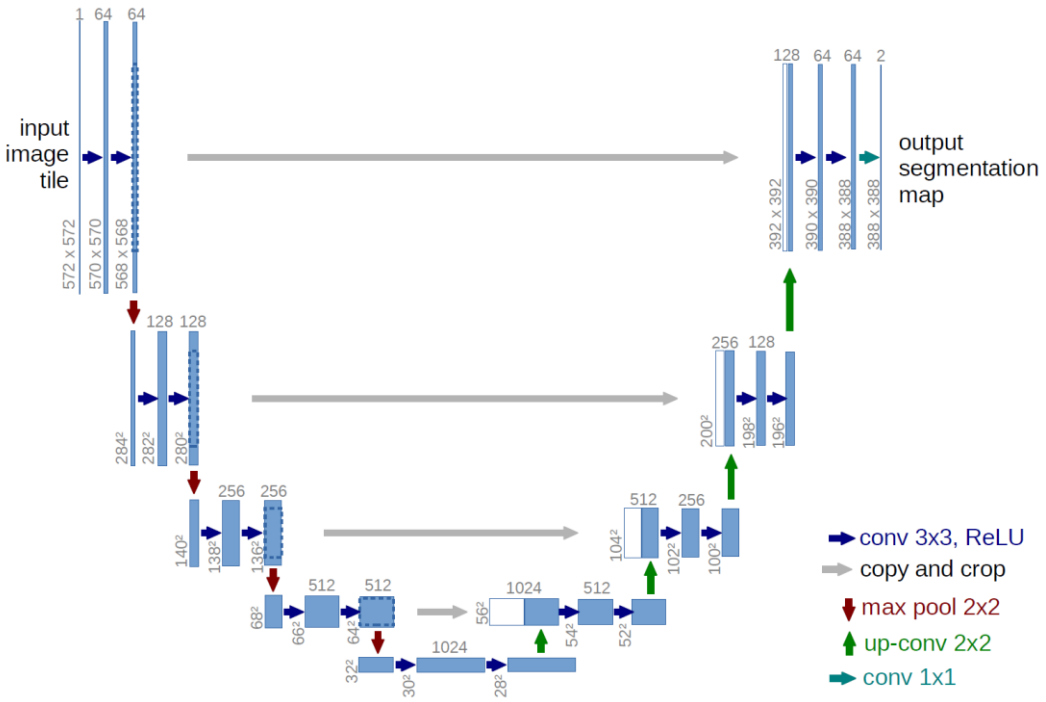


Figure 2.5 : *architecture U-Net*

II.4.1. Architecture U-net

II.4.1.1. Chemin de contraction (encodeur)

L'entrée dans U-Net est une image avec une résolution spatiale et un nombre de canaux spécifiques (par exemple, une image en niveaux de gris a un seul canal, tandis qu'une image couleur a trois canaux, rouge, vert et bleu). Les blocs successifs du chemin de retrait sont constitués de deux couches convolutives, suivies d'une couche d'activation ReLU et d'une opération de pooling maximum. Les motifs locaux de l'image sont capturés par de petits filtres (généralement 3x3) utilisés dans chaque couche convolutive.

La résolution spatiale de l'image diminue à chaque étape de rétrécissement (en raison du regroupement maximal), tandis que la profondeur de la carte des caractéristiques augmente (en raison de l'ajout de filtres). Cela facilite la capture.

II.4.1.2. Chemin d'expansion (décodeur)

L'expansion du chemin inverse la contraction. Cela commence par une opération de convolution supérieure (ou convolution de-top), qui permet d'améliorer la résolution spatiale des aspects. Après chaque convolution ascendante, une opération de « copie et recadrage » est effectuée, où la carte caractéristique correspondante du chemin de contraction est copiée et fusionnée avec la sortie actuelle. Cela facilite la récupération des données spatiales perdues lors du retrait. Les couches convolutives sont utilisées pour affiner les caractéristiques et restaurer l'image à sa résolution d'origine lors de l'expansion.

II.5. Architecture U-net pour la Segmentation

U-Net produit un masque qui segmente précisément l'arbre vasculaire rétinien.

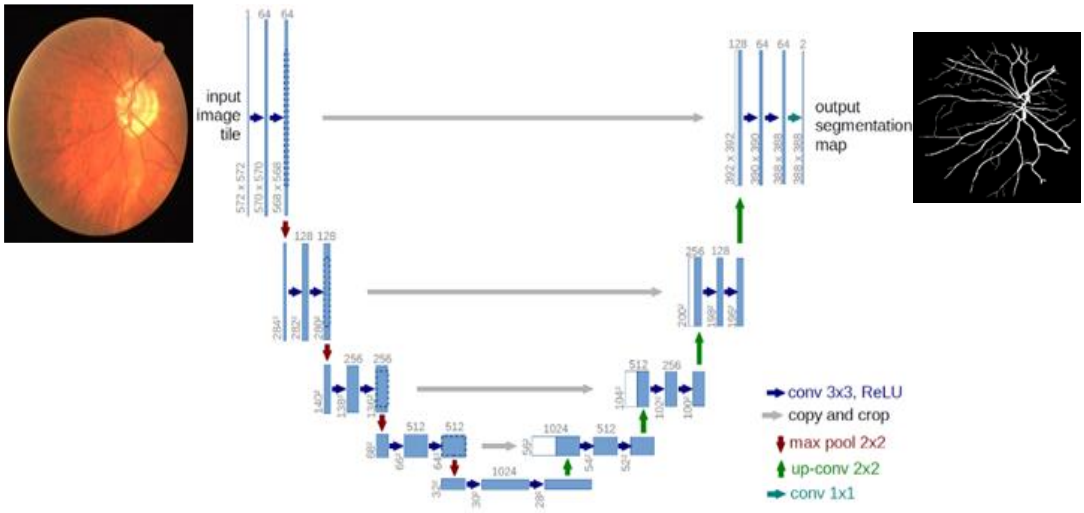


Figure 2.6 : *Architecture U-net pour la Segmentation*

II.6. Structures modifiées de U net

À première vue, notre U-Net a une forme en « U ». L'architecture est symétrique et se compose de deux parties principales : la partie gauche, appelée chemin de contraction (ou encodeur), qui est constituée par le processus général de convolution ; et la partie droite, le chemin expansif (ou décodeur), qui est constitué de couches convolutives 2D transposées (que nous pouvons considérer comme une technique de suréchantillonnage pour l'instant).

Chaque processus constitue deux couches convolutives. Dans le premier processus, les deux couches de convolution sont composées de 64 filtres de taille 3x3. Chacune de nos couches de convolution est suivie d'une fonction d'activation ReLU et d'une foulée de 1. En raison du problème de rembourrage, nous utilisons le padding = « same ». Ces couches sont séparées par une couche de batchnormalisation. Nous avons ajouté la normalisation par lot pour garantir des taux d'apprentissage plus rapides et plus stables, car la normalisation assure qu'il n'y a pas de valeurs d'activation trop élevées ou trop faibles, tout en permettant à chaque couche d'apprendre indépendamment des autres. Ensuite, nous appliquons une couche de max pooling de dimension 2x2 et de foulée de 2.

Nous répétons la même chose avec les autres processus, en augmentant progressivement le nombre de filtres : 128 filtres, 256 filtres, et 512 filtres.

Ensuite, nous arrivons à la couche de bottleneck avec 1024 filtres, suivie de deux convolutions 3x3, des activations ReLU, et des couches de batchnormalisation.

Le chemin expansif commence par une couche de convolution transposée avec 512 filtres, suivie d'une concaténation avec la sortie du bloc correspondant du chemin de contraction. Nous appliquons ensuite deux convolutions 3x3, des activations ReLU, et des couches de batchnormalisation. Ce processus est répété en diminuant le nombre de filtres : 256 filtres, 128 filtres, et 64 filtres.

La convolution transposée est une technique de suréchantillonnage qui élargit la taille des images. Fondamentalement, elle applique un peu de rembourrage sur l'image d'origine suivie d'une opération de convolution. Après la convolution transposée, l'image est redimensionnée, puis cette image est concaténée avec l'image correspondante du chemin de contraction pour créer une image de taille correspondante. La raison de cette concaténation est de combiner les informations des couches précédentes afin d'obtenir une prédiction plus précise.

En fin de compte, notre dernière couche est une couche de convolution avec 1 filtre de taille 1x1. Cette couche contient également une fonction d'activation sigmoïde et une foulée de 1.

Voici une vue d'ensemble détaillée de notre modèle :

1. **Entrée** :
    - inputs = Input(input_shape)
    - Définition de la couche d'entrée avec une forme donnée (SIZE, SIZE, 1).
2. **Chemin de Contraction (Encodeur)** :
    - **Bloc 1** : 64 filtres, convolutions 3x3, ReLU, BatchNormalization, MaxPooling 2x2.
    - **Bloc 2** : 128 filtres, convolutions 3x3, ReLU, BatchNormalization, MaxPooling 2x2.
    - **Bloc 3** : 256 filtres, convolutions 3x3, ReLU, BatchNormalization, MaxPooling 2x2.
    - **Bloc 4** : 512 filtres, convolutions 3x3, ReLU, BatchNormalization, MaxPooling 2x2.
3. **Couche de Bottleneck** :
    - **Bloc 5** : 1024 filtres, convolutions 3x3, ReLU, BatchNormalization.
4. **Chemin Expansif (Décodeur)** :
    - **Bloc 6** : Convolution transposée à 512 filtres, concaténation avec la sortie du Bloc 4, convolutions 3x3, ReLU, BatchNormalization.
    - **Bloc 7** : Convolution transposée à 256 filtres, concaténation avec la sortie du Bloc 3, convolutions 3x3, ReLU, BatchNormalization.
    - **Bloc 8** : Convolution transposée à 128 filtres, concaténation avec la sortie du Bloc 2, convolutions 3x3, ReLU, BatchNormalization.
    - **Bloc 9** : Convolution transposée à 64 filtres, concaténation avec la sortie du Bloc 1, convolutions 3x3, ReLU, BatchNormalization.
5. **Sortie** :
    - **Bloc 10** : Convolution 1x1 avec 1 filtre, fonction d'activation sigmoïde.

Voici le **Tableau 2.1** détaillant les caractéristiques de chaque couche de notre U-Net pour la segmentation de l'arbre rétinien :

Tableau 2.1 : une partie d'architecture de l'U-Net avec BatchNormalization

Couche	Filtres/ Neurones	Dimension du Filtre	Foulée	Rembourrage	Taille de Carte des Caractéristiques	BatchNormalization	Fonction d'Activation
Entrée	-	(SIZE, SIZE, 1)	-	-	(SIZE, SIZE, 1)	Non applicable	-
Conv2D	64	3x3	1	'same'	(SIZE, SIZE, 64)	Oui	ReLU
BatchNormalization	-	-	-	-	(SIZE, SIZE, 64)	-	-
Conv2D	64	3x3	1	'same'	(SIZE, SIZE, 64)	Oui	ReLU
BatchNormalization	-	-	-	-	(SIZE, SIZE, 64)	-	-
MaxPooling2D	-	2x2	2	-	(SIZE/2, SIZE/2, 64)	Non applicable	-
Conv2D	128	3x3	1	'same'	(SIZE/2, SIZE/2, 128)	Oui	ReLU
BatchNormalization	-	-	-	-	(SIZE/2, SIZE/2, 128)	-	-
Conv2D	128	3x3	1	'same'	(SIZE/2, SIZE/2, 128)	Oui	ReLU
BatchNormalization	-	-	-	-	(SIZE/2, SIZE/2, 128)	-	-
MaxPooling2D	-	2x2	2	-	(SIZE/4, SIZE/4, 128)	Non applicable	-

II.7. Les techniques d'apprentissage

Les techniques d'apprentissage automatique couramment utilisées dans la conception d'applications de réseaux neuronaux comprennent Méthodes d'apprentissage : segmentation, classification, régression, reconnaissance de formes et clustering.

La plupart des spécialistes en machine Learning font appel à un apprentissage supervisé. Les variables d'entrée (x) et de sortie (Y) sont utilisées dans l'apprentissage supervisé. La fonction de mappage de l'entrée à la sortie Y = f (X) est étudiée à l'aide d'un algorithme. L'objectif est d'avoir une compréhension approfondie de la fonction de mappage afin que lorsque vous avez de nouvelles données d'entrée (x), vous puissiez anticiper les variables de sortie (Y) correspondantes.

Il s'agit de l'apprentissage supervisé, car le processus d'un algorithme en utilisant l'ensemble de données peut être perçu comme un enseignant qui supervise le processus d'apprentissage. Les réponses correctes sont bien connues, l'algorithme réalise des prédictions itératives sur les données d'apprentissage et est suivi de corrections par l'enseignant. [25]

II.8. Les paramètres d'entraînement

II.8.1. Optimisation des fonctions

L'amélioration des fonctionnalités est un élément essentiel de l'apprentissage en profondeur. La majorité des algorithmes de Deep Learning nécessitent l'amélioration des paramètres (poids, coefficients, etc.) en fonction des données d'apprentissage. L'optimisation définit également le processus de recherche du meilleur ensemble d'hyper paramètres qui définissent l'entraînement d'un algorithme de Deep Learning.

II.8.1.1. Taux d'apprentissage (Learning Rate)

Le taux d'apprentissage peut jouer un rôle essentiel dans la configuration des réseaux neuronaux. Chaque fois que les poids du modèle sont mis à jour, il surveille le niveau de changement du modèle en fonction de l'erreur suggérée. [26]

II.8.1.2. Optimisation Adam

Le terme Adam provient de l'estimation adaptative du moment. Cette méthode permet d'améliorer le gradient de premier ordre des fonctions objectives en utilisant des estimations adaptatives des moments d'ordre inférieur. [28]

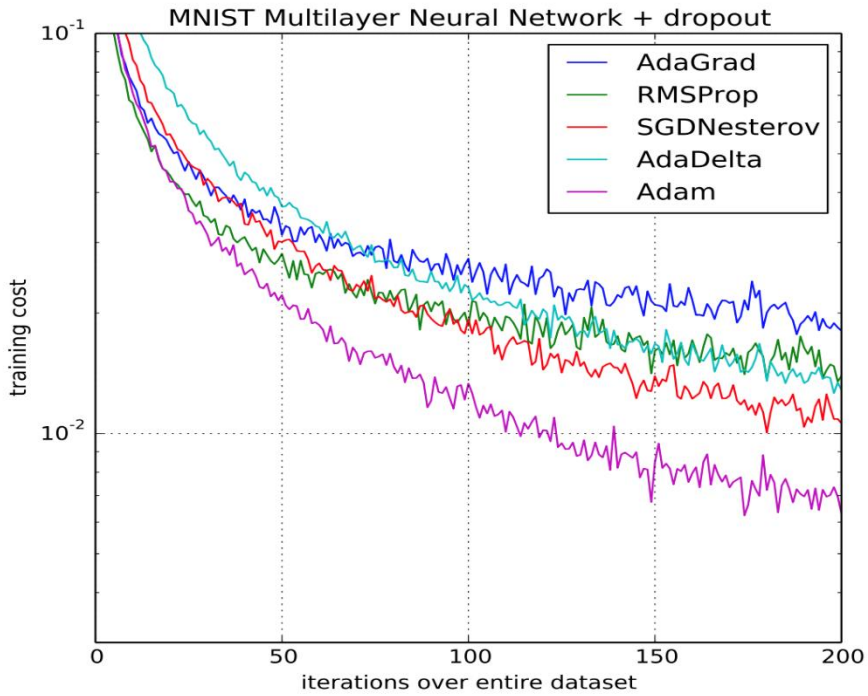


Figure 2.7 : *Optimisation Adam*

II.8.1.3. Epoch

Epoch signifie utiliser toutes les données d'entraînement en une seule époque pour entraîner le réseau neuronal. Epoch utilise toutes les données une seule fois. Les passes avant et les passes arrière sont combinées en une seule passe. [27]

II.8.1.4. Batch size

La taille du lot correspond au nombre de valeurs de données d'entrée que vous saisissez dans le formulaire à la fois. Ceci est très important pendant la formation et moins important pendant les tests.

II.8.1.5. La fonction du perte(loss)

La fonction de perte est une fonction qui évalue la différence entre les prédictions d'un réseau de neurones et les valeurs réelles des observations utilisées lors de l'entraînement. Plus le résultat de cette fonction est petit, meilleur est le fonctionnement du réseau neuronal. [29]

II.8.1.6. Précision (Accuracy)

En général, la précision est une mesure utilisée pour évaluer les performances du modèle dans toutes les catégories.

On calcule la précision en comparant le nombre d'échantillons positifs correctement

classés avec le nombre total d'échantillons classés comme positifs (correctement ou incorrectement). La précision évalue la précision du modèle dans la classification d'un échantillon comme positif.

La précision constitue une métrique utilisée pour évaluer les modèles de classification. Dans une perspective informelle, la précision correspond à la proportion des prédictions que notre modèle a été exacte. [30]

II.8.1.7. Input Shape

Il s'agit d'une image avec les deux premières valeurs représentant la largeur et la hauteur de l'image, et la troisième représentant les canaux de l'image.

II.8.1.8. Padding

Lorsque cette valeur est définie sur "identique", les éléments de bordure de la forme ne seront pas jetés.

II.8.1.9. Shuffle

Après avoir défini cette valeur sur True, l'ensemble des données sera mélangé avant l'entraînement.

II.8.1.10. Step per epoch

Le nombre d'itérations du lot avant la fin d'une période d'apprentissage est calculé en multipliant le nombre d'échantillons formant un ensemble par la taille du lot.

II.9. Les fonctions d'activation

II.9.1. La fonction ReLu (Rectified Linear Unit)

La fonction d'activation non saturante, également connue sous le nom de "fonction d'activation non saturante", accroît les caractéristiques non linéaires de la fonction de décision et de l'ensemble du réseau sans impacter les champs reçus de la couche. Après la convolution, on applique la fonction d'activation qui maintient la taille et le volume présents, ce qui constitue une couche de correction. [31]

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

La ReLU à plusieurs propriétés intéressantes :

- **Biologiquement plausible :** comparé unilatéral à laantisymétrique tanh.
- **Activation creuses :** 50% des neurones sur une activation nulle après

l'initialisation aléatoire des poids.

- **Pas de vanishing gradient :** La dérivée vaut 1 partout dans la portion positive.
- **Complexité faible :** comparaison.

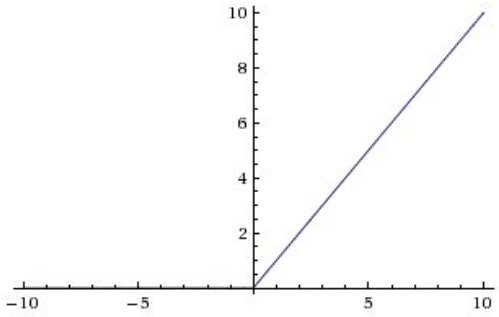


Figure 2.8 : *La fonction Relu*

II.9.2. La fonction sigmoïde

Elle exprime une valeur allant de 0 à 1. Il s'agit d'une imitation efficace d'un neurone qui peut être activé ou désactivé car il renvoie généralement des valeurs proches de 0 ou 1. Le principal souci réside dans la disparition du gradient sur les files d'attente. De cette manière, l'algorithme de rétropropagation ne change que très peu les paramètres et l'apprentissage est accéléré.

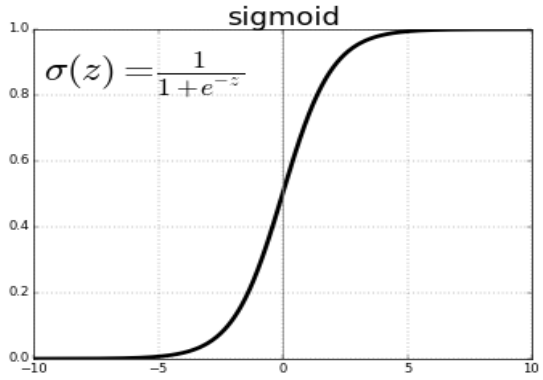


Figure 2.9 : *Fonction d'activation Sigmoïde*

II.9.3. Fonction de Perte (Loss)

Pour évaluer la disparité entre les prédictions du modèle et les masques réels, nous avons utilisé la fonction de perte binaire_crossentropy. Cette fonction mesure la divergence entre les distributions de probabilité des prédictions et des étiquettes.

II.9.4. Optimisation et Hyperparamètres

Nous avons utilisé l'optimiseur Adam pour entraîner notre modèle, avec un taux

d'apprentissage de 0.0001 . Le modèle a été entraîné sur 25 epochs, avec un batch size de 16 . Le choix du batch size est crucial dans le processus d'entraînement, car il détermine le nombre d'échantillons utilisés pour mettre à jour les poids du modèle à chaque itération. Un batch size plus élevé peut accélérer le processus d'entraînement, mais il nécessite également plus de mémoire GPU. D'autre part, un batch size plus petit peut permettre une meilleure généralisation du modèle, mais il peut rendre l'entraînement plus lent et moins stable.

II.9.5. Évaluation du Modèle

Nous avons évalué les performances de notre modèle à l'aide de différentes métriques, notamment la perte d'entraînement et de validation, ainsi que l'accuracy. De plus, nous avons calculé l'indice de similarité Jaccard moyen (Mean IoU) pour évaluer la qualité de la segmentation .

En incluant le paramètre de batch size, nous avons fourni une perspective plus complète sur les hyperparamètres utilisés pour entraîner notre modèle de segmentation des vaisseaux rétiniens.

II.10. Conclusion

Ce chapitre a présenté la conception et l'implémentation d'une architecture de deep learning pour la segmentation des vaisseaux rétiniens, en utilisant le modèle U-Net. Nous avons expliqué les composants essentiels de U-Net, ses chemins de contraction et d'expansion, et l'importance de l'optimisation des hyper paramètres. Le prétraitement des données et les techniques d'augmentation ont été essentiels pour améliorer la généralisation du modèle. L'utilisation du jeu de données DRIVE et les évaluations effectuées ont démontré l'efficacité de notre modèle pour la segmentation précise de l'arbre rétinien

Chapitre 03 :

Segmentation de l'arbre rétinien par U-net

III.1. Introduction

Dans ce chapitre, nous décrivons le processus de développement et d'entraînement du modèle U-Net pour la segmentation de l'arbre rétinien. Nous détaillons l'environnement de travail utilisé, comprenant les langages de programmation, les bibliothèques, et les plateformes de développement. Nous abordons également les étapes d'implémentation du réseau de segmentation, incluant la préparation des données, la construction du modèle, et l'entraînement. Enfin, nous discutons des résultats obtenus, des performances du modèle, et des améliorations potentielles pour optimiser la segmentation de l'arbre rétinien

III.2. Environnement de travail

III.2.1. Le langage de programmation utilisé (Python)

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

III.2.2. Les bibliothèques Python

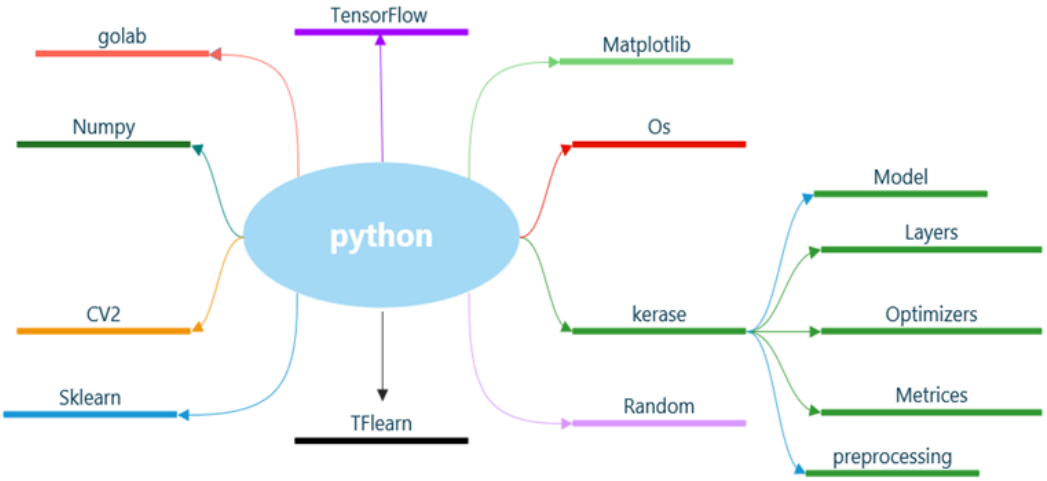


Figure 3.1 : *Les bibliothèques utilisées*

III.2.2.1. TensorFlow

TensorFlow est une bibliothèque de Deep Learning créée par l'équipe Google Brain de Google. Elle offre une palette d'outils permettant de résoudre des problèmes mathématiques complexes et de créer des architectures d'apprentissage expérimental

III.2.2.2. TFlearn

TFlearn est une bibliothèque modulaire d'apprentissage profond construite sur TensorFlow. Elle fournit une API de plus haut niveau, facilitant et accélérant les expérimentations, comme le redimensionnement d'images.

III.2.2.3. Keras

Keras est une API d'apprentissage profond écrite en Python, fonctionnant sur TensorFlow. Elle permet une expérimentation rapide grâce à sa convivialité, sa modularité et sa facilité d'extension. Keras offre des composants autonomes tels que des couches neuronales, des fonctions de coût, des optimiseurs, des fonctions d'activation et des schémas de régularisation, combinables pour créer de nouveaux modèles.

III.2.2.4. Open CV

OpenCV est une bibliothèque gratuite de vision par ordinateur compatible avec plusieurs plateformes, telles que GNU/Linux, macOS, Windows et Android. Initialement développée par Intel, elle est utilisée dans des applications variées, comme les systèmes de sécurité et les applications de contrôle de processus nécessitant la reconnaissance d'objets. Sa licence BSD permet une utilisation libre à des fins commerciales et de recherche.

III.2.2.5. Matplotlib

Matplotlib est une bibliothèque Python open source, créée par John Hunter en 2002 pour visualiser les signaux électriques du cerveau. Elle permet de créer divers types de graphiques avec peu de lignes de code, offrant des visualisations de données détaillées.

III.2.3. Google Colab

Google Colab, ou « Colaboratory », est une plateforme cloud gratuite de Google pour la recherche en Deep Learning et Intelligence Artificielle. Elle permet d'entraîner des modèles de Deep Learning directement dans le cloud sans nécessiter d'installation locale, à l'exception d'un navigateur. Colab offre un environnement de développement (Jupyter Notebook) avec accès gratuit à un GPU et TPU, et les fichiers sont sauvegardés dans Google Drive.

III.2.4. Avantages de Colab

Colab est simple à utiliser et flexible dans sa configuration. Parmi ses avantages :

- Prise en charge de Python 2.7 et 3.11
- Accélération GPU gratuite
- Bibliothèques préinstallées, comme TensorFlow et Scikit-learn
- Prise en charge des commandes Bash

En résumé, Colab facilite grandement le développement en Deep Learning avec ses nombreuses fonctionnalités prêtes à l'emploi.

- **Spécifications du PC :**

Voilà les spécifications de votre PC utilisé pour l'opération de segmentation des vaisseaux rétiniens à l'aide du modèle U-Net dans Google Colab :

- **Processeur :** AMD Ryzen 3100
- **Carte graphique :** NVIDIA GTX 1050
- **RAM :** 8 Go DDR4
- **Stockage :** SSD 500 Go (M.2)

III.3. Implémentation de U-net pour la segmentation :

Nous examinons les différentes phases du travail :

D'abord, nous préparons l'environnement de travail. Ensuite, nous divisons et importons la base de données. Après cela, nous appliquons l'augmentation des données et le transfert de connaissances. Enfin, nous procédons à l'apprentissage.

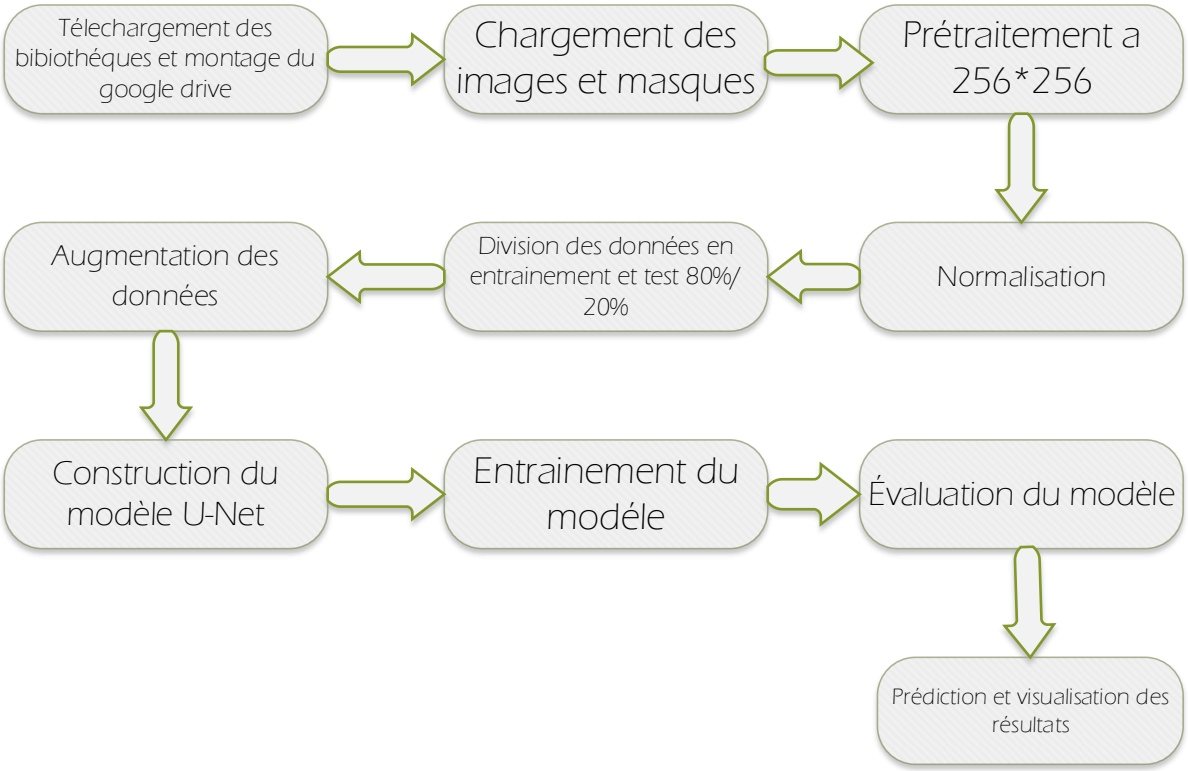


Figure 3.2 : *Les étapes de segmentation des images de rétine*

III.4. Préparer l'environnement

Dans cette partie nous importerons toutes les bibliothèques nécessaires pour le traitement d'images, la construction du modèle U-Net, l'optimisation, la visualisation et la gestion des données.

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate, Conv2DTranspose, BatchNormalization, Activation
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.metrics import MeanIoU
import glob
import random
```

Figure 3.3 : *Code d'importation des bibliothèques et les paramètres d'entrainement*

III.4.1. Importation de bibliothèque

Dans cette section, nous allons importer toutes les bibliothèques nécessaires ainsi que les outils pour générer la matrice de confusion et le rapport de segmentation.

Versions des bibliothèques

Pour assurer la reproductibilité et la cohérence de nos résultats, nous avons utilisé les versions suivantes des bibliothèques Python clés dans notre environnement de développement :

- google-colab : 1.0.0
- graphviz : 0.20.3
- keras : 2.15.0
- matplotlib : 3.7.1
- numpy : 1.25.2
- opencv-python : 4.8.0.76
- pydot : 1.4.2
- scikit-learn : 1.2.2
- tensorflow : 2.15.0

```python
# Importer les bibliothèques nécessaires
import pkg_resources
import sys

# Définir une liste des bibliothèques que vous avez installées explicitement
packages_to_check = [
    'pydot',
    'graphviz',
    'tensorflow',
    'google-colab',
    'opencv-python',
    'numpy',
    'matplotlib',
    'scikit-learn',
    'keras',
]

# Afficher la version de Python
print(f"Python version: {sys.version}\n")

# Récupérer toutes les bibliothèques installées avec leurs versions
installed_packages = pkg_resources.working_set

# Afficher chaque bibliothèque et sa version si elle est dans la liste des bibliothèques à vérifier
for package in installed_packages:
    if package.key in packages_to_check:
        print(f"{package.key}=={package.version}")
```

Figure 3.4 : *code pour voir versions des bibliothèques*

```
Python version: 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]

google-colab==1.0.0
graphviz==0.20.3
keras==2.15.0
matplotlib==3.7.1
numpy==1.25.2
opencv-python==4.8.0.76
pydot==1.4.2
scikit-learn==1.2.2
tensorflow==2.15.0
```

Figure 3.5 : *les versions des bibliothèques importées.*

III.5. Redimensionnement et visualisation des images

III.5.1. Définition de la patchification

La "patchification" (ou patchify en anglais) est une technique de traitement des données souvent utilisée dans le domaine de la vision par ordinateur et de l'apprentissage automatique pour optimiser l'entraînement des modèles de segmentation. Elle consiste à découper une grande image en plusieurs petites régions rectangulaires appelées "patchs". Chaque patch est généralement extrait avec une taille fixe, par exemple 256x256 pixels, à partir de l'image d'origine par exemple 512 x 512. Cette approche permet de traiter chaque région de manière individuelle et de faciliter l'analyse locale des caractéristiques, améliorant ainsi la gestion des données et l'efficacité des algorithmes de segmentation.

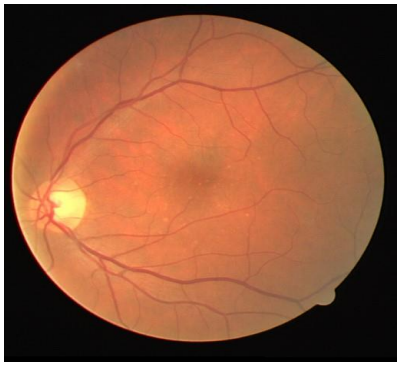
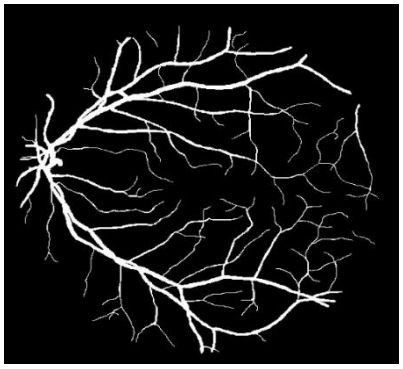
Imaginez une grande image représentant une partie de la rétine d'un œil, par exemple une image de 512x512 pixels. Pour analyser cette image et détecter les structures importantes comme l'arbre rétinien, la patchification est utilisée. Cela signifie diviser l'image en morceaux plus petits, comme découper un puzzle géant en petites pièces.

1. **Découpage en patchs** :
    - Vous divisez l'image de 512x512 pixels en morceaux plus gérables de 256x256 pixels chacun. Ainsi, une seule grande image peut être divisée en quatre patchs (ou morceaux) de 256x256 pixels.
2. **Pourquoi faire cela ?** :
    - La patchification permet d'analyser chaque partie de l'image individuellement. C'est comme regarder de plus près différentes sections de l'image pour identifier et étudier les détails importants, comme la structure de l'arbre rétinien, sans être distrait par le reste de l'image.

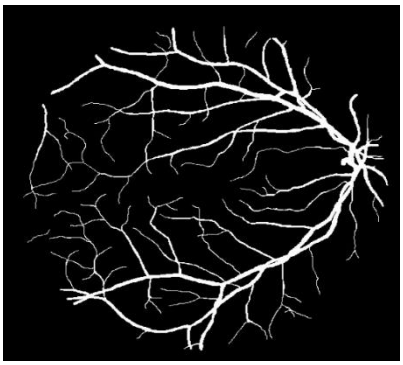
3. **Amélioration de la précision** :
    - En analysant chaque patch séparément, les algorithmes peuvent se concentrer précisément sur les caractéristiques locales de l'arbre rétinien. Cela améliore la précision des résultats lors de la détection et de la segmentation de cette structure anatomique complexe.
4. **Gestion des données** :
    - Gérer de petites images (patchs) est souvent plus efficace que de manipuler une seule grande image, surtout lorsqu'il s'agit de traitement informatique intensif. La patchification permet une gestion optimale des données et améliore les performances des algorithmes.
5. **Application dans la recherche médicale** :
    - Dans des études comme celles utilisant le jeu de données DRIVE, qui contient des images de rétines avec des structures comme l'arbre rétinien, la patchification est cruciale. Elle facilite l'analyse détaillée et la segmentation précise de ces structures anatomiques clés.

Avant l'application de la patchification, nous travaillions avec des images rétiniennes complètes de taille 512x512 pixels et leurs masques correspondants. Chaque image et son masque étaient traités comme une entité unique, ce qui pouvait rendre la détection précise de structures comme l'arbre rétinien plus complexe et moins efficace. Voici un exemple d'une image rétinienne de taille 512x512 pixels avec son masque associé :

Des exemples :

Image	**Mask**
	
a	b

c



d

Figure 3.6 : *l'image rétinienne complète de 512x512 pixels avec son masque*

Appliquer la patchification :

Donc Nous avons programmé toutes ces opérations avec Python sur Spyder d'Anaconda. Cette méthode a été utilisée pour éviter la saturation de la RAM lors de l'entraînement dans Google Colab.

Dans cette section, nous avons redimensionné les 80 images à une taille de 512x512 pixels. Ensuite, nous avons divisé chaque image en 4 parties en utilisant la technique de patchify(la même méthode a été appliquée aux masques). Nous avons ensuite affiché quelques images ainsi que leurs masques correspondants.

Name	Type	Size	Value
filename	str	5	9.png
i	int	1	1
img	Array of uint8	(512, 512, 3)	[[[0 0 0] [0 0 0]
input_dir	str	64	C:/Users/AURES/Downloads/dataset216/small data for training/mask
j	int	1	1
mask	Array of uint8	(512, 512)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
output_dir	str	14	patches/masks/
patches_img	Array of uint8	(2, 2, 1, 256, 256, 3)	[[[[[[0 0 0] [0 0 0]
patches_mask	Array of uint8	(2, 2, 256, 256)	[[[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
single_patch_img	Array of uint8	(256, 256, 3)	[[[60 124 255] [62 126 255]
single_patch_mask	Array of uint8	(256, 256)	[[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]

Figure 3.7 : *Patchification des images et masques et les démontions sur Sypder*

Après avoir appliqué la technique de patchification, chaque image rétinienne de 512x512 pixels a été découpée en quatre patchs de 256x256 pixels. De même, les masques

correspondants ont été patchifiés pour correspondre aux nouvelles dimensions des images. Voici comment l'image rétinienne et son masque ont été transformés après la patchification:

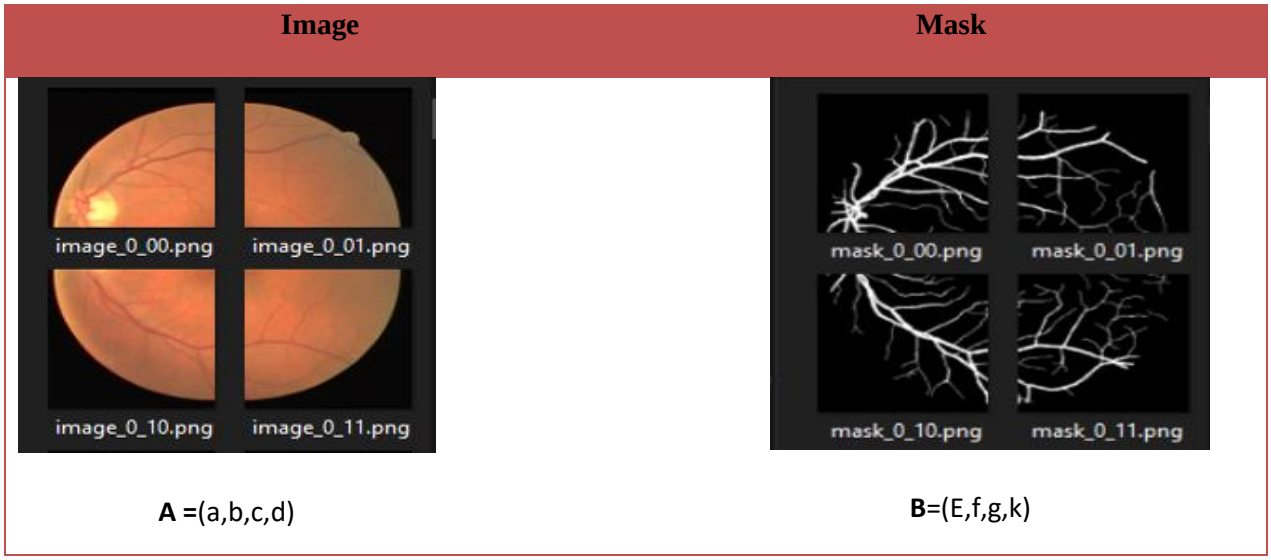


Figure 3.8 : *les patchs résultants de l'image rétinienne et du masque correspondant découpés en 256x256 pixels*

Cette approche nous permet maintenant d'analyser chaque partie de l'image rétinienne individuellement, améliorant ainsi notre capacité à détecter et segmenter des structures spécifiques telles que l'arbre rétinien avec une précision accrue. Les patchs résultants facilitent également la gestion des données et optimisent les performances de nos algorithmes d'analyse d'images.

III.6. Chargement des données

III.6.1. Définir les répertoires des images et des masques

Nous spécifions les répertoires où sont stockées les images et les masques correspondants sur Google Drive.

```
[ ]  # Load data
     image_directory = '/content/drive/MyDrive/patches/images'
     mask_directory = '/content/drive/MyDrive/patches/masks'
```

Figure 3.9 : *les répertoires des images et des masques*

III.6.2. Récupérer les noms des fichiers d'images et de masques

Nous utilisons la bibliothèque "glob" pour récupérer tous les fichiers .png dans les répertoires spécifiés.

```
[ ]  image_names = glob.glob(os.path.join(image_directory, '*.png'))
     mask_names = glob.glob(os.path.join(mask_directory, '*.png'))
```

Figure 3.10 : *Code de Récupérer les noms des fichiers d'images et de masques*

III.6.3. Trier les noms des fichiers

Nous trions les noms des fichiers pour nous assurer que les images et les masques correspondent correctement par ordre.

```
image_names.sort()
mask_names.sort()
```

Figure 3.11 : *Trier les noms des fichiers*

Dans cette partie, les données sont préparées pour l'entraînement du modèle. Les images et leurs masques correspondants sont chargés à partir des répertoires spécifiés dans Google Drive.

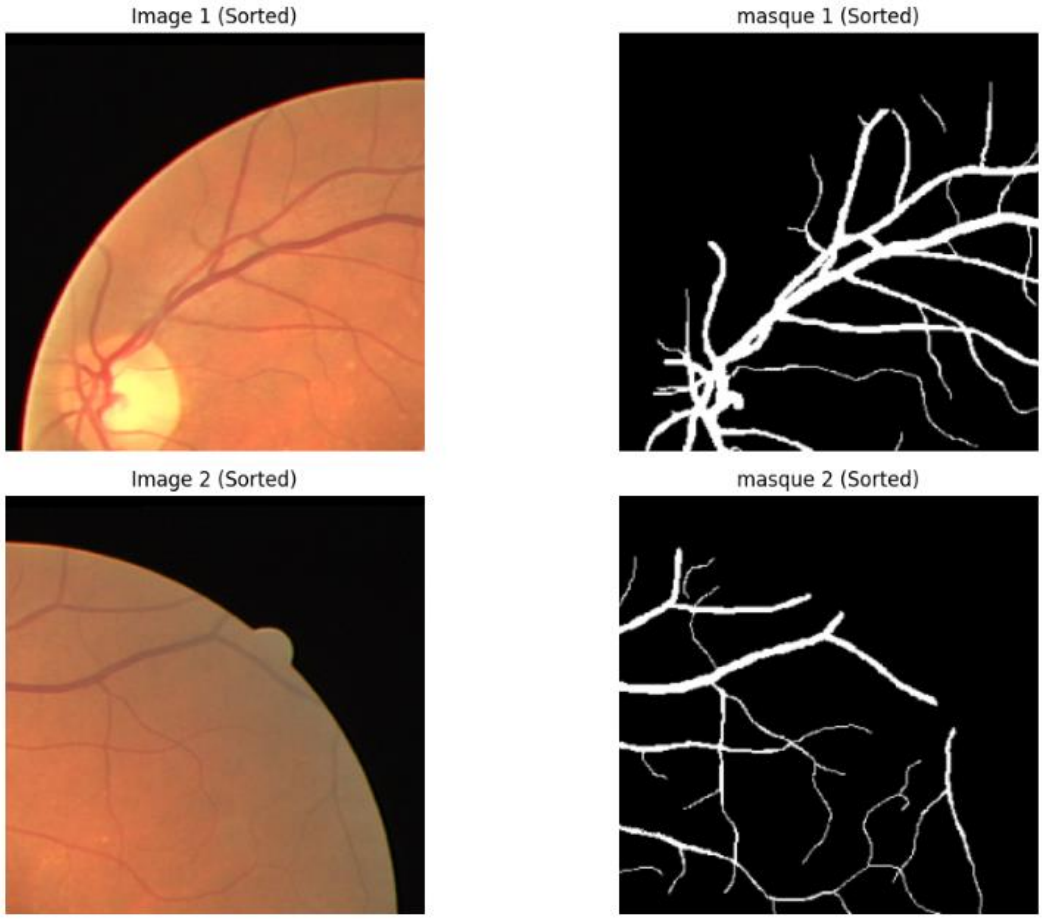


Figure 3.12 : *la vérification des images et leur masque correspondant*

```
Image Names (first 10):
/content/drive/MyDrive/patches/images/image_0_00.png     /content/drive/MyDrive/patches/images/image_0_01.png
/content/drive/MyDrive/patches/images/image_0_10.png     /content/drive/MyDrive/patches/images/image_0_11.png
/content/drive/MyDrive/patches/images/image_10_00.png    /content/drive/MyDrive/patches/images/image_10_01.png
/content/drive/MyDrive/patches/images/image_10_10.png    /content/drive/MyDrive/patches/images/image_10_11.png
/content/drive/MyDrive/patches/images/image_11_00.png    /content/drive/MyDrive/patches/images/image_11_01.png

Mask Names (first 10):
/content/drive/MyDrive/patches/masks/mask_0_00.png       /content/drive/MyDrive/patches/masks/mask_0_01.png
/content/drive/MyDrive/patches/masks/mask_0_10.png       /content/drive/MyDrive/patches/masks/mask_0_11.png
/content/drive/MyDrive/patches/masks/mask_10_00.png      /content/drive/MyDrive/patches/masks/mask_10_01.png
/content/drive/MyDrive/patches/masks/mask_10_10.png      /content/drive/MyDrive/patches/masks/mask_10_11.png
/content/drive/MyDrive/patches/masks/mask_11_00.png      /content/drive/MyDrive/patches/masks/mask_11_01.png
```

III.7. Préparation des données

III.7.1. Sélectionner un sous-ensemble des images et des masques

Nous sélectionnons un sous-ensemble des fichiers d'images et de masques pour l'entraînement et le test.

```
SIZE = 256
num_images = 320

image_names_subset = image_names[:num_images]
mask_names_subset = mask_names[:num_images]
```

Figure 3.13 : *Code pour Sélectionner un sous-ensemble des images et des masques*

III.7.2. Chargement des images et des masques en utilisant OpenCV

Nous chargeons les images et les masques en niveaux de gris.

```
images = [cv2.imread(img, 0) for img in image_names_subset]
masks = [cv2.imread(mask, 0) for mask in mask_names_subset]
```

Figure 3.14 : *Code Chargement des images et des masques*

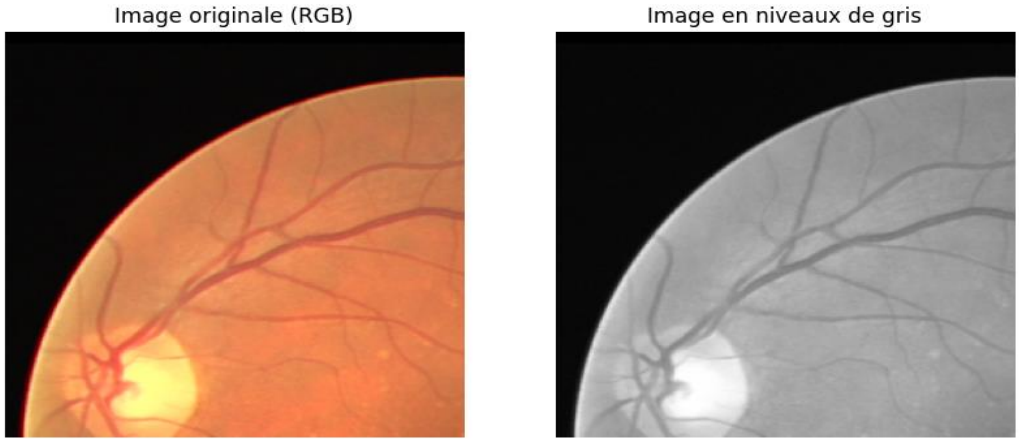


Figure 3.15 : *Image et masque en niveaux de gris*

III.7.3. Conversion des listes d'images et de masques en tableaux numpy

Nous convertissons les listes en tableaux numpy et ajoutons une dimension supplémentaire pour les canaux d'image.

```
image_dataset = np.array(images)
image_dataset = np.expand_dims(image_dataset, axis=3)

mask_dataset = np.array(masks)
mask_dataset = np.expand_dims(mask_dataset, axis=3)
```

Figure 3.16 : *code de Conversion des listes d'images et de masques en tableaux numpy*

III.7.3.1. Le tableau de numpy obtenue

```
Dimensions de image_dataset avant : (256, 256)
Dimensions de mask_dataset avant : (256, 256)
Dimensions de image_dataset après : (1, 256, 256, 1)
Dimensions de mask_dataset après : (1, 256, 256, 1)
```

Figure 3.17 : *résultat de tableaux de numpy*

III.7.4. Normalisation des données

Nous normalisons les valeurs des pixels pour être comprises entre 0 et 1 afin de faciliter l'entraînement du modèle.

```
# Normalize data
image_dataset = image_dataset / 255.
mask_dataset = mask_dataset / 255.
```

Figure 3.18 : *code pour Normalisation des données*

Cette section sélectionne un sous-ensemble des images et masques, les lit en utilisant OpenCV, et les normalise pour avoir des valeurs entre 0 et 1.

III.7.4.1. Normalisation et visualisation des images et leurs matrices avant et après

Voici comment cela peut être fait, avec une visualisation des images et leurs matrices avant et après normalisation. La même chose pour le masque.

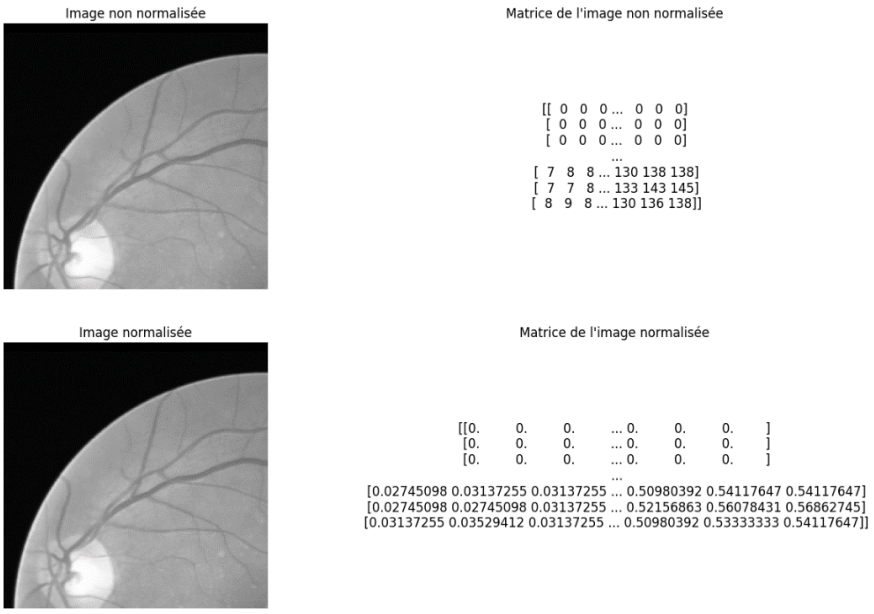


Figure 3.19 : *Normalisation et visualisation des images est leurs matrices avant et après*

Les images et les masques avant normalisation auront des valeurs de pixels allant de 0 à 255, tandis que ceux après normalisation auront des valeurs de pixels allant de 0 à 1. Cette transformation aide à stabiliser et à accélérer le processus d'entraînement du modèle.

III.8. Séparation des données en ensembles d'entraînement et de test

Nous séparons les données en ensembles d'entraînement et de test en utilisant un ratio de 80%/20%, ce qui permet de valider les performances du modèle sur des données non vues pendant l'entraînement.

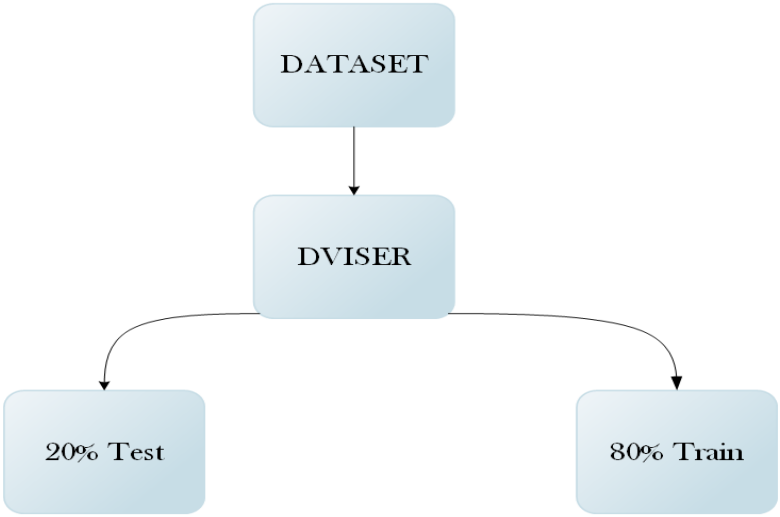


Figure 3.20 : *Schéma synoptique du principe de la technique Séparation des données en ensembles d'entraînement et de test*

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(image_dataset, mask_dataset, test_size=0.2, random_state=42)
```

Figure 3.21: *code Séparation des données en ensembles d'entraînement et de test*

III.9. Visualisation de la base de données

III.9.1. Augmentation des données

Dans le domaine du Deep Learning, il est bien établi qu'une base de données plus grande améliore la qualité et la précision de l'apprentissage. Pour obtenir des résultats optimaux, nous avons donc élargi notre base de données. Nous avons également appliqué diverses techniques d'augmentation des données précédemment étudiées, telles que :

- La rotation
- Le redimensionnement
- Le flip vertical
- Le flip horizontal
- Le décalage de hauteur
- La modification de la luminosité

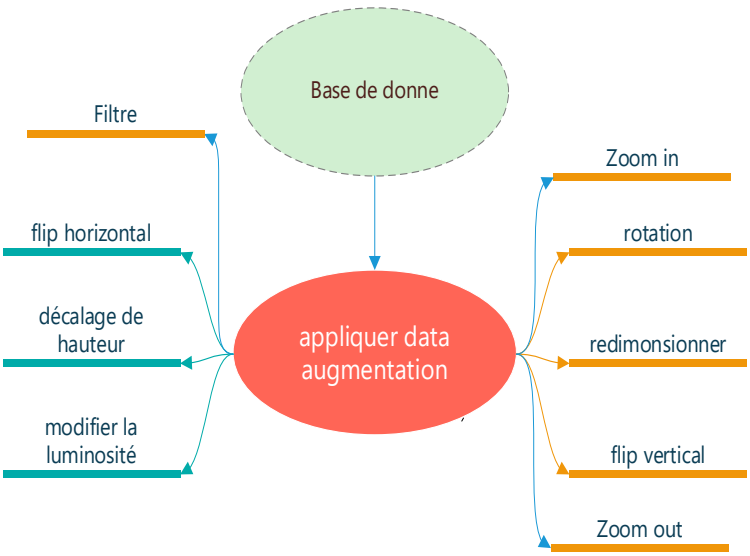


Figure 3.22 : *Schéma synoptique du principe de la technique d'augmentation des données*

L'augmentation des données est essentielle pour améliorer la robustesse du modèle. Cette section utilise "ImageDataGenerator" pour générer des images augmentées.

```
# Data augmentation
datagen = ImageDataGenerator(**datagen_args)

# Define augmentation parameters
datagen_args = dict(
    rotation_range=40,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.2,
    zoom_range=0.3,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    brightness_range=[0.8, 1.2],
    channel_shift_range=20,
)
```

Figure 3.23 : *code pour les paramètres d'augmentation utilisés*

Nous présentons dans le **Tableau 3.1** récapitulatif des paramètres d'augmentation utilisés :

Tableau 3.1 : augmentation des données

Paramétré	Valeur	Description
Rotation range	40	Rotation des images jusqu'à 40 degrés
Width_shit_range	0.3	Décalage horizontal des images jusqu'à 30% de leur largeur
Height_shift_range	0.3	Décalage horizontal des images jusqu'à 30% de leur hauteur
Shear_range	0.2	Cisaillement des images jusqu'à 20%
Zoom range	0.3	Zoom sur les images jusqu'à 30%
Horizontal flip	Ture	Flip horizontal des images
Vertical flip	Ture	Flip vertical des images
Fill_mode	'Nearest'	Remplissage des pixels manquants par la valeur la plus proche
brightness_range	[0.8, 1.2]	Intervalle de modification de la luminosité
channel shift range	20	Amplitude du décalage des canaux de couleur

En utilisant ces paramètres, on a pu générer une variété d'images augmentées qui ont enrichi notre base de données, ce qui a conduit à une meilleure performance et une plus

grande robustesse de notre modèle.

– Appliquer l'augmentation de données pour toute la base de données

```
# Data augmentation
num_batches = 200
X_train_augmented_list = [X_train]
y_train_augmented_list = [y_train]
for _ in range(num_batches):
    for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=batch_size, seed=42):
        X_train_augmented_list.append(X_batch)
        y_train_augmented_list.append(y_batch)
        break
```

Figure 3.24 : *ImageDataGenerator batch*

Dans cette section, "ImageDataGenerator" est utilisé pour générer des transformations aléatoires des images d'entraînement, augmentant ainsi la diversité des données. Ces transformations incluent des rotations, des translations, des zooms, des inversions horizontales et verticales, et des cisaillements.

Après avoir appliqué ces techniques d'augmentation de données, nous avons considérablement enrichi notre base de données initiale. Nous sommes ainsi parvenus à obtenir un total de **1856 images**. Cette expansion significative de notre ensemble de données nous a permis d'améliorer la capacité de notre modèle à généraliser et à produire des prédictions précises

```
Number of images before data augmentation: 256
Size of X_train before data augmentation: (256, 256, 256, 1)
Size of y_train before data augmentation: (256, 256, 256, 1)
Number of augmentation batches: 200
Size of X_train_augmented after data augmentation: (1856, 256, 256, 1)
Size of y_train_augmented after data augmentation: (1856, 256, 256, 1)
```

Figure 3.25 : *La capacité de notre base de données*

Après avoir appliqué ces techniques d'augmentation de données, nous avons considérablement enrichi notre base de données initiale. Nous sommes ainsi parvenus à obtenir un total de **1856 images**. Cette expansion significative de notre ensemble de données nous a permis d'améliorer la capacité de notre modèle à généraliser et à produire des prédictions précises.

III.10. Affichage des images augmentées

Pour vérifier visuellement les images augmentées, la section suivante affiche quelques exemples.

```
# Plot augmented images and masks
augmented_samples_to_display = 4
plt.figure(figsize=(12, 6))
for i in range(augmented_samples_to_display):
    random_index = random.randint(0, len(X_train_augmented) - 1)
    augmented_img = X_train_augmented[random_index, :, :, 0]
    augmented_mask = y_train_augmented[random_index, :, :, 0]

    # Resize mask to match image dimensions if necessary
    if augmented_img.shape[:2] != augmented_mask.shape[:2]:
        augmented_mask = cv2.resize(augmented_mask, (augmented_img.shape[1], augmented_img.shape[0]))

    plt.subplot(2, augmented_samples_to_display, i + 1)
    plt.imshow(augmented_img, cmap='gray')
    plt.title('Augmented Image')
    plt.axis('off')
    plt.subplot(2, augmented_samples_to_display, i + 1 + augmented_samples_to_display)
    plt.imshow(augmented_mask, cmap='gray')
    plt.title('Corresponding Mask')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

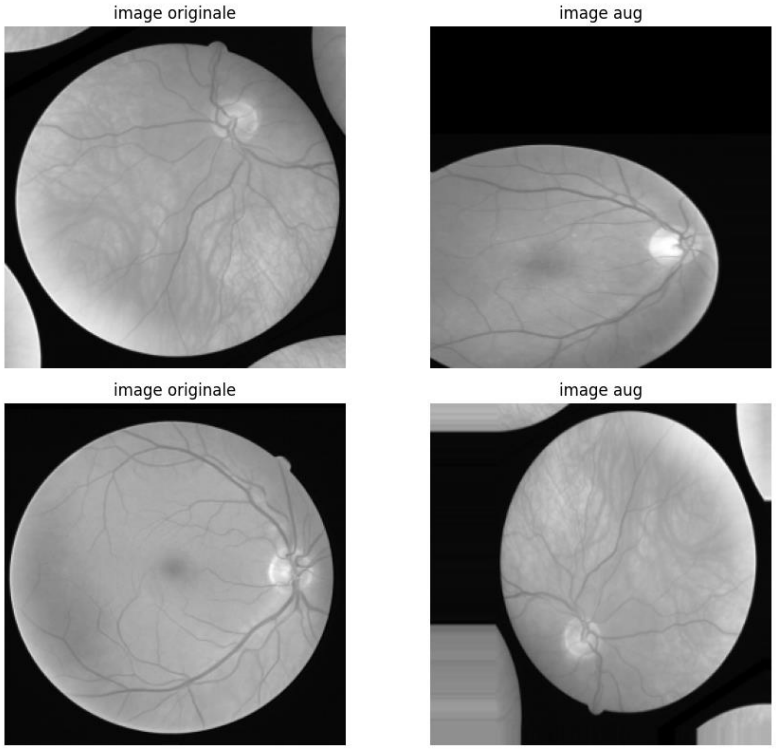


Figure 3.26 : *Code pour Affichage des images augmentées*

Cette section visualise quelques exemples d'images et de masques augmentés pour vérification.

III.10.1. Configuration du modèle

Nous définissons les dimensions d'entrée du modèle en fonction de la taille des images d'entrée.

```
# Model configuration
IMG_HEIGHT = image_dataset.shape[1]
IMG_WIDTH = image_dataset.shape[2]
IMG_CHANNELS = image_dataset.shape[3]
input_shape = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)
```

'IMG_HEIGHT', 'IMG_WIDTH' et I'MG_CHANNELS' sont respectivement la hauteur, la largeur et le nombre de canaux (généralement 1 pour une image en niveaux de gris ou 3 pour une image couleur).

input_shape est un tuple qui spécifie la forme de l'entrée du modèle.

III.11. Compilation du modèle

Après avoir défini l'architecture du modèle, nous le compilons en spécifiant l'optimiseur, la fonction de perte et les métriques à utiliser pendant l'entraînement.

```
# Fine-tune hyperparameters
optimizer = Adam(learning_rate=0.0001)
batch_size = 16
```

Figure 3.27 : *code de configuration de la fonction d'activation et l'optimiseur ADAM*

Model(inputs=[inputs], outputs=[outputs]) : Crée une instance de modèle Keras avec l'entrée inputs et la sortie outputs.

optimizer = Adam(learning_rate=0.0001) : Adam est un optimiseur efficace pour l'entraînement des réseaux de neurones profonds. Nous spécifions un taux d'apprentissage de 0.0001.

model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy']) : Compile le modèle en spécifiant l'optimiseur à utiliser (Adam), la fonction de perte (binary_crossentropy pour une tâche de segmentation binaire), et les métriques à suivre pendant l'entraînement (accuracy pour évaluer la précision du modèle).

Cette structure de modèle et de compilation est cruciale pour préparer le réseau à être entraîné sur les données d'entraînement et évalué sur les données de validation.

III.12. Construction du Modèle U-Net

La construction du modèle U-Net est réalisée à l'aide de couches convolutives, de pooling et de convolution transposée. Le modèle U-Net est composé de deux parties principales : l'encodeur et le décodeur, avec une partie centrale souvent appelée le bottleneck. Voici comment elles sont structurées :

```
# Build U-Net model
input_shape = (SIZE, SIZE, 1)
inputs = Input(input_shape)
s1 = Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(inputs)
s1 = BatchNormalization()(s1)
s1 = Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(s1)
s1 = BatchNormalization()(s1)
p1 = MaxPooling2D(pool_size=(2, 2))(s1)

s2 = Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(p1)
s2 = BatchNormalization()(s2)
s2 = Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(s2)
s2 = BatchNormalization()(s2)
p2 = MaxPooling2D(pool_size=(2, 2))(s2)

s3 = Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(p2)
s3 = BatchNormalization()(s3)
s3 = Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(s3)
s3 = BatchNormalization()(s3)
p3 = MaxPooling2D(pool_size=(2, 2))(s3)

s4 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(p3)
s4 = BatchNormalization()(s4)
s4 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(s4)
s4 = BatchNormalization()(s4)
p4 = MaxPooling2D(pool_size=(2, 2))(s4)
```

```
b = Conv2D(1024, 3, activation='relu', padding='same', kernel_initializer='he_normal')(p4)
b = BatchNormalization()(b)
b = Conv2D(1024, 3, activation='relu', padding='same', kernel_initializer='he_normal')(b)
b = BatchNormalization()(b)
```

```
u1 = Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(b)
u1 = concatenate([u1, s4])
c1 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(u1)
c1 = BatchNormalization()(c1)
c1 = Conv2D(512, 3, activation='relu', padding='same', kernel_initializer='he_normal')(c1)
c1 = BatchNormalization()(c1)

u2 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c1)
u2 = concatenate([u2, s3])
c2 = Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(u2)
c2 = BatchNormalization()(c2)
c2 = Conv2D(256, 3, activation='relu', padding='same', kernel_initializer='he_normal')(c2)
c2 = BatchNormalization()(c2)

u3 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c2)
u3 = concatenate([u3, s2])
c3 = Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(u3)
c3 = BatchNormalization()(c3)
c3 = Conv2D(128, 3, activation='relu', padding='same', kernel_initializer='he_normal')(c3)
c3 = BatchNormalization()(c3)

u4 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c3)
u4 = concatenate([u4, s1])
c4 = Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(u4)
c4 = BatchNormalization()(c4)
c4 = Conv2D(64, 3, activation='relu', padding='same', kernel_initializer='he_normal')(c4)
c4 = BatchNormalization()(c4)

outputs = Conv2D(1, 1, activation='sigmoid')(c4)

model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 3.28 : *architecture de model UNET*

Nous construisons l'architecture U-Net, incluant des couches de convolution pour extraire des caractéristiques, des couches de pooling pour réduire les dimensions, et des couches de convolution transposée pour augmenter les dimensions. Les couches de normalisation par lot (BatchNormalization) sont ajoutées pour stabiliser et accélérer l'entraînement. Le modèle est compilé avec l'optimiseur Adam, la fonction de perte binary_crossentropy et les métriques accuracy et MeanIoU.

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

```
conv2d_14 (Conv2D)          (None, 128, 128, 128)       295040    ['concatenate_2[0][0]']

batch_normalization_14 (Ba  (None, 128, 128, 128)       512       ['conv2d_14[0][0]']
tchNormalization)

conv2d_15 (Conv2D)          (None, 128, 128, 128)       147584    ['batch_normalization_14[0][0]
                                                                  ']

batch_normalization_15 (Ba  (None, 128, 128, 128)       512       ['conv2d_15[0][0]']
tchNormalization)

conv2d_transpose_3 (Conv2D  (None, 256, 256, 64)        32832     ['batch_normalization_15[0][0]
Transpose)                                                        ']

concatenate_3 (Concatenate  (None, 256, 256, 128)       0         ['conv2d_transpose_3[0][0]',
)                                                                  'batch_normalization_1[0][0]'
                                                                  ]

conv2d_16 (Conv2D)          (None, 256, 256, 64)        73792     ['concatenate_3[0][0]']

batch_normalization_16 (Ba  (None, 256, 256, 64)        256       ['conv2d_16[0][0]']
tchNormalization)

conv2d_17 (Conv2D)          (None, 256, 256, 64)        36928     ['batch_normalization_16[0][0]
                                                                  ']

batch_normalization_17 (Ba  (None, 256, 256, 64)        256       ['conv2d_17[0][0]']
tchNormalization)

conv2d_18 (Conv2D)          (None, 256, 256, 1)         65        ['batch_normalization_17[0][0]
                                                                  ']

==================================================================================================
Total params: 31054145 (118.46 MB)
Trainable params: 31042369 (118.42 MB)
Non-trainable params: 11776 (46.00 KB)
```

Figure 3.29 : *UNET sommaire/ visualisation du résumé du modèle*

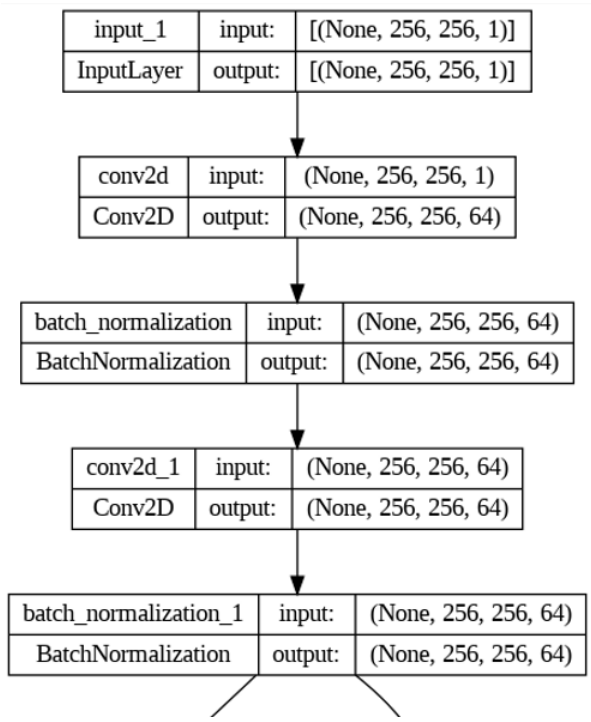


Figure 3.30 : *Afficher une partie de l'architecture du modèle U net dans le notebook*

III.13. Entrainement de modelé

Lors de l'entraînement du modèle U-Net,

III.13.1. Paramètres utilisés

- **batch_size :** 16 échantillons par itération, influence vitesse et stabilité.
- **epochs :** Entraînement sur 25 epochs pour améliorer performance sans surapprentissage.
- **verbose :** Affiche détails précis sur chaque epoch pendant l'entraînement.
- **validation_data :** Utilise X_test et y_test pour évaluer la performance du modèle.
- **shuffle :** Mélange des données pour éviter surapprentissage, introduit variabilité nécessaire.

```
# Model training
history = model.fit(X_train_augmented, y_train_augmented,
                    batch_size=16,
                    verbose=1,
                    epochs=25,
                    validation_data=(X_test, y_test),
                    shuffle=True)
```

```
116/116 [==============================] - 95s 819ms/step - loss: 0.0685 - accuracy: 0.9265 - val_loss: 0.1565 - val_accuracy: 0.9112
Epoch 12/25
116/116 [==============================] - 95s 822ms/step - loss: 0.0647 - accuracy: 0.9265 - val_loss: 0.1525 - val_accuracy: 0.9114
Epoch 13/25
116/116 [==============================] - 95s 820ms/step - loss: 0.0604 - accuracy: 0.9267 - val_loss: 0.1483 - val_accuracy: 0.9119
Epoch 14/25
116/116 [==============================] - 95s 822ms/step - loss: 0.0575 - accuracy: 0.9267 - val_loss: 0.1513 - val_accuracy: 0.9119
Epoch 15/25
116/116 [==============================] - 95s 822ms/step - loss: 0.0554 - accuracy: 0.9267 - val_loss: 0.1464 - val_accuracy: 0.9120
Epoch 16/25
116/116 [==============================] - 96s 824ms/step - loss: 0.0530 - accuracy: 0.9268 - val_loss: 0.1515 - val_accuracy: 0.9121
Epoch 17/25
116/116 [==============================] - 95s 822ms/step - loss: 0.0508 - accuracy: 0.9269 - val_loss: 0.1457 - val_accuracy: 0.9122
Epoch 18/25
116/116 [==============================] - 95s 819ms/step - loss: 0.0495 - accuracy: 0.9269 - val_loss: 0.1470 - val_accuracy: 0.9125
Epoch 19/25
116/116 [==============================] - 95s 821ms/step - loss: 0.0485 - accuracy: 0.9269 - val_loss: 0.1475 - val_accuracy: 0.9121
Epoch 20/25
116/116 [==============================] - 95s 820ms/step - loss: 0.0475 - accuracy: 0.9269 - val_loss: 0.1494 - val_accuracy: 0.9121
Epoch 21/25
116/116 [==============================] - 95s 820ms/step - loss: 0.0457 - accuracy: 0.9270 - val_loss: 0.1530 - val_accuracy: 0.9125
Epoch 22/25
116/116 [==============================] - 96s 824ms/step - loss: 0.0447 - accuracy: 0.9270 - val_loss: 0.1501 - val_accuracy: 0.9126
Epoch 23/25
116/116 [==============================] - 95s 820ms/step - loss: 0.0442 - accuracy: 0.9270 - val_loss: 0.1482 - val_accuracy: 0.9126
Epoch 24/25
116/116 [==============================] - 95s 821ms/step - loss: 0.0434 - accuracy: 0.9270 - val_loss: 0.1506 - val_accuracy: 0.9124
Epoch 25/25
116/116 [==============================] - 95s 821ms/step - loss: 0.0426 - accuracy: 0.9270 - val_loss: 0.1532 - val_accuracy: 0.9124
```

Figure 3.31 : *Visualisation de l'entrainement*

III.13.2. Hyperparamètres utilisés

- **batch_size** : 16
- **epochs** : 25
- **verbose** : 1 (affiche les détails)
- **optimizer** : Adam avec un taux d'apprentissage de 0.0001
- **loss** : binary_crossentropy (fonction de perte pour la segmentation binaire)
- **metrics** : accuracy (métrique pour évaluer la performance du modèle)

Ensuite, nous devons le sauvegarder :

III.14. Sauvegarde du modèle

```
# Save the model
model.save('/content/drive/MyDrive/Colab Notebooks/saved_models/code_using_unet_256.hdf5')
```

Figure 3.32 : *code de Sauvegarde du modèle*

Cette méthode enregistre l'architecture, les poids et l'état de l'entraînement du modèle dans un fichier HDF5. Cela permet de recharger le modèle plus tard sans avoir à l'entraîner de nouveau.

III.15. Visualisation des résultats de l'entraînement

Pour évaluer la performance du modèle, les pertes et précisions d'entraînement et de validation sont tracées. Les résultats obtenus de l'entraînement et de la validation du modèle U-Net pour la segmentation des vaisseaux rétiniens sont les suivants :

- **Moyenne Training Loss** : 0.1062
- **Moyenne Validation Loss** : 0.1808
- **Moyenne Training Accuracy** : 0.9144
- **Moyenne Validation Accuracy** : 0.9057

Nous calculons les moyennes et les écart-types des métriques pour évaluer les performances globales du modèle.

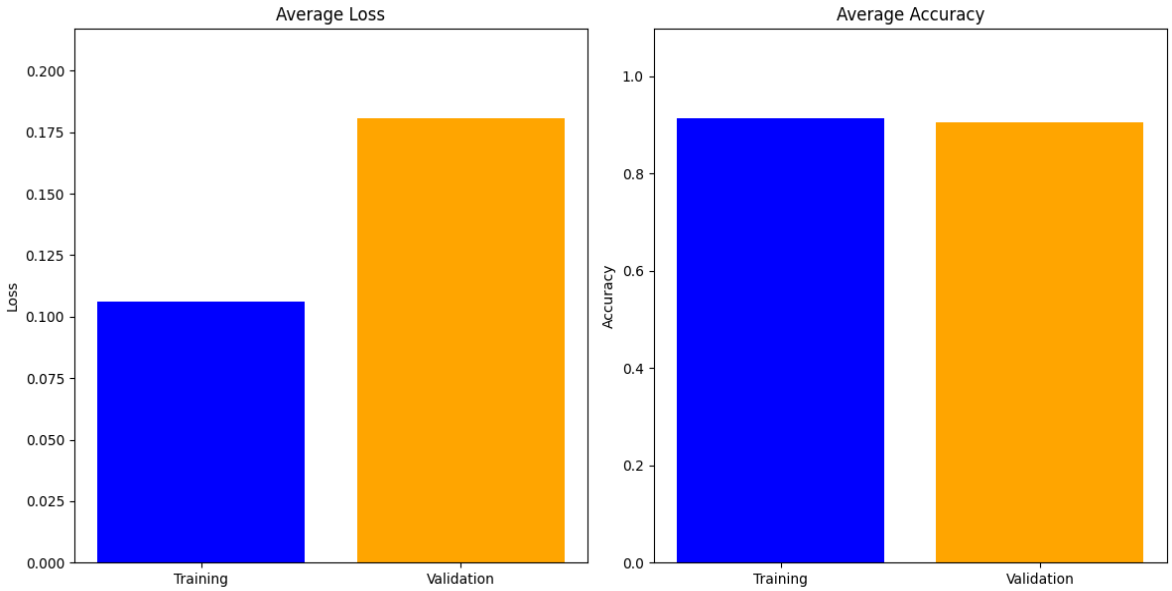


Figure 3.33 : *Affichage le nombre perte (loss) et le nomber de précision (accuracy) du réseau Unet*

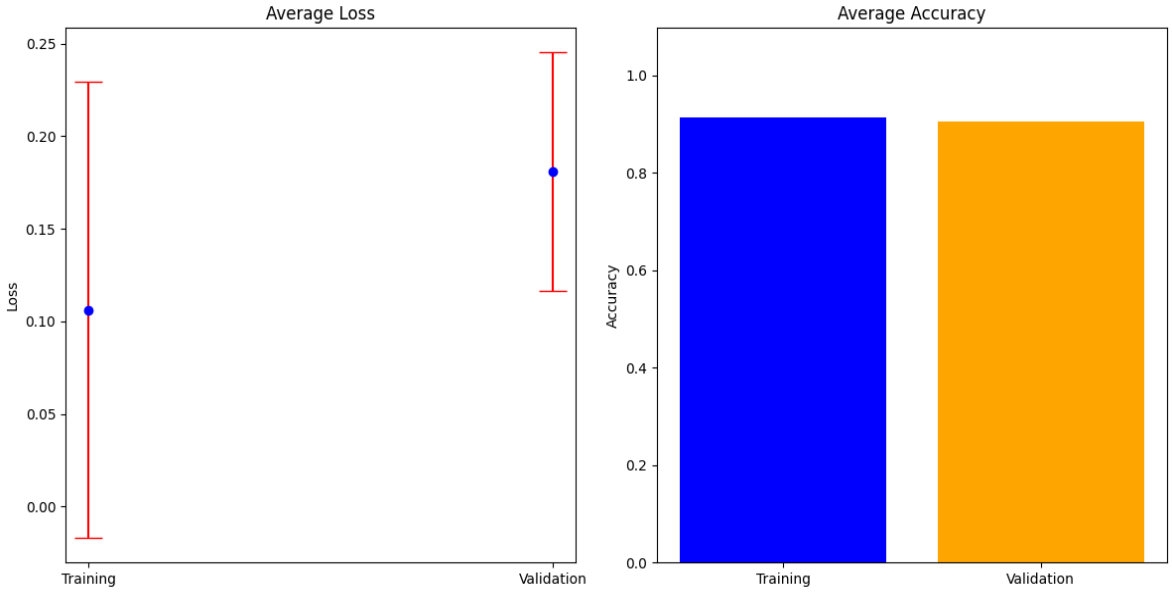


Figure 3.34 : *Affichage moyenne le nombre perte (loss) et le nombre de précision (accuracy) du réseau Unet*

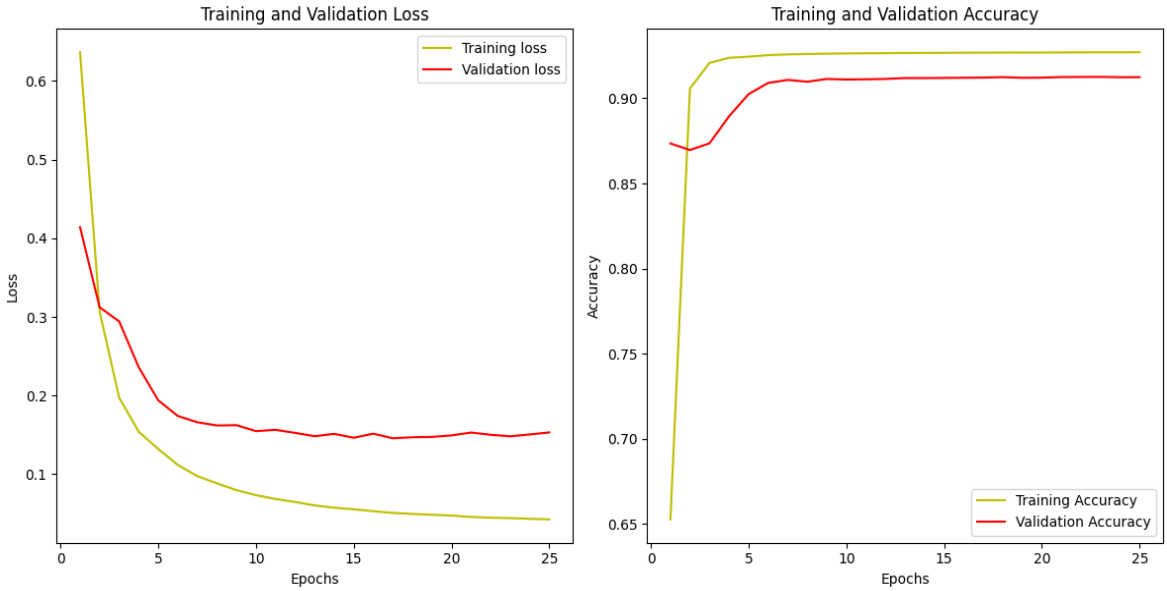


Figure 3.35 : *La courbe de perte (loss) et la courbe de précision (accuracy) du réseau Unet*

III.16. Ces résultats montrent une performance globale solide du modèle sur les tâches de segmentation des images de rétine.

III.16.1. Évaluation du modèle

Nous évaluons le modèle en utilisant le Mean IoU et visualisons les prédictions par rapport aux images de test et aux vérités terrain. Nous évaluons les performances du modèle sur l'ensemble de test en termes de perte, précision et Mean IoU. Nous affichons également les courbes d'apprentissage pour visualiser la perte et la précision pendant l'entraînement et la validation.

- **Calcul de l'IoU**

Pour évaluer la qualité des prédictions du modèle, nous calculons l'Intersection over Union (IoU) moyen. Cela se fait après avoir appliqué un seuil aux prédictions pour obtenir des masques binaires.

```
# Calcule de Mean IoU
IOU_keras = MeanIoU(num_classes=2)
IOU_keras.update_state(y_test, y_pred_thresholded)
print("Mean IoU =", IOU_keras.result().numpy())
```

```
Mean IoU = 0.73998153
```

Figure 3.36 : *code de calcule mean iou*

L'IoU moyen est une métrique clé pour évaluer les performances des modèles de

segmentation, particulièrement en ce qui concerne la qualité de la prédiction des masques. La valeur de l'IoU moyen obtenue est :

– **Mean IoU** : 0.73998153

Cette valeur montre que le modèle a une bonne capacité à segmenter les images, avec une précision acceptable. L'IoU étant proche de 0.74 indique que le modèle est capable de prédire les contours des masques avec une bonne précision.

III.17. Visualisation des prédictions

– **Affichage des Prédictions avec IoU**

Pour visualiser quelques exemples de prédictions et d'évaluation IoU, nous sélectionnons aléatoirement quelques images de test.

```
# Sélectionner quelques images de test aléatoires pour la visualisation
num_samples_to_display = 4
random_indices = random.sample(range(len(X_test)), num_samples_to_display)

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

for i, idx in enumerate(random_indices):
    # Image originale
    plt.subplot(num_samples_to_display, 4, i*4 + 1)
    plt.imshow(X_test[idx][:, :, 0], cmap='gray')
    plt.title('Image Originale')
    plt.axis('off')

    # Masque de vérité terrain
    plt.subplot(num_samples_to_display, 4, i*4 + 2)
    plt.imshow(y_test[idx][:, :, 0], cmap='gray')
    plt.title('Masque de Vérité Terrain')
    plt.axis('off')

    # Masque prédit (seuillé)
    plt.subplot(num_samples_to_display, 4, i*4 + 3)
    y_pred = model.predict(np.expand_dims(X_test[idx], axis=0))[0, :, :, 0]
    y_pred_thresholded = (y_pred > 0.4).astype(np.uint8)
    plt.imshow(y_pred_thresholded, cmap='gray')
    plt.title('Masque Prédit')
    plt.axis('off')

    # Superposition de l'image originale et du masque prédit
    plt.subplot(num_samples_to_display, 4, i*4 + 4)
    plt.imshow(X_test[idx][:, :, 0], cmap='gray')
    plt.imshow(y_pred_thresholded, alpha=0.4, cmap='jet')
    plt.title('Superposition')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

Figure 3.37 : *Code pour Visualisation des prédictions*

- Nous sélectionnons aléatoirement quelques indices d'images de test (random_indice) pour la visualisation.

- Pour chaque image sélectionnée, nous affichons l'image originale, le masque réel, le masque prédit seuillé et une superposition de l'image originale avec le masque prédit.
- Le titre de chaque subplot de masque prédit inclut la valeur d'IoU moyenne calculée précédemment (mean iou), permettant ainsi une évaluation visuelle rapide de la qualité des prédictions.

En séparant ces parties distinctement, il devient plus clair comment le modèle est évalué sur un ensemble de données de test séparé et comment ses performances sont visualisées et interprétées à l'aide de l'IoU moyen.

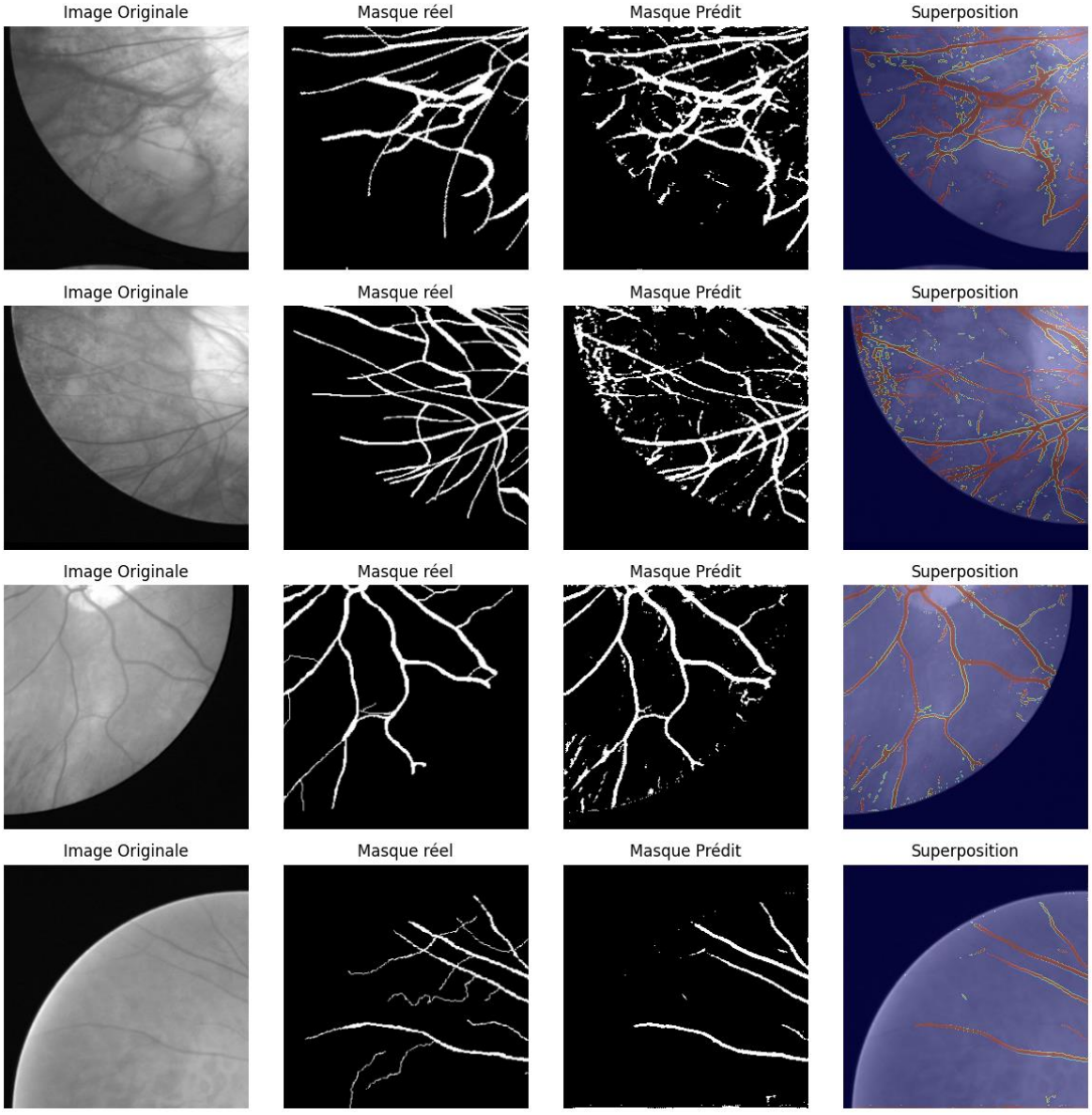


Figure 3.38 : *le résultat*

III.18. Discussion

III.18.1. Analyse des Résultats

1. **Perte Moyenne d'Entraînement et de Validation** :
    - La perte moyenne d'entraînement de 0.1062 indique que le modèle a bien appris les caractéristiques des données d'entraînement.
    - La perte moyenne de validation de 0.1808 est légèrement plus élevée, ce qui est attendu, car le modèle peut rencontrer des données qu'il n'a jamais vues pendant la validation. Cependant, une augmentation significative de la perte de validation peut indiquer un certain sur apprentissage.
2. **Précision Moyenne d'Entraînement et de Validation** :
    - La précision moyenne d'entraînement de 0.9144 indique que le modèle fait de bonnes prédictions sur les données d'entraînement.
    - La précision moyenne de validation de 0.9057, étant très proche de celle de l'entraînement, suggère que le modèle généralise bien aux nouvelles données. Cela indique que le modèle n'est pas fortement sur appris.
3. **Mean IoU** :
    - La valeur de l'IoU moyen de 0.73998153 indique une bonne capacité du modèle à segmenter l’arbre rétinien. Un IoU moyen supérieur à 0.7 est généralement considéré comme un bon résultat pour les tâches de segmentation.

III.18.2. Points Forts du Modèle

- **Bonne Précision** : Les hautes valeurs de précision pour les ensembles d'entraînement et de validation montrent que le modèle est efficace pour la prédiction des pixels appartenant aux l’arbre rétinien.
- **Généralisation** : La faible différence entre les performances d'entraînement et de validation suggère que le modèle généralise bien, ce qui est essentiel pour les applications réelles.

III.18.3. Limites et Améliorations Potentielles

- **Légère Sur apprentissage** : Bien que les performances de validation soient bonnes, la légère augmentation de la perte de validation par rapport à la perte d'entraînement suggère un léger sur apprentissage. L'ajout de techniques de régularisation supplémentaires, comme le dropout, ou l'utilisation de davantage de données d'entraînement pourrait aider à atténuer ce problème.
- **Amélioration de l'IoU** : Bien que l'IoU soit bon, il y a toujours place à l'amélioration. Des techniques avancées de prétraitement des images, une augmentation de la diversité des données d'entraînement ou l'optimisation des hyper paramètres pourraient contribuer à augmenter ce score.

III.19. Conclusion

Le modèle U-Net montre des performances prometteuses pour la segmentation de l’arbre rétinien, avec une haute précision et un IoU moyen satisfaisant. Les résultats indiquent que le modèle est capable de généraliser bien aux nouvelles données, tout en maintenant une bonne capacité de prédiction des masques de segmentation. Des améliorations peuvent encore être apportées pour optimiser les performances du modèle et mieux gérer les défis des données non vues.

Chapitre 04 :

Implémentation sur FPGA

IV.1. Introduction

Les réseaux de neurones convolutifs profonds (CNN) représentent actuellement la pointe de la technologie pour la classification et la segmentation des images, principalement en raison de leur grande précision. L'accélération de ces réseaux est cruciale pour permettre leur utilisation dans des applications en temps réel. Les unités de traitement graphique (GPU) sont souvent employées à cet effet, mais leur consommation d'énergie élevée limite leur intégration dans des dispositifs utilisés quotidiennement. C'est pourquoi nous avons choisi d'explorer l'utilisation des FPGA (Field-Programmable Gate Arrays) qui offrent une faible consommation d'énergie et une architecture flexible, mieux adaptée aux implémentations des CNN.

Nous décrivons les étapes d'implémentation d'un modelé CNN predéfini.la carte dont disposons n'étant pas appropriée (vu ses caractéristiques) a l'implémentation d'un modèle plus grand comme U-NET.

Nous implémentons sur une carte FPGA PYNQ Z2 (Xilinx) un modèle CNN prédéfini pour la détection des numéros en utilisant la base de données MNIST. Ce modèle reçoit en entrée des images de chiffres manuscrits (de 0 à 9) et les reconnaît, La configuration permet d'afficher le résultat de l'inférence du réseau neuronal via des LED embarquées ou de le renvoyé à un PC hôte via UART.

Ce chapitre détaille les étapes suivies pour réaliser cette implémentation.

IV.2. Les différents langage et plateforme de développement

IV.2.1. Les langages de description matérielle

- **Verilog**

Verilog est un langage de description matériel utilisé pour modéliser et concevoir des circuits électroniques numériques.

- **VHDL**

VHDL (VHSIC Hardware Description Langage) est un langage de description matériel utilisé pour modéliser, concevoir et vérifier des systèmes électroniques complexes.

IV.2.2. Les outils de conception

- **Vitis Model Composer :** Conception de systèmes, modélisation graphique, génération automatique de code.
- **Xilinx Vivado Design Suite :** Conception, synthèse, implémentation, simulation et débogage de FPGA. Supporte VHDL et Verilog.
- **Unified Software Platform :** Développement unifié pour matériel et logiciel, supporte l'accélération matérielle, programmation en C/C++ et OpenCL.
- **Intel Quartus Prime (ancienne Altera) :** Conception, synthèse, simulation et vérification de FPGA. Supporte VHDL et Verilog.

IV.2.2.1. Les outils choisissent pour notre cas de développements FPGA

Pour cette implémentation, nous avons utilisé :

- **Vitis HLS 2021.1** : une plateforme de conception de circuits logiques complexes pour FPGA, permettant l'utilisation de langages de haut niveau tels que C++, C et OpenCL.
- **Vivado 2021.1 :** un environnement pour la conception et la simulation de circuits logiques pour FPGA, afin de générer le bitstream nécessaire à l'implémentation.
- **Vitis 2021.1** : pour le développement des applications nécessitant l'accélération matérielle des FPGA.
- **Les bibliothèques et application utilisées sont :** Tensorflow, Open CV, Python et Jupyter.

IV.3. Présentation de la carte PYNQ Z2

La carte FPGA PYNQ-Z2 est une carte de développement développée sur le Soc ZYNQ XC7Z020, spécialement conçue pour le Framework open-source PYNQ. Les développeurs peuvent créer des systèmes intégrés de haute performance en combinant la puissance (arm A9) de calcul et la logique programmable. [32]

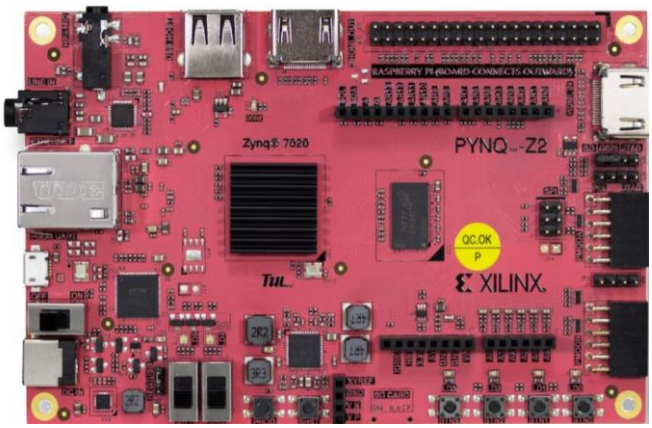


Figure 4.1 : *les interfaces de la carte PYNQ Z2*

Voici une photo détaillée sur laquelle vous pouvez voir toutes les interfaces :

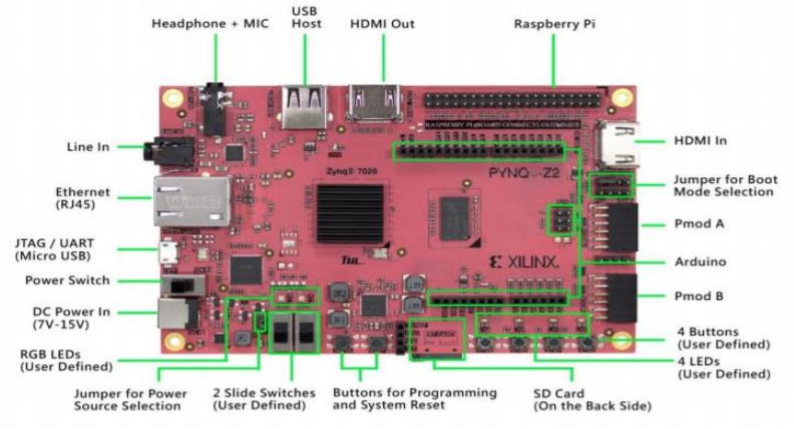


Figure 4.2 : *L'emplacement des composants de la carte PYNQ Z2*

IV.4. Les Avantages de la carte PYNQ Z2

L'enivrement de développement de PYNQ Z2 est simple qui permet aux développeurs de créer des systèmes intégrés sans avoir besoin de créer des circuits logiques programmables.

La programmation en Python permet de programmer directement la carte, ce qui rend la conception d'applications intégrées plus facile.

Les bibliothèques matérielles sont soutenues : Il est possible d'importer et de programmer les bibliothèques matérielles grâce à des API, ce qui permet une programmation plus performante.

IV.4.1. Utilisations

Création d'applications englobantes : La carte a été développée dans le but d'assister les développeurs dans la création de systèmes intégrés de haute performance, en particulier pour la mise en place parallèle de logiciels.

La conception de systèmes embarqués constitue une solution parfaite pour les applications qui requièrent une combinaison de puissance de calcul et de programmation logique.

Voici une photo **Figure 4.3** détaillée sur laquelle vous pouvez voir toutes les interfaces [33] :

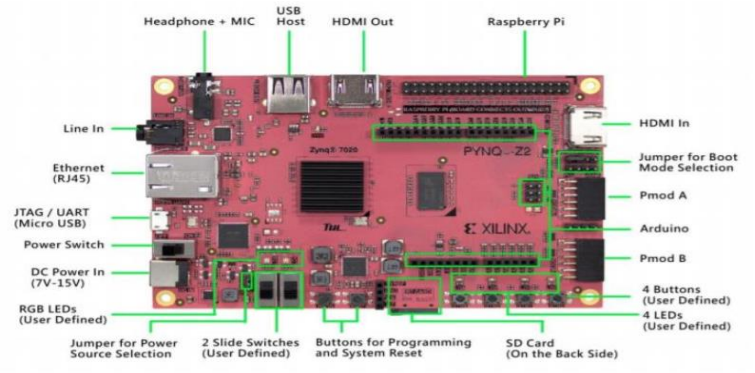


Figure 4.3 : *Interfaces PYNQ-Z2*

IV.4.2. Caractéristique Principales

Tableau 3.2 : les caractéristique principales [32]

Les composants	Caractéristiques
Interfaces E/S	• Circuit de programmation USB-JTAG • USB OTG 2.0 • Pont USB-UART • Un Ethernet 10/100/1G • Entrée HDMI • Sortie HDMI • Interface I2S avec DAC 24 bits avec prise TRRS 3,5 mm • Entrée ligne avec jack 3,5 mm
Mémoire	• 512 Mo DDR3 avec bus 16 bits à 1 050 Mbit/s • Flash Quad-SPI 128 Mbits

	• Connecteur de carte Micro SD
Interrupteurs et LED	• 2 interrupteurs à glissière • 2 LED RVB • 4 LED • 4 boutons poussoirs
Horloges	• Un 125 MHz pour PL • Un 50 MHz pour PS
Ports d'extension	• 2 ports Pmod o 16 E/S FPGA au total (8 broches partagées avec connecteur Raspberry Pi) • 1 connecteur de blindage Arduino o 24 E/S FPGA au total o 6 entrées analogiques asymétriques 0-3,3 V vers XADC • Connecteur Raspberry Pi o 28 E/S FPGA au total (8 broches partagées avec port Pmod A)
Surveillance d'alimentation	• Surveillance active des courants et tensions d'alimentation

IV.5. Préparation de la carte :

PYNQ-Z2 repose sur le système d'exploitation ZYNQ-7020 qui utilise des processeurs ARM Cortex-A9. On nomme cette partie du FPGA le système de traitement (PS).

La partie Programmable Logic (PL) est la deuxième partie, qui fonctionne de la même manière que tous les autres FPGA.

La partie matérielle est principalement la partie que vous pouvez développer en utilisant un IDE appelé VIVADO.

PS et PL peuvent communiquer entre eux à l'aide de l'interface d'interconnexion AXI. L'idée est donc d'exploiter la puissance de Linux sur les processeurs ARM tout en accélérant

les algorithmes sur le PL. Essentiellement, préparer une carte SD signifie que vous installerez un système d'exploitation sur la carte SD à l'aide d'un fichier image et démarrerez les processeurs ARM à partir de la carte SD. Préparons la carte SD et commençons.

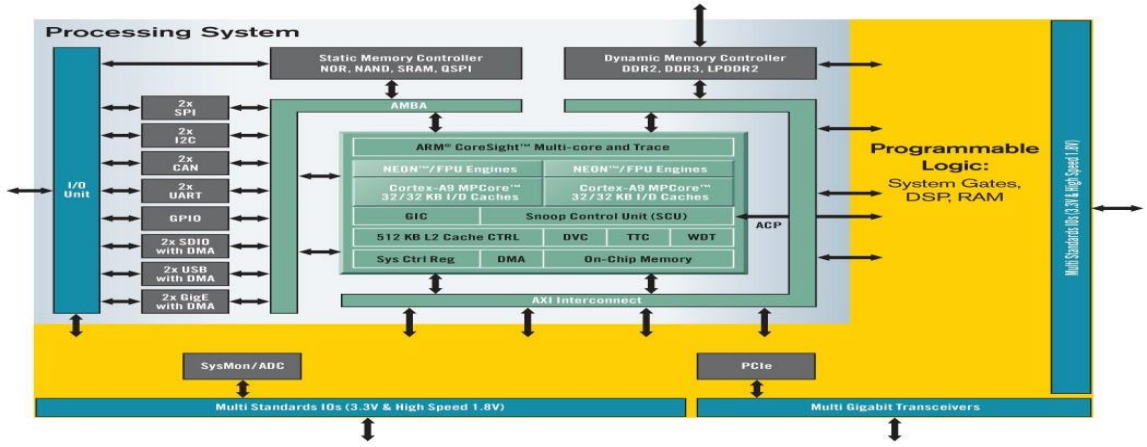


Figure 4.4 : *Système ZYNQ-70xx sur puce, architecture de haut niveau*

Pour préparer cette la carte il faut :

- **Win32 Disk Imager :** nous avons installé le système d'exploitation c'est un utilitaire permettant de transférer l'image disque d'un système d'exploitation sur une carte mémoire SD. [34]
- **PuTTY :** c'est un émulateur de terminal populaire, gratuit et open-source, ainsi qu'une application de console série et de transfert de fichiers réseau. Il prend en charge divers protocoles réseau, notamment SSH, Telnet, login et les connexions de socket brut.
- Nous avons téléchargé le fichier image (dernière image PYNQ-Z2) à partir de [4.2] Une fois le fichier téléchargé, nous avons décompresse l'image iso. [35]

Downloadable PYNQ images

If you have a Zynq board, you need a PYNQ SD card image to get started. You can download a pre-compiled PYNQ image from the table below. If an image is not available for your board, you can build your own SD card image (see details below).

Board	SD card image	Previous versions	Documentation	Board webpage
PYNQ-Z2	v3.0.1	v2.7 v2.6	PYNQ setup guide	TUL Pynq-Z2
PYNQ-Z1	v3.0.1	v2.7 v2.6	PYNQ setup guide	Digilent Pynq-Z1
PYNQ-ZU	v3.0.1	v2.7 v2.6	GitHub project page	TUL PYNQ-ZU
Kria KV260*	Ubuntu 22.04		Kria PYNQ setup	Xilinx Kria KV260
Kria KR260*	Ubuntu 22.04		Kria PYNQ setup	Xilinx Kria KR260
ZCU104	v3.0.1	v2.7 v2.6	PYNQ setup guide	Xilinx ZCU104
RFSoC 2x2	v3.0.1	v2.7 v2.6	RFSoC-PYNQ	XUP RFSoC 2x2
RFSoC 4x2	v3.0.1	v2.7	RFSoC-PYNQ	XUP RFSoC 4x2
ZCU111	v3.0.1	v2.7 v2.6	RFSoC-PYNQ	Xilinx ZCU111
ZCU208	v3.0.1		RFSoC-PYNQ	Xilinx ZCU208
Ultra96V2	v3.0.1	v2.7 v2.6	Avnet PYNQ webpage	Avnet Ultra96V2
Ultra96 (legacy)	v3.0.1	v2.7 v2.6	See Ultra96V2	See Ultra96V2
ZUBoard 1CG	v3.0.1		GitHub project page	Avnet ZUBoard 1CG
TySOM-3-ZU7EV	v2.7		GitHub project page	Aldec TySOM-3-ZU7EV
TySOM-3A-ZU19EG	v2.7		GitHub project page	Aldec TySOM-3A-ZU19EG

*For the Kria KV260 and KR260, follow the links above for guide for getting started with the Ubuntu image, and then follow the Kria PYNQ setup instructions to install PYNQ.

Figure 4.5 : *différentes images iso*

- Après nous avons gravé l'image iso déjà téléchargé sur la carte SD avec utilitaire Win32 Disk Imager.

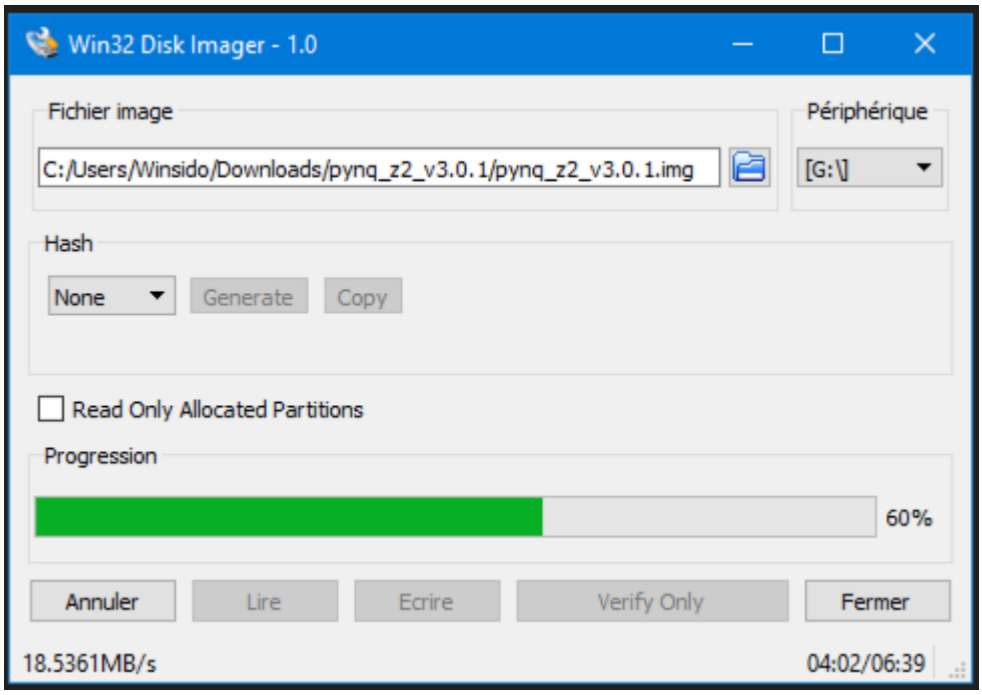


Figure 4.6 : *gravé image sur SD*

À ce moment-là, vous avez exécuté le flash de votre carte Micro SD.

IV.5.1. Configurations de la carte

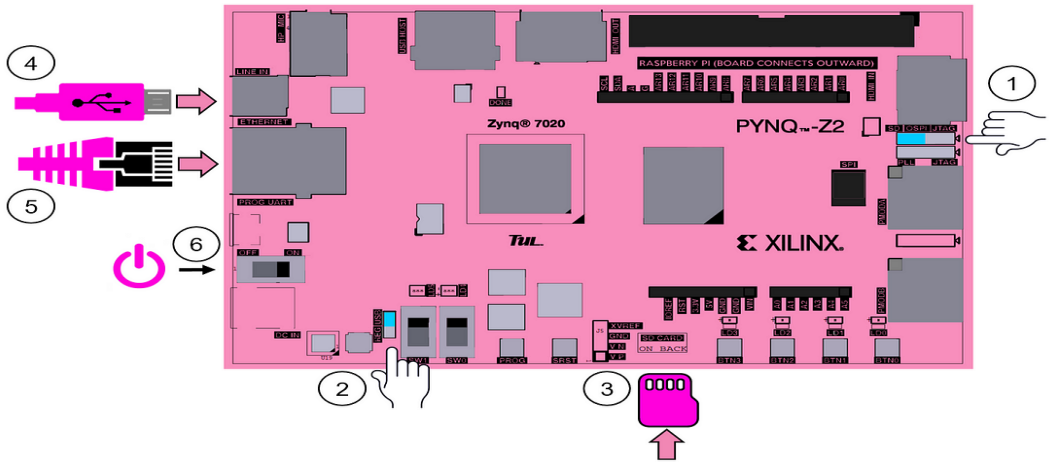


Figure 4.7 : *Configuration de la carte et configuration des cavaliers*

Afin de démarrer votre carte avec une carte Micro SD, nous avons positionné le cavalier en position SD (bleue) et vérifiez que la source d'alimentation est correctement configurée.

Nous avons connecté le câble Micro USB entre la carte et votre ordinateur (pour l'alimenter). Nous avons mise en marche la carte et attendre que les LED rouge s'allument, puis la LED DONE s'allume, puis les LED bleues et Vertes clignotent, ce qui signifie que la carte est prête à être utilisée.

Voici une représentation graphique du démarrage de la carte PYNQ-Z2.

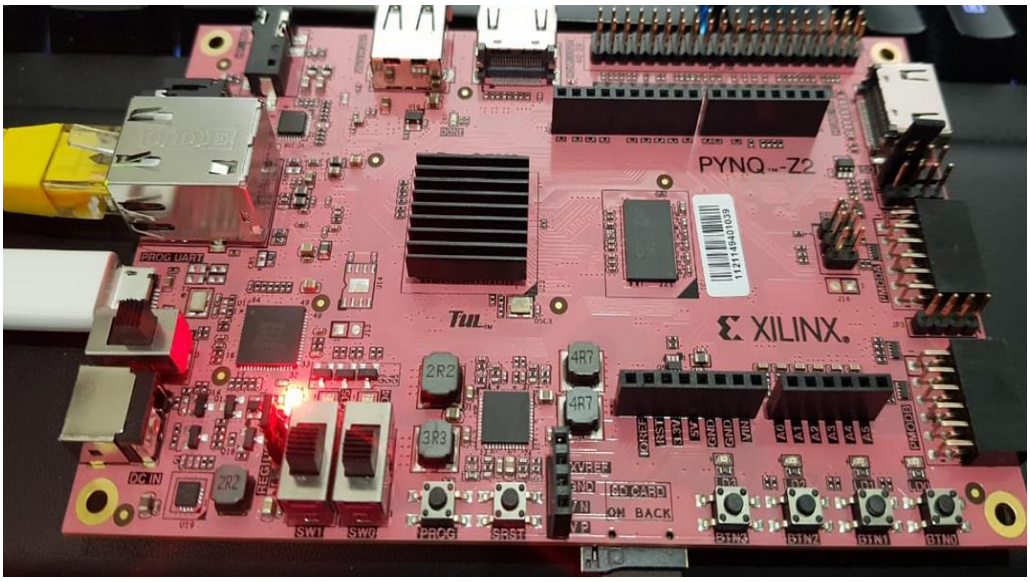


Figure 4.8 : *carte PYNQ en démarrage*

Lorsque vous allumez la carte, seule la LED ROUGE Power s'allumera

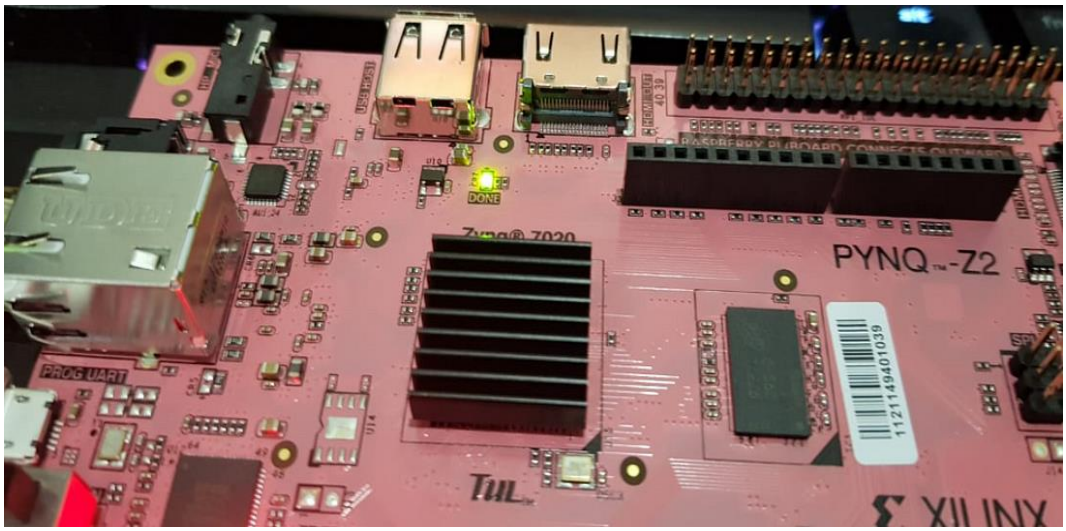


Figure 4.9 : *carte PYNQ en démarrage 1*

Après un certain temps, si votre carte MicroSD clignote correctement, la LED DONE s'allumera.

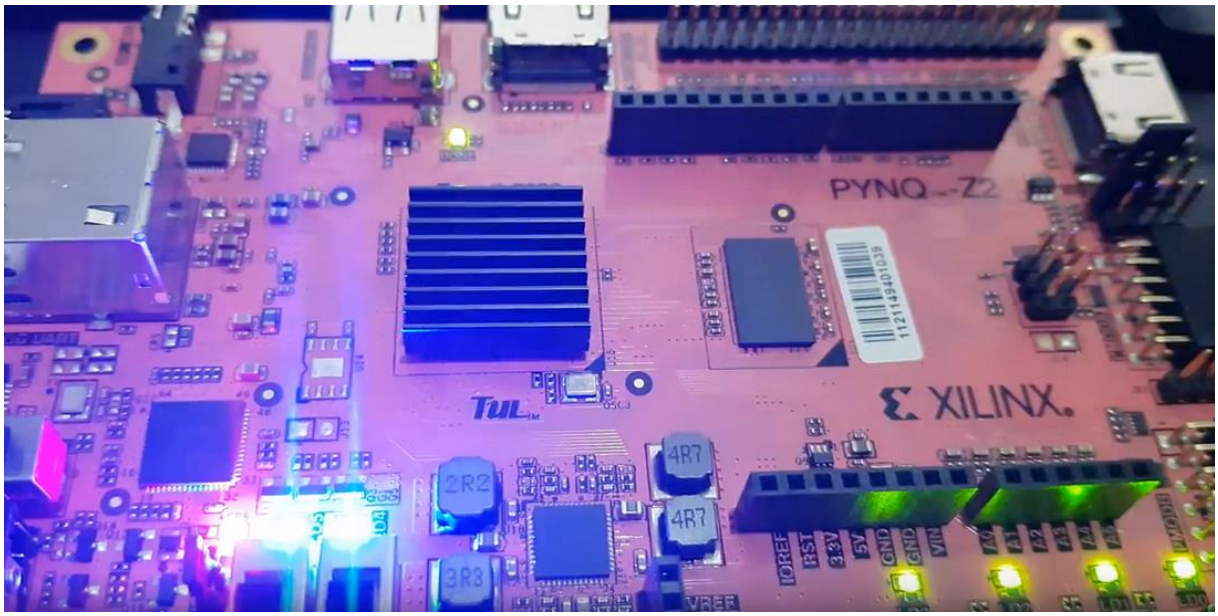


Figure 4.10 : *carte PYNQ en démarrage 2*

Après un certain temps, les LED bleues et vertes commenceront à clignoter.

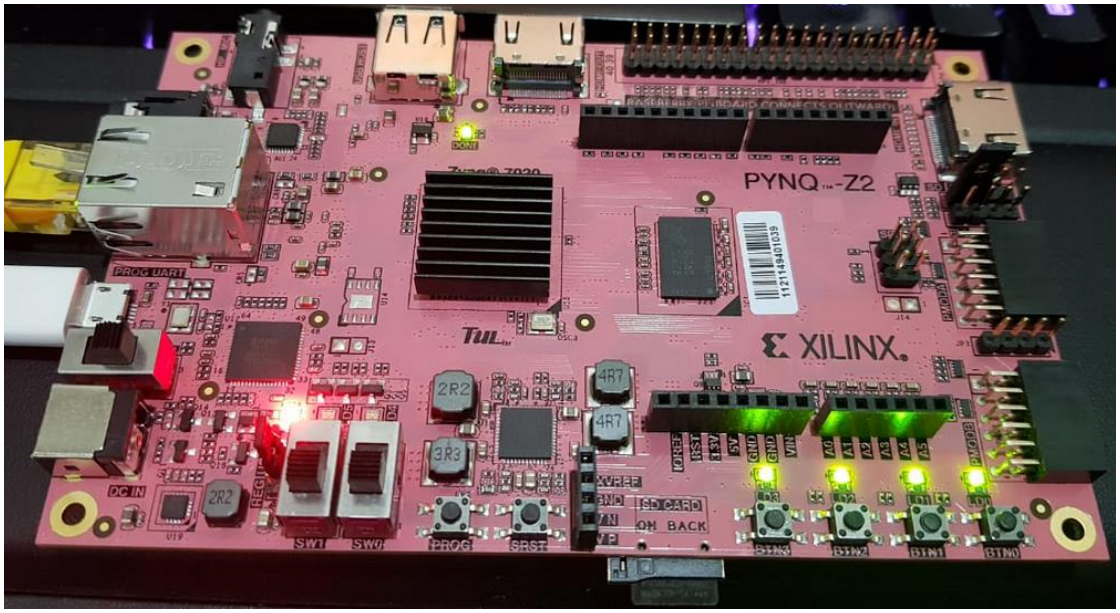


Figure 4.11 : *carte PYNQ en démarrage 3*

Les LED bleues s'éteindront tandis que les LED jaunes resteront allumées. Cela indique que la carte est prête à l'emploi.

IV.5.1.1. Les types de connexion

- **Via Série :**

On a besoin d'un utilitaire tel que (hyper terminal, teratram), Putty utilisé dans notre cas, on configure le port COM4 a une vitesse de 115200 baud/s

User : xilinx

Mot de passe : xilinx

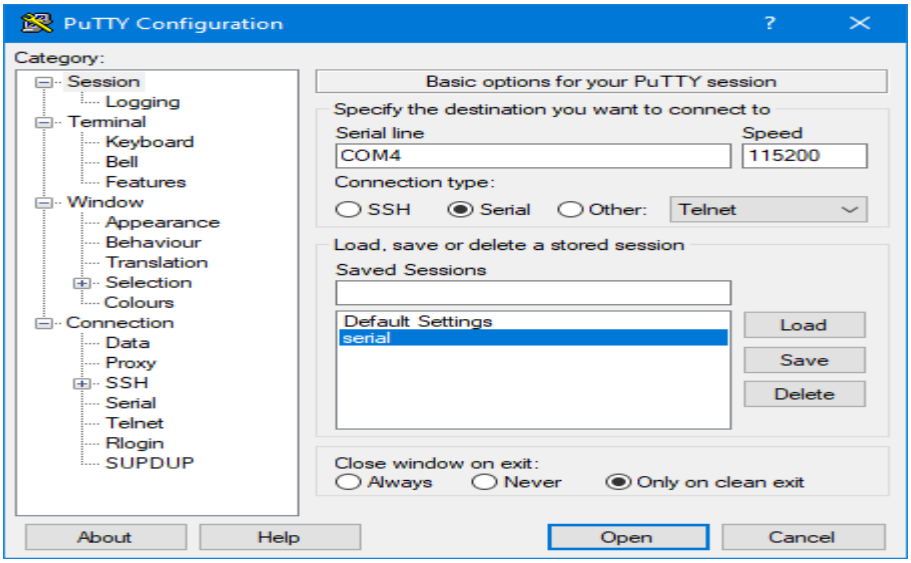


Figure 4.12 : *accès via série*

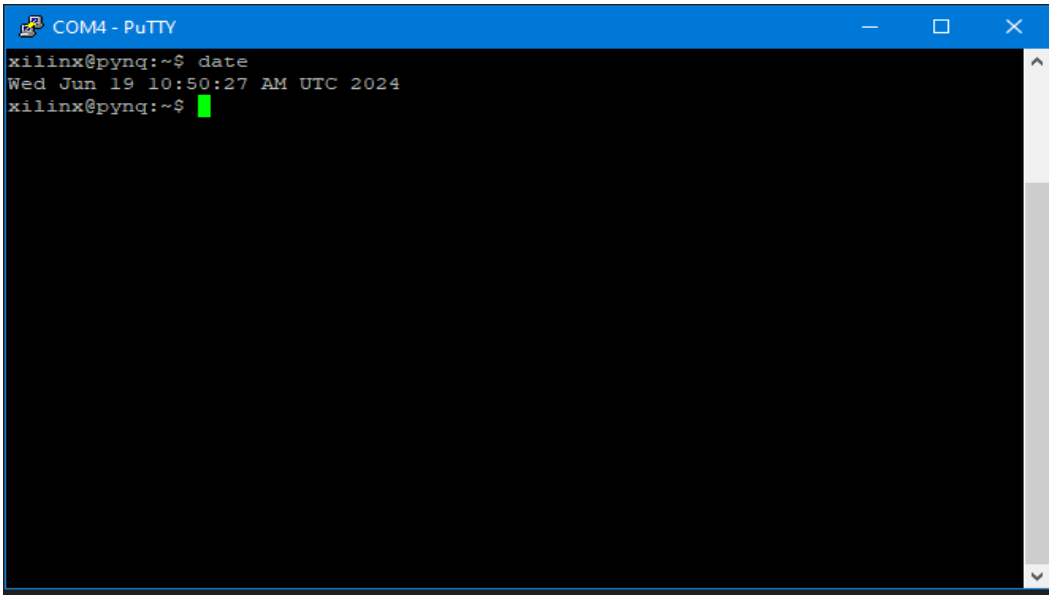


Figure 4.13 : *interface CMD accès série*

Maintenant on peut récupérer adresse IP via commande ‘’ipconfig’’.

- **Via Ethernet**

Si on dispose un routeur avec Protocol DHCP activé pour donner une adresse IP pour la carte PYNQ Z2, et on peut le connecter directement via carte wifi du pc ou Ethernet.

Autrement, on peut configurer une adresse IP fixe sur votre ordinateur en utilisant 192.168.2.1. Et accédé à la carte PYNQ Z2 via adresse par défaut 192.168.2.99.

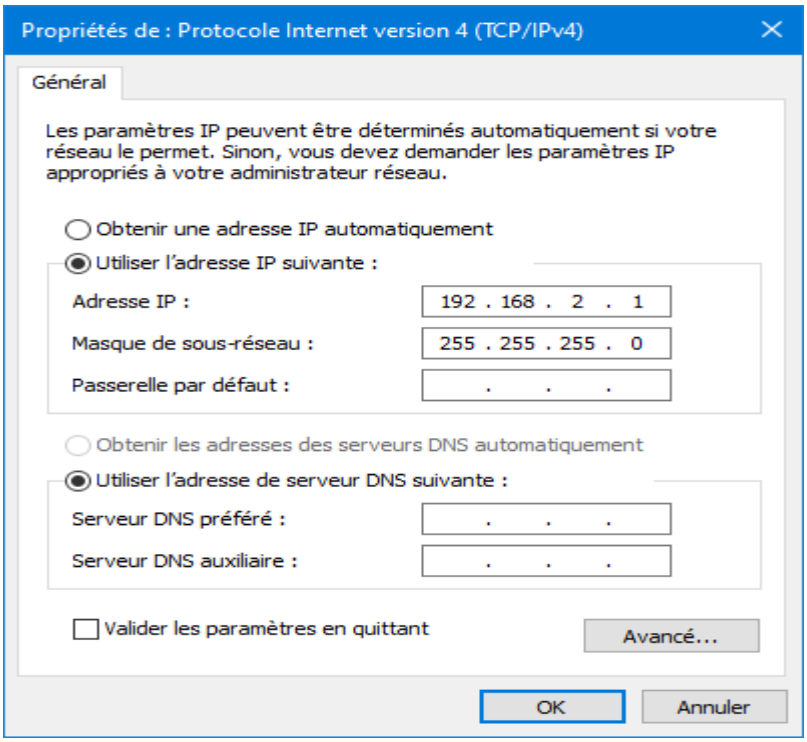


Figure 4.14 : *IP statique pour Ethernet*

On accède via Putty ou bien sur cmd via SSH.

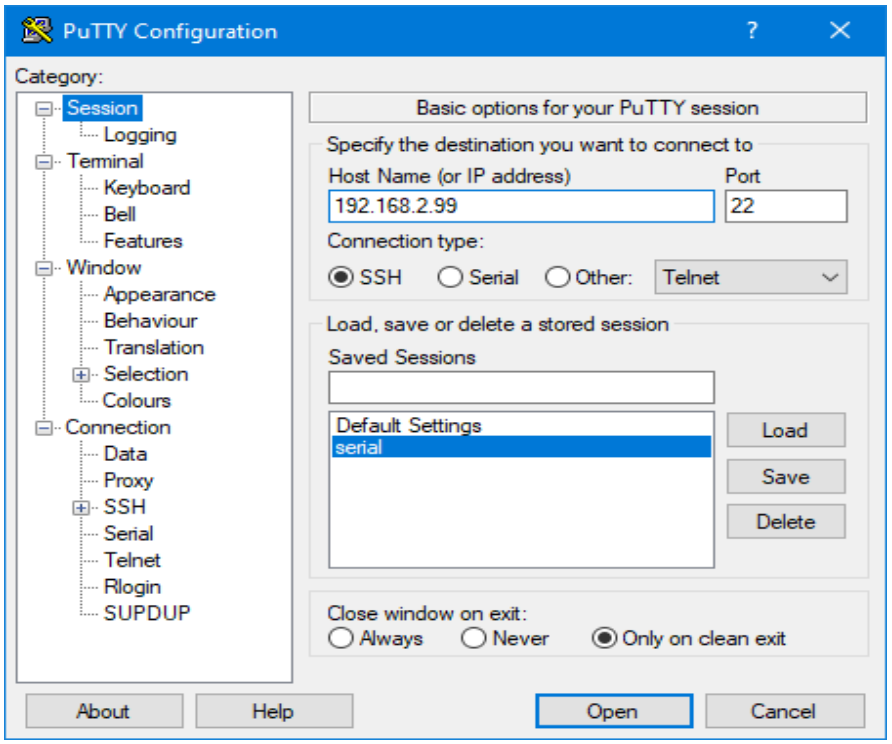


Figure 4.15 : *accès via SSH*

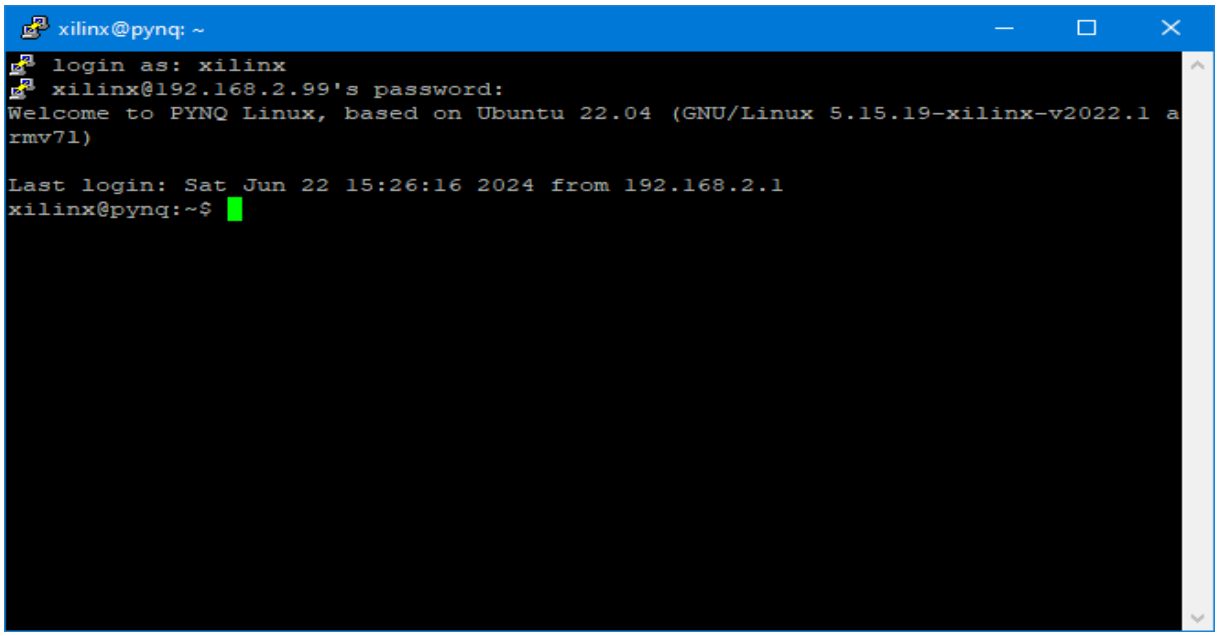


Figure 4.16 : *interface cmd SSH*

Nous avons accédé via navigateur web.

Figure 4.17 : *interface web 1*

En cas de succès, vous serez informé de l'écran qui suit.

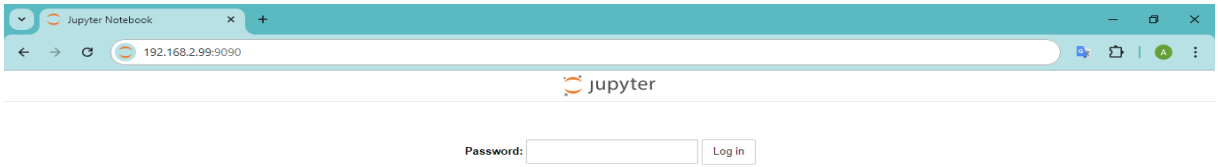


Figure 4.18 : *interface web 2*

Figure 4.19 : *interface web principal de Jupyter-Notebook*

➢ Création de votre premier notebook

Afin de créer notre premier fichier Notebook. Nous allons tout d'abord ouvrir un nouveau dossier en cliquant sur l'option "Nouveau" → "Dossier".

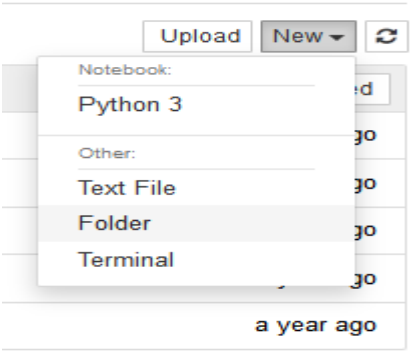


Figure 4.20 : *Création d'un nouveau dossier dans Jupiter-Notebook*

Il y aura un dossier nommé « Dossier sans titre ». On peut le renommer.

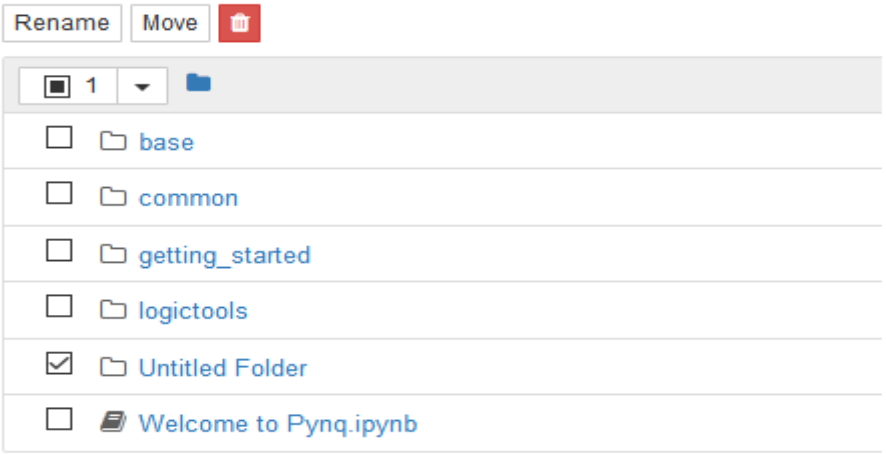


Figure 4.21 : *Création d'un nouveau dossier dans Jupiter-Notebook 1*

Renommer le dossier

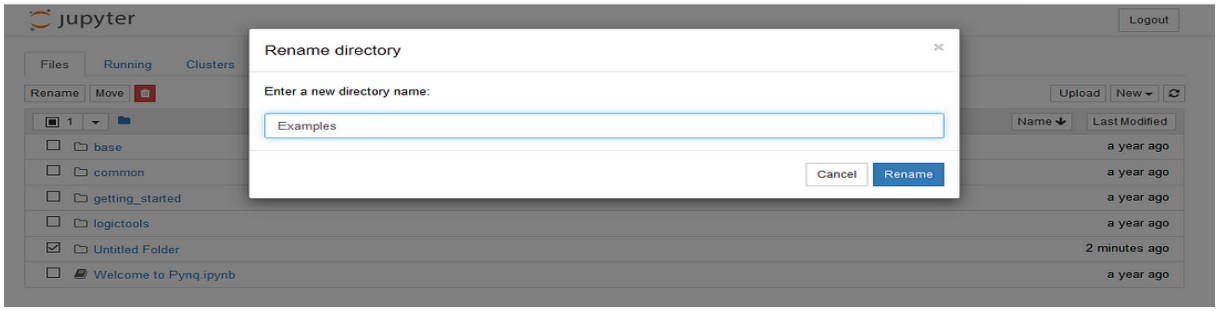


Figure 4.22 : *Création d'un nouveau dossier dans Jupiter-Notebook 2*

Entrez « Exemples » et appuyez sur Renommer. Vous pouvez nommer ce que vous voulez.

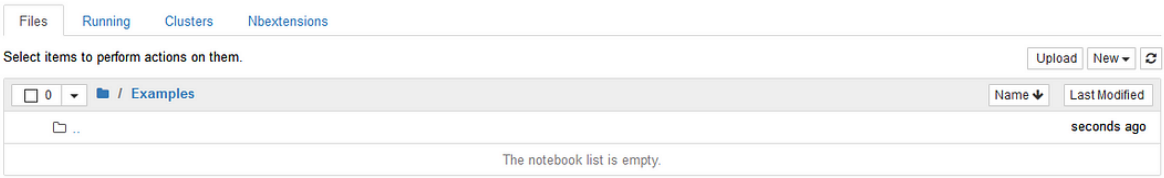


Figure 4.23 : *Exemple de dossier*

Maintenant, nous allons générer un nouveau fichier Python et y exécuter quelques commandes nous avons nommé le fichier « PFE_2024 ».

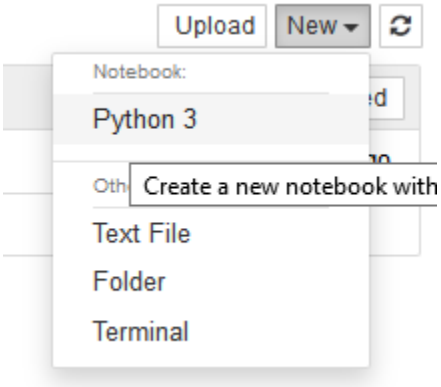


Figure 4.24 : *Création d'un nouveau fichier dans Jupiter-Notebook*

Créer un nouveau bloc-notes Python3

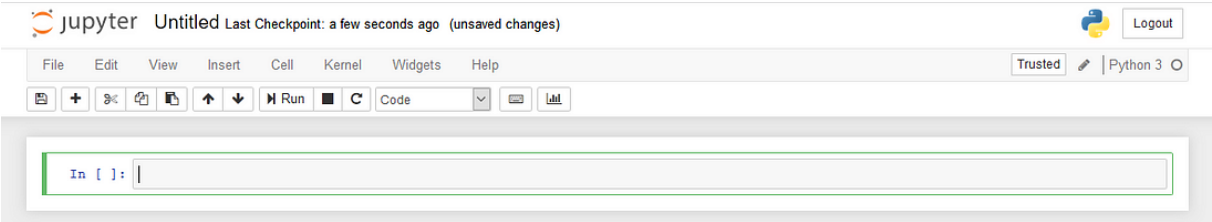


Figure 4.25 : *Un carnet « Sans titre » sera créé.*

IV.6. Les étapes de la réalisation du projet : pour l'implantation d'un CNN sur la carte FPGA PYNQ Z2

Dans notre projet, nous avons commencé par un travail d'initiation afin de nous familiariser avec l'environnement de développement Vitis et les FPGA. Pour cela, nous avons décidé de réaliser un mini-projet. Ce mini-projet nous permet de nous adapter à l'environnement de travail tout en mettant en pratique nos nouvelles connaissances sur Vitis

et les FPGA. L'objectif final de notre projet est l'implémentation de U-net pour la segmentation de l'arbre rétinien sur FPGA.

Dans notre mini-projet on va essayer de suivre les étapes indiquées figure N° 4.23.

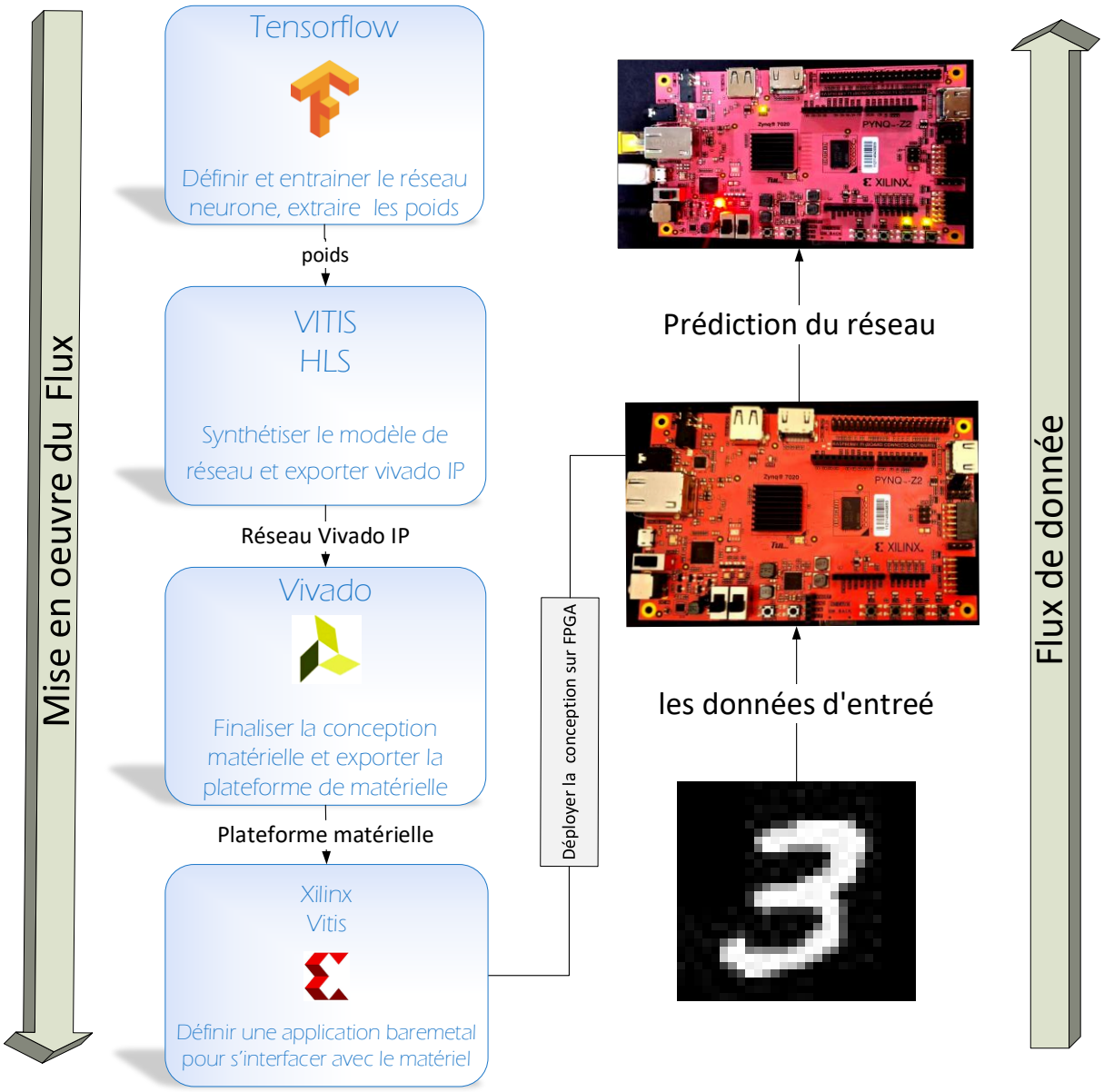


Figure 4.26 : *Les étapes de la réalisation du projet*

IV.6.1. Préparation de l'espace de travail (sur pc)

Nous commençons par préparer on prépare notre espace de travail par téléchargement de la base de données afin de commencer l'entrainement de notre CNN et pouvoir récupérer les poids.

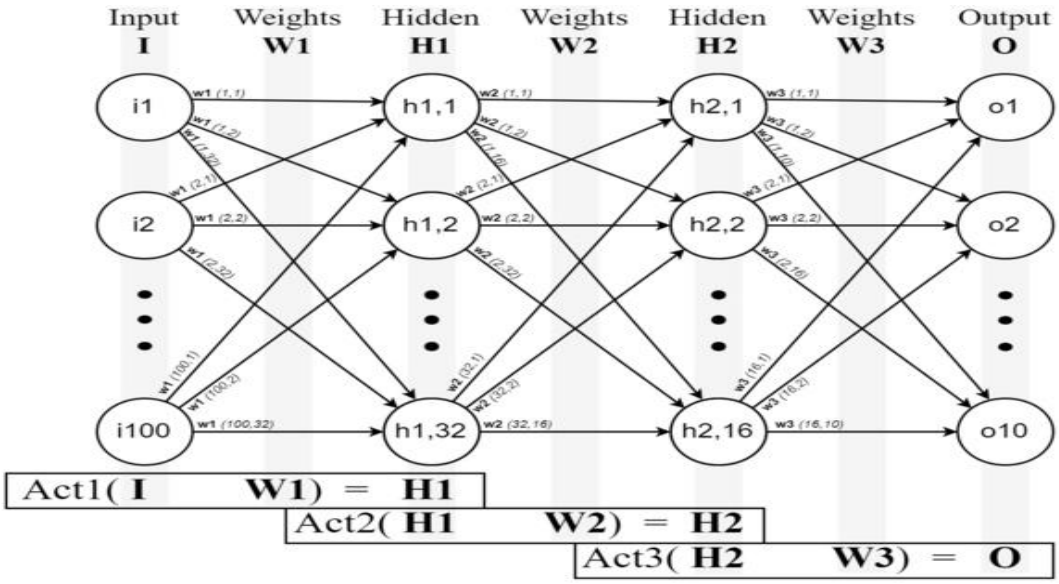


Figure 4.27 : *model de notre réseau neurone*

IV.6.2. Création d'un Project HLS dans Vitis

Nous avons créé un Projet HLS : avec un modèle C++ afin d'exportez le RTL pour obtenir un cœur IP.

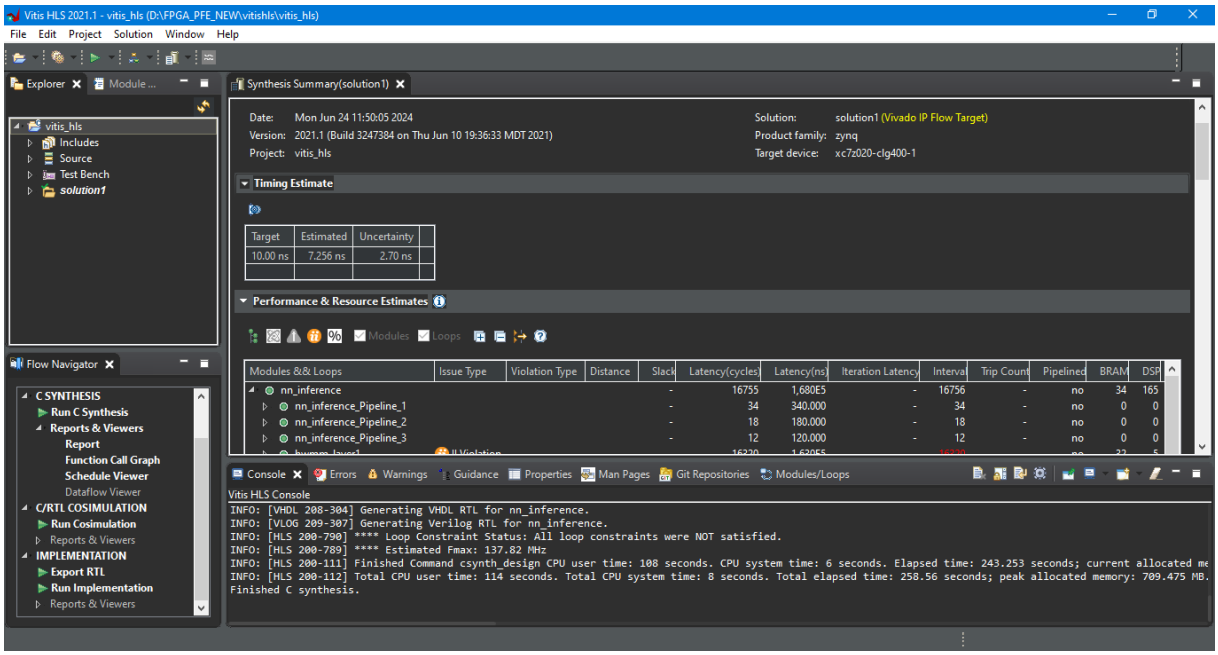


Figure 4.28 : *Vitis HLS*

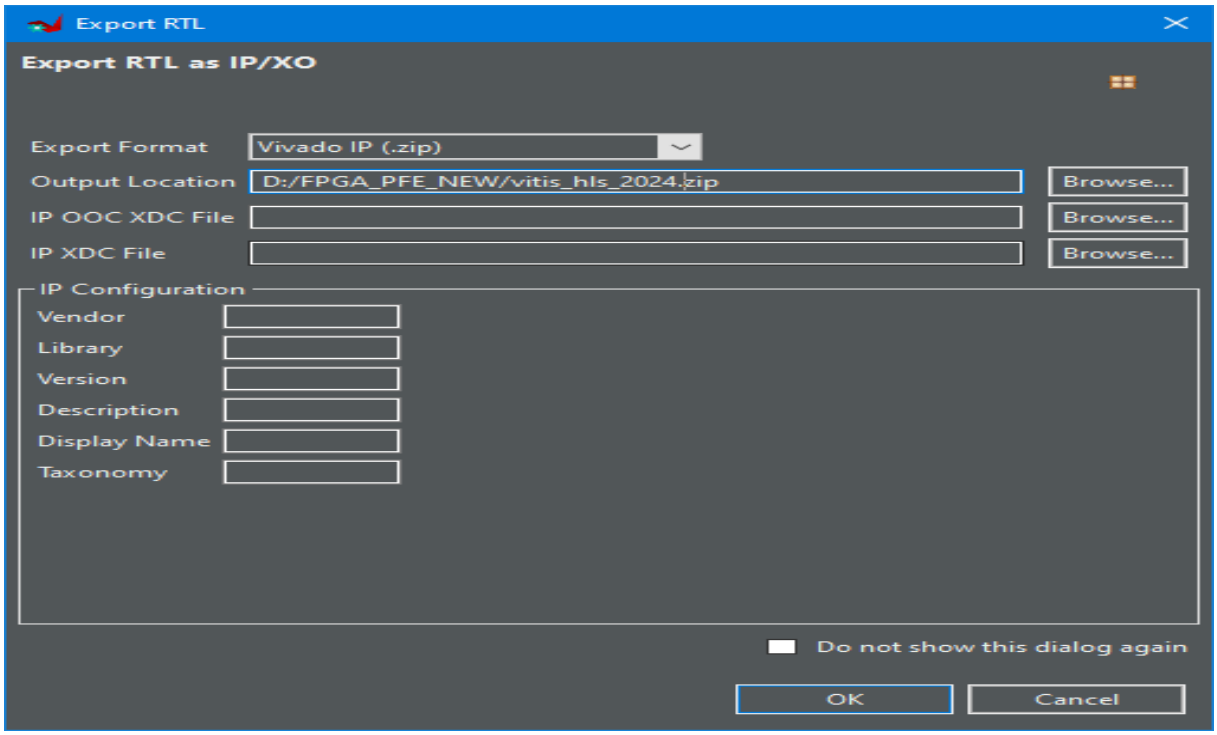


Figure 4.29 : *export RTL*

IV.6.3. Créations d'un Project Vivado

Nous avons créé un projet Vivado: il est nécessaire d'importer le code VHDL puis crée dans Vivado, afin de créer le design nécessaire puis générer le bitstream et l'exporte vers le matériel.

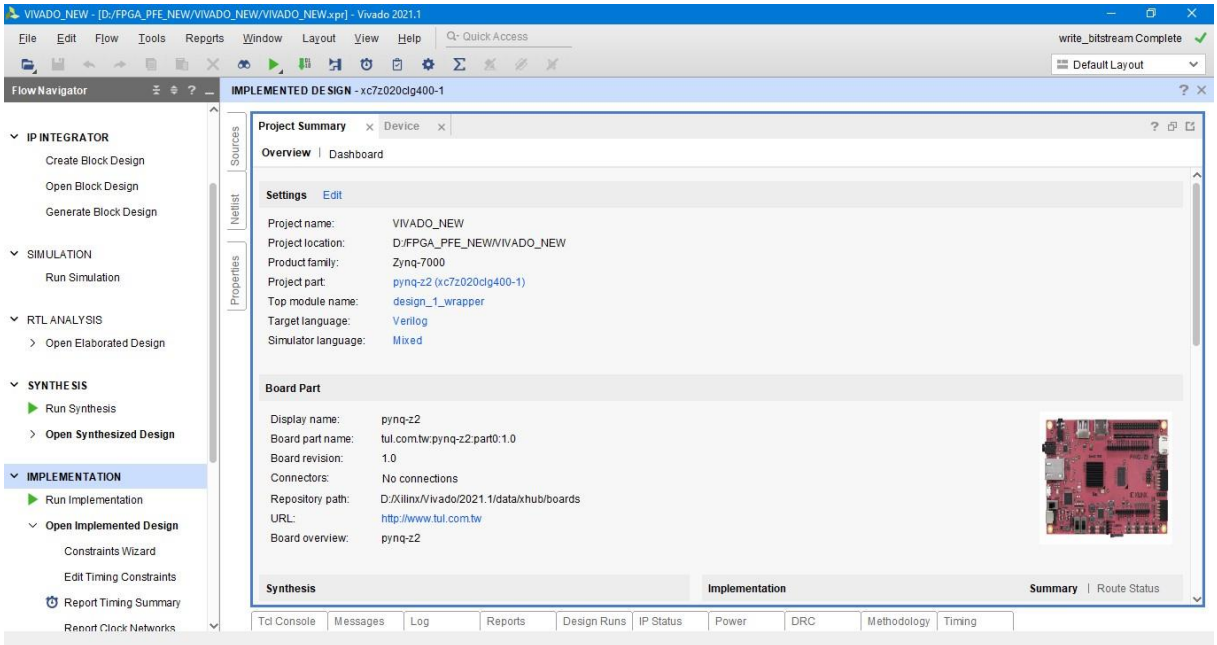


Figure 4.30 : *projet Vivado*

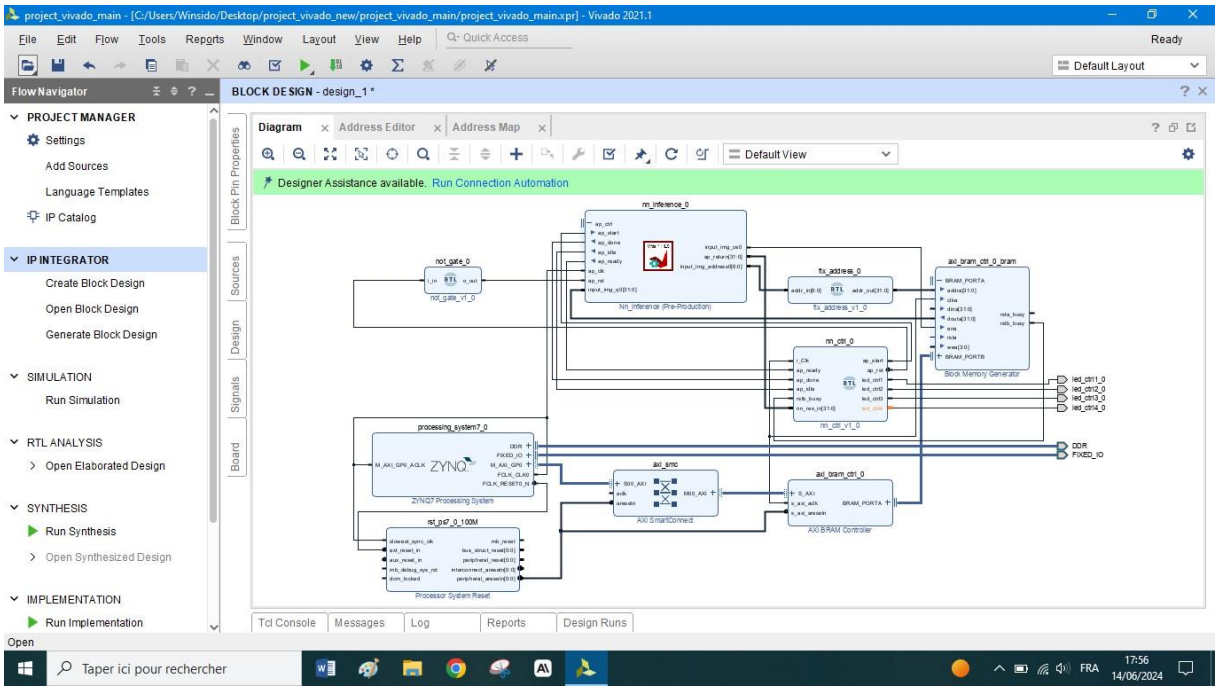


Figure 4.31 : *Vivado diagramme*

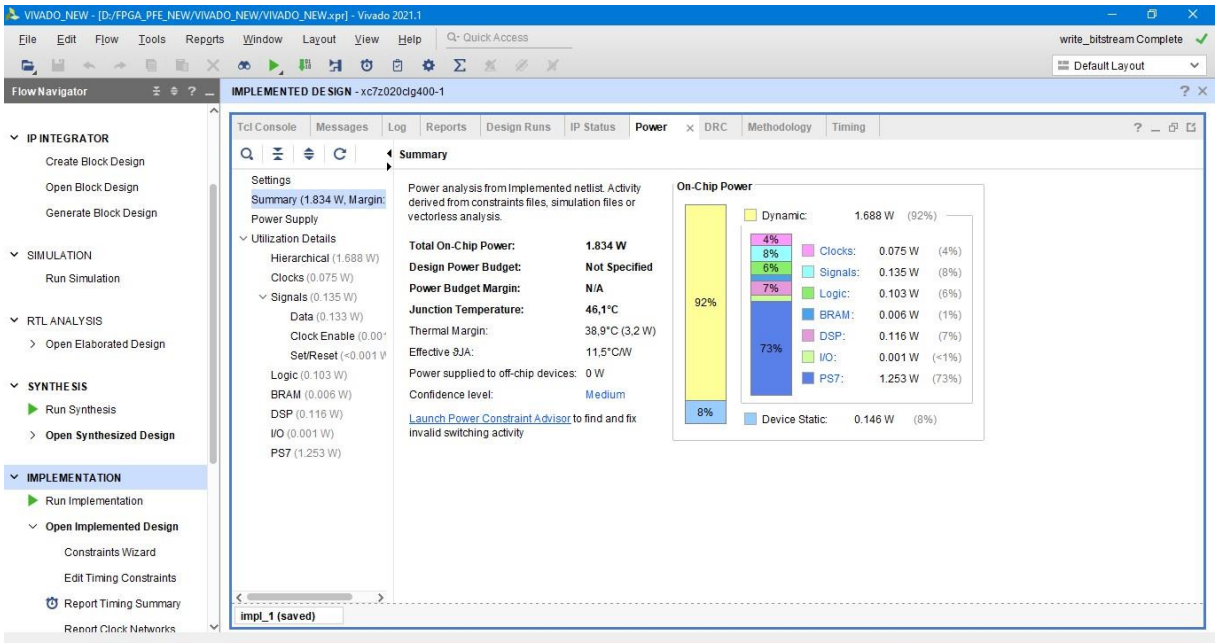


Figure 4.32 : *les résultats sur Vivado*

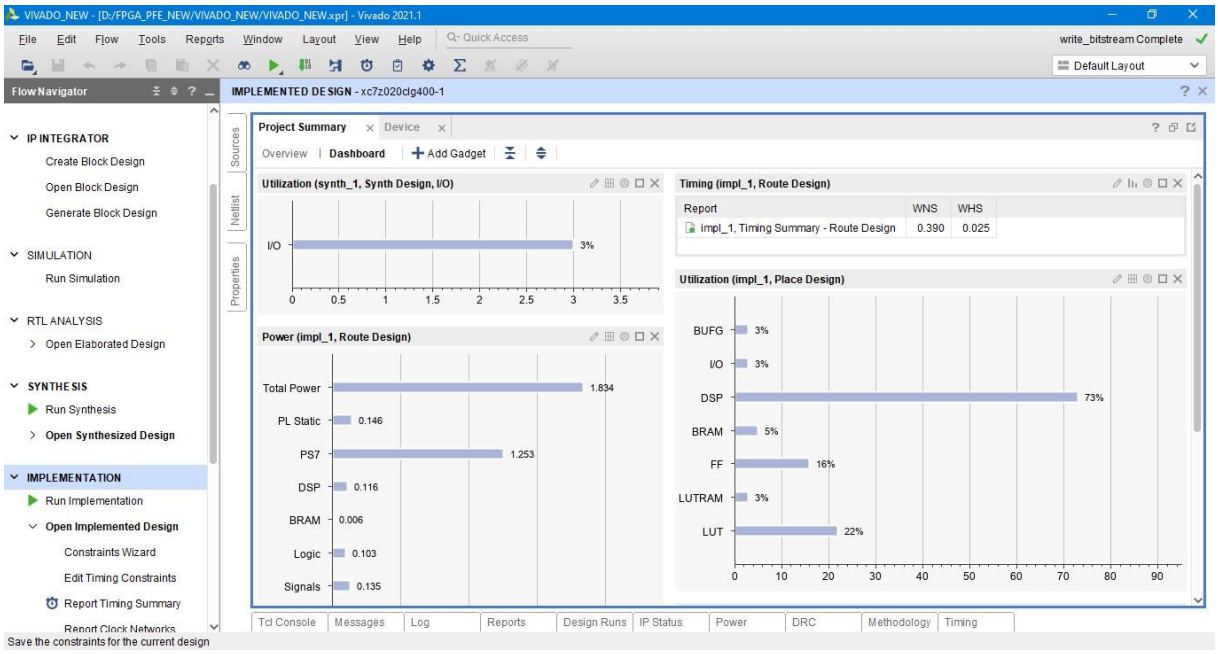


Figure 4.33 : *les résultats sur Vivado 1*

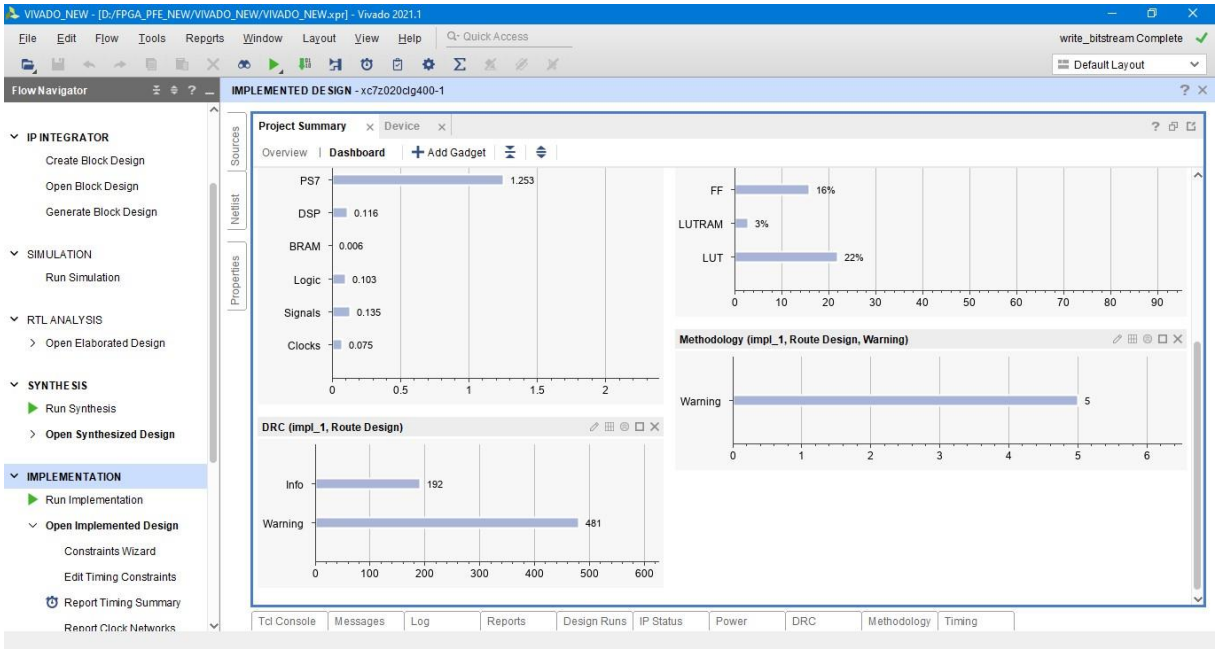


Figure 4.34 : *les résultats sur Vivado 2*

- **Créer un Projet Vitis :** nous avons développé un projet sur la plateforme Vitis et créer un projet d'application pour le déployer sur la carte Pynq-Z2.

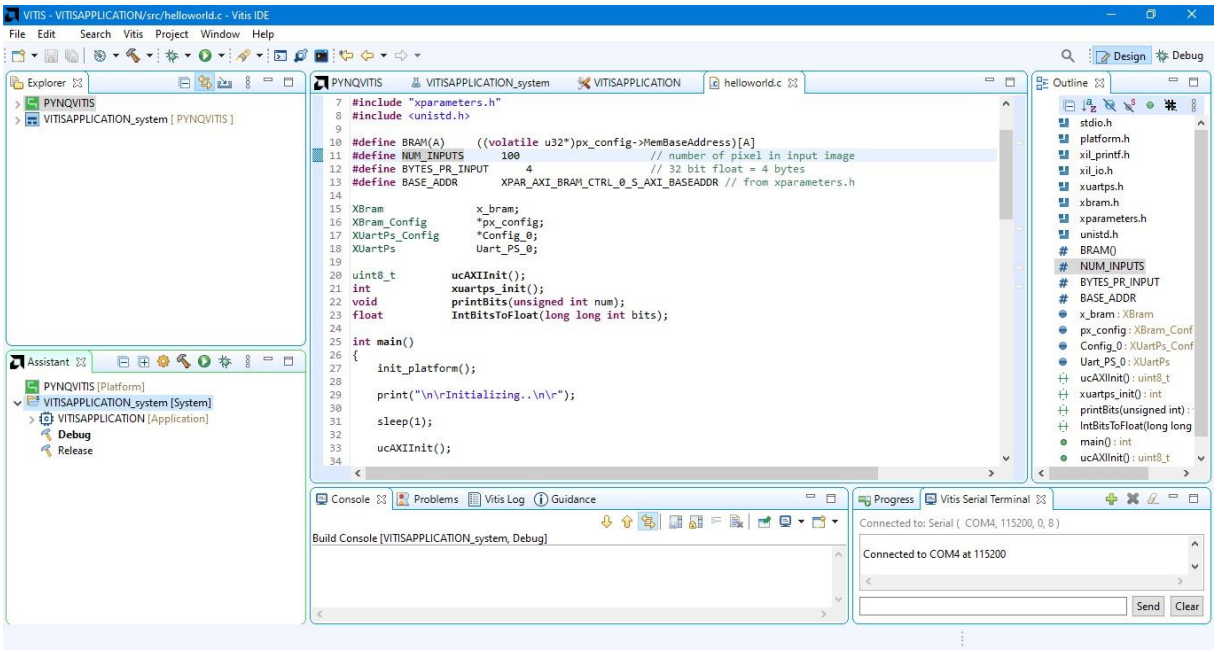


Figure 4.35 : *interface Vitis*

Nous avons configuré les port série a une vitesse de 115200 baud/s sur COM4 pour pouvoir déployer sur FPGA.

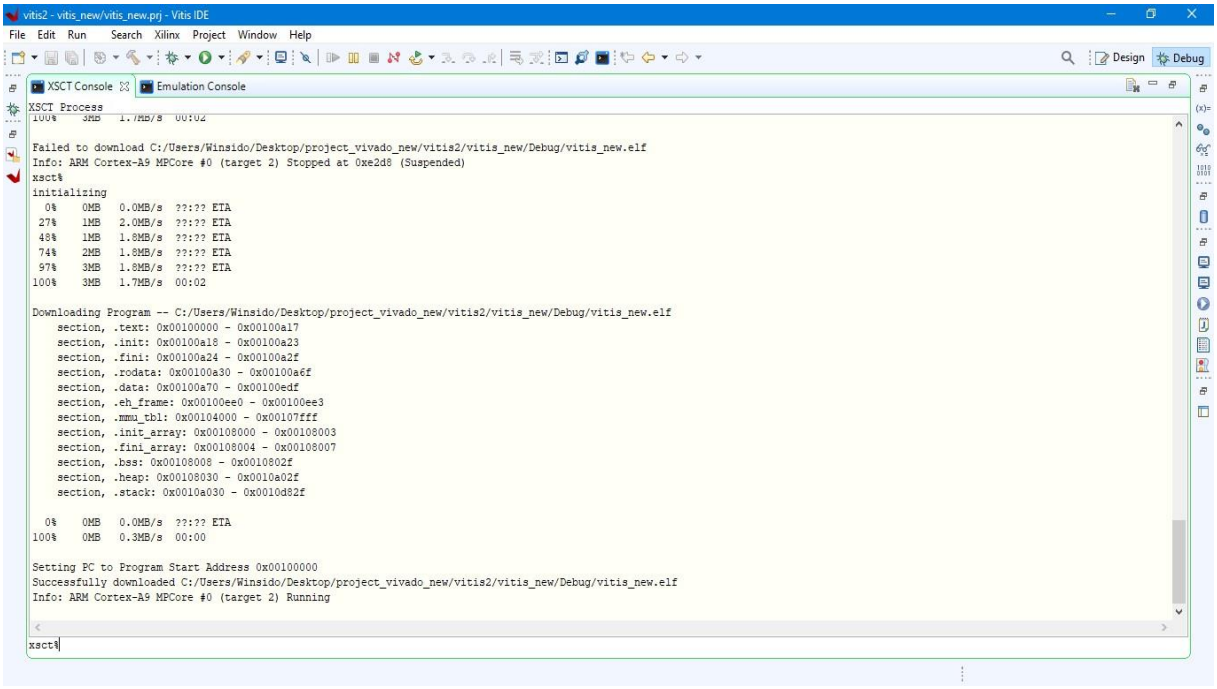


Figure 4.36 : *fin de déploiement*

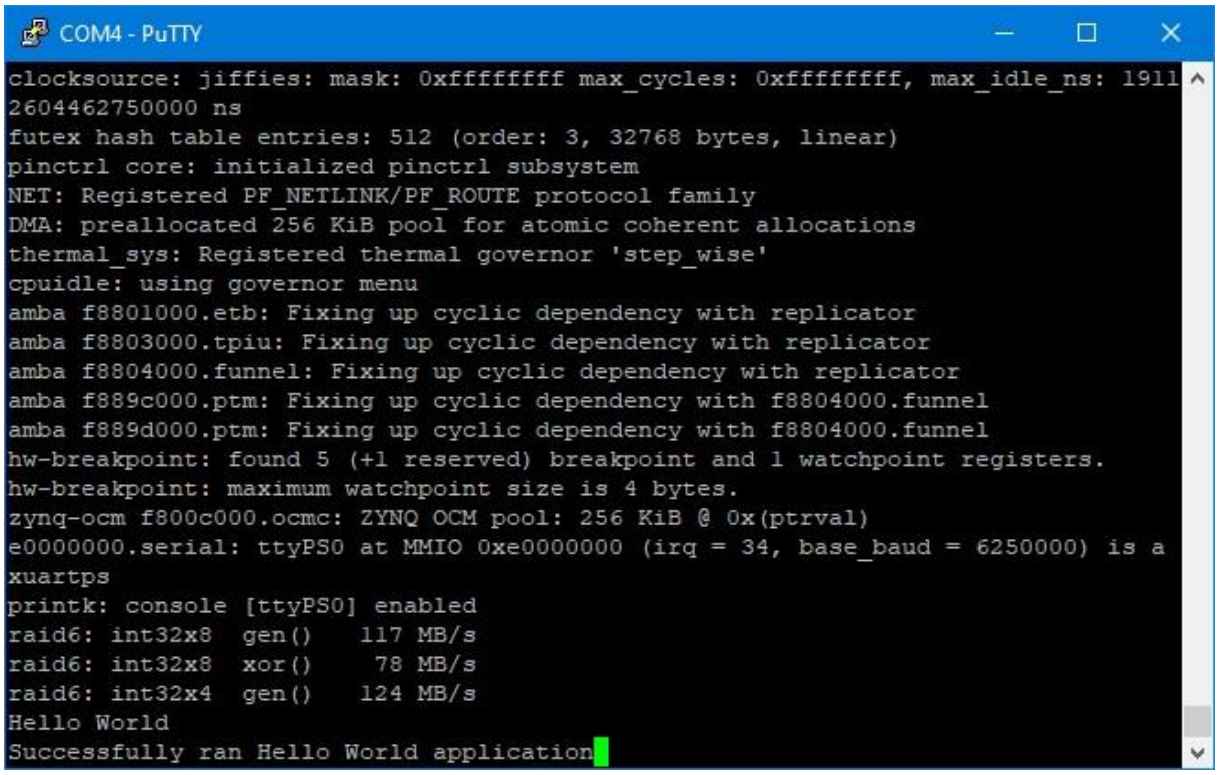


Figure 4.37 : *fin de déploiement 1*

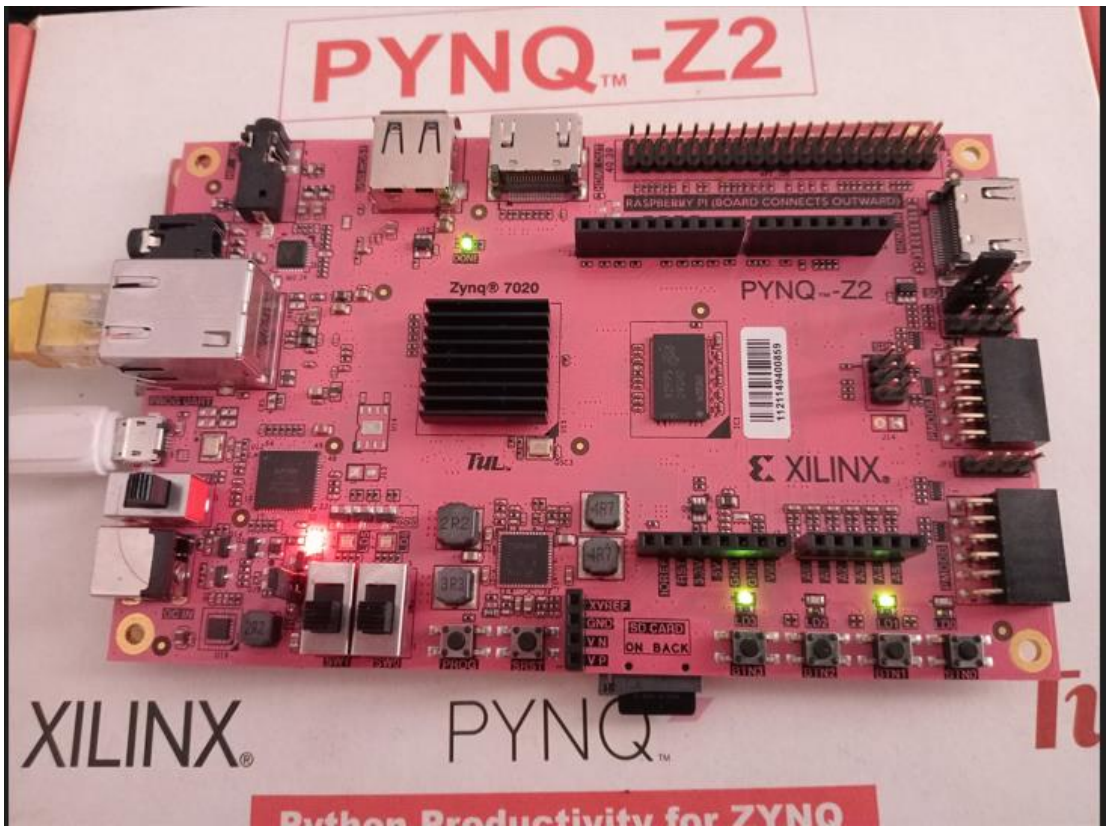


Figure 4.38 : *fin de déploiement 2*

- **Tester le Réseau Neuronal :** nous avons utilisé le code Python Test_FPGA.py pour tester le réseau neuronal implémenté sur FPGA. Pour ce faire, envoyez des images de test représentant les chiffres (0 à 9), générées aléatoirement depuis notre PC via UART, et observez les résultats affichés sur les LED et l'interface CMD PC.

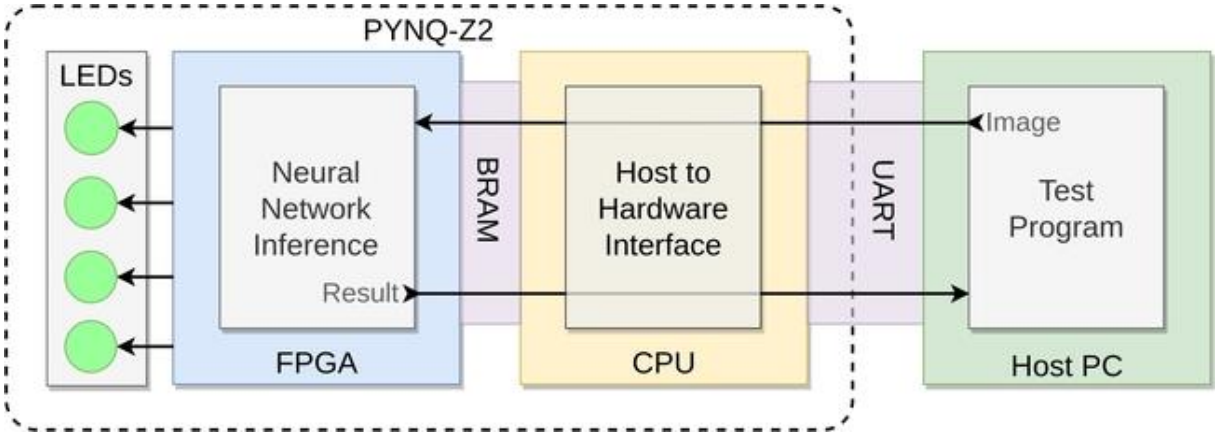


Figure 4.39 : *explication du protocole de test*

Voici les résultats de quelques essais :

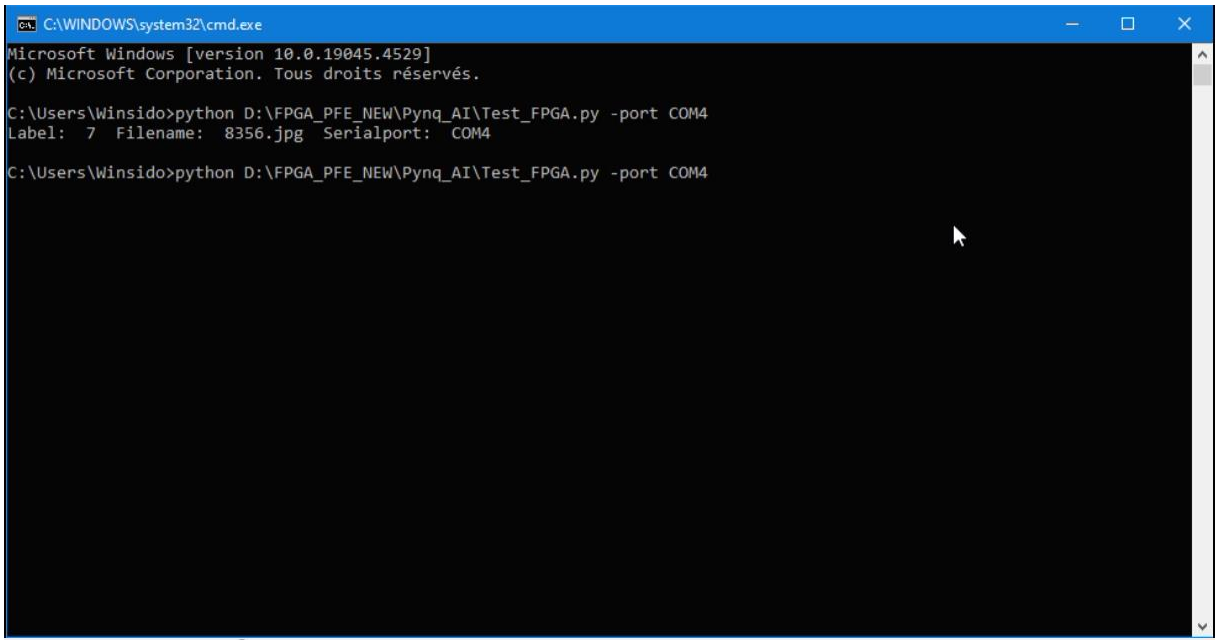


Figure 4.40 : *résultat sur pc pour numéro 7*

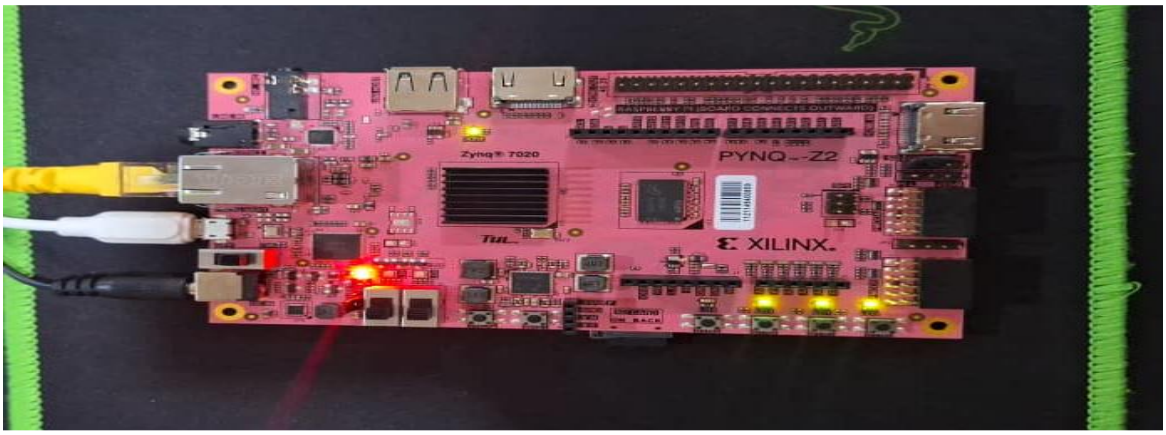


Figure 4.41 : *résultat sur PYNQ pour numéro 7*

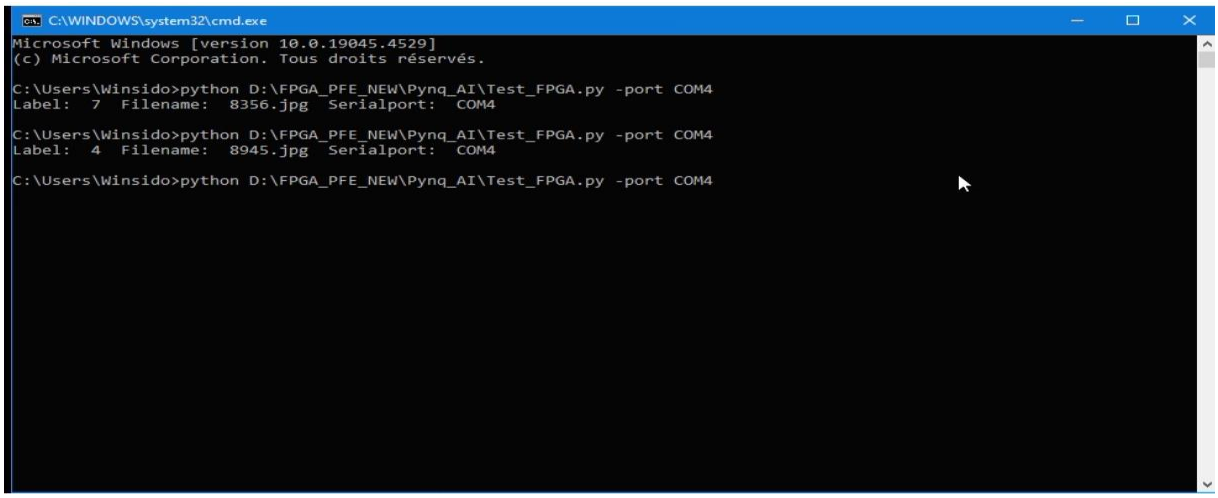


Figure 4.42 : *résultat sur pc pour numéro 4*

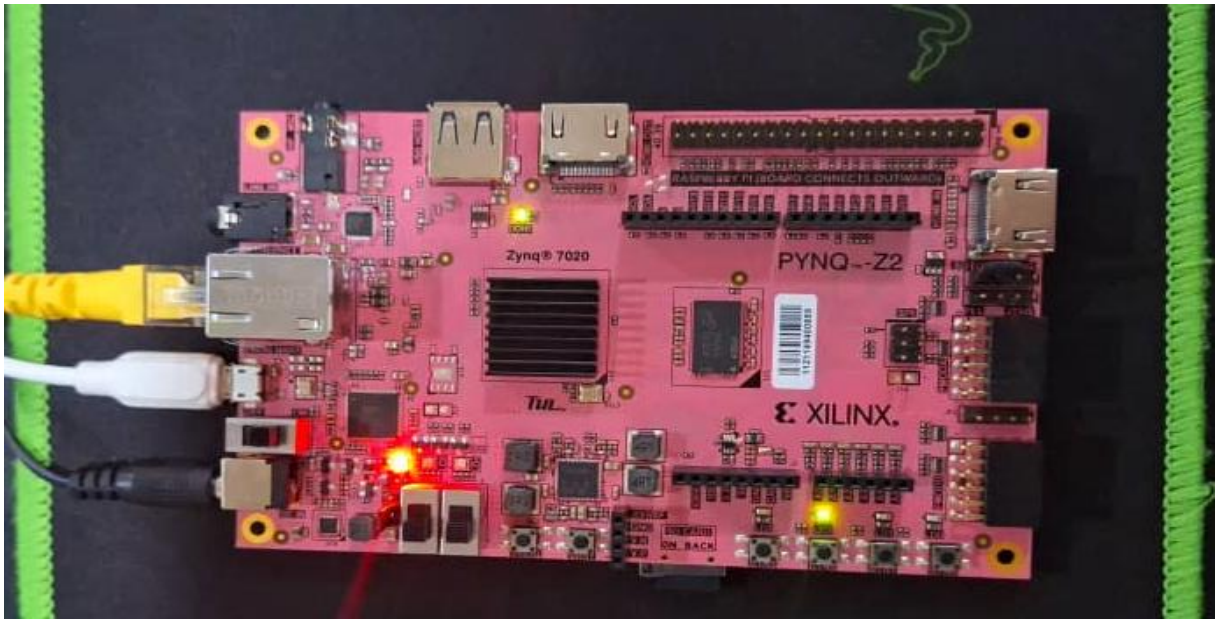


Figure 4.43 : *résultat sur PYNQ pour numéro 4*

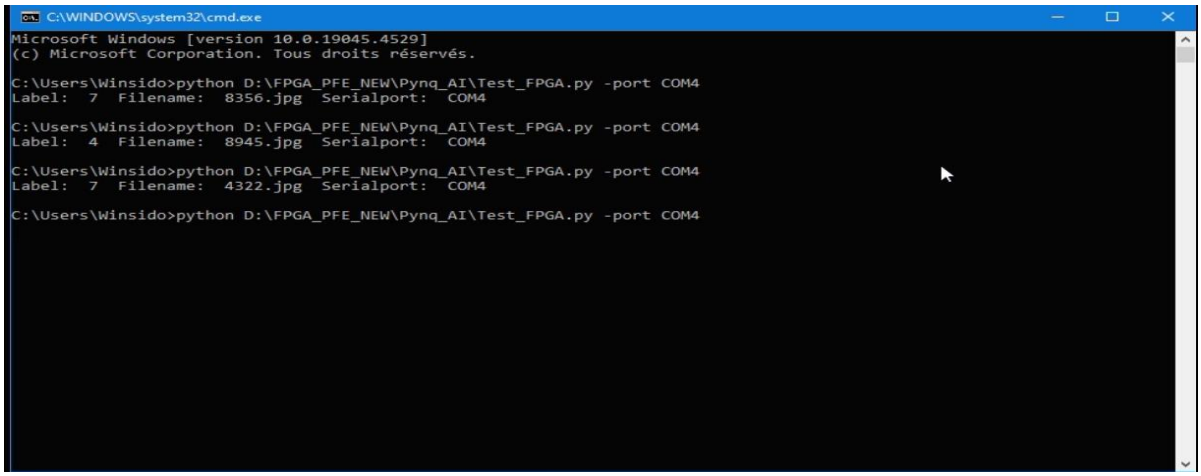


Figure 4.44 : *résultat sur pc pour numéro 7*

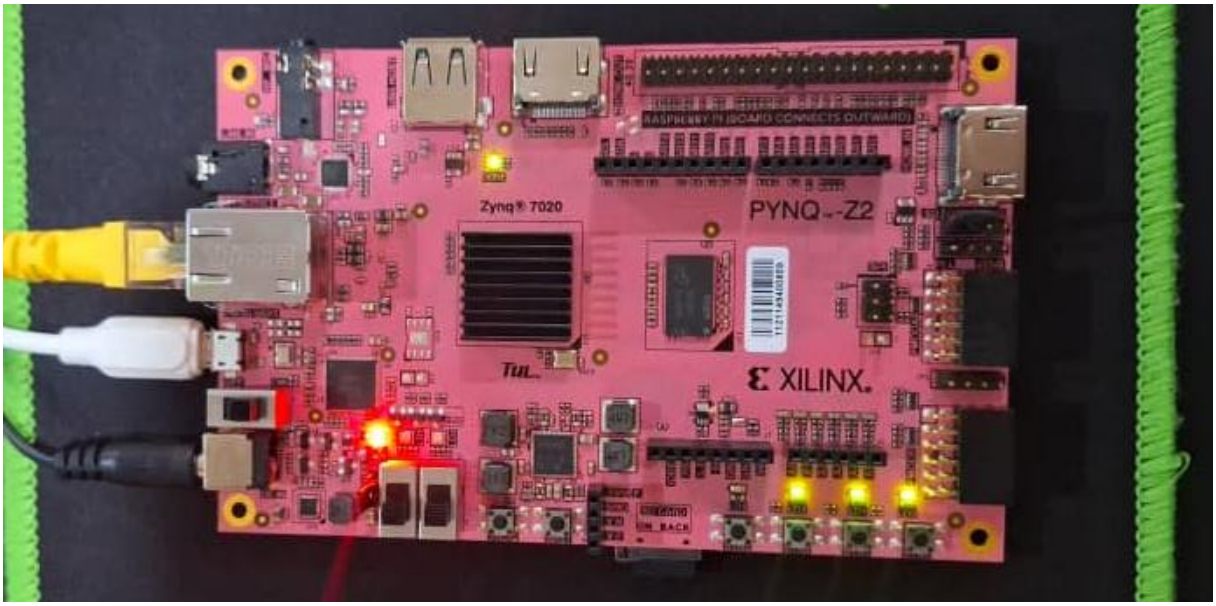


Figure 4.45 : *résultat sur PYNQ pour numéro 7*

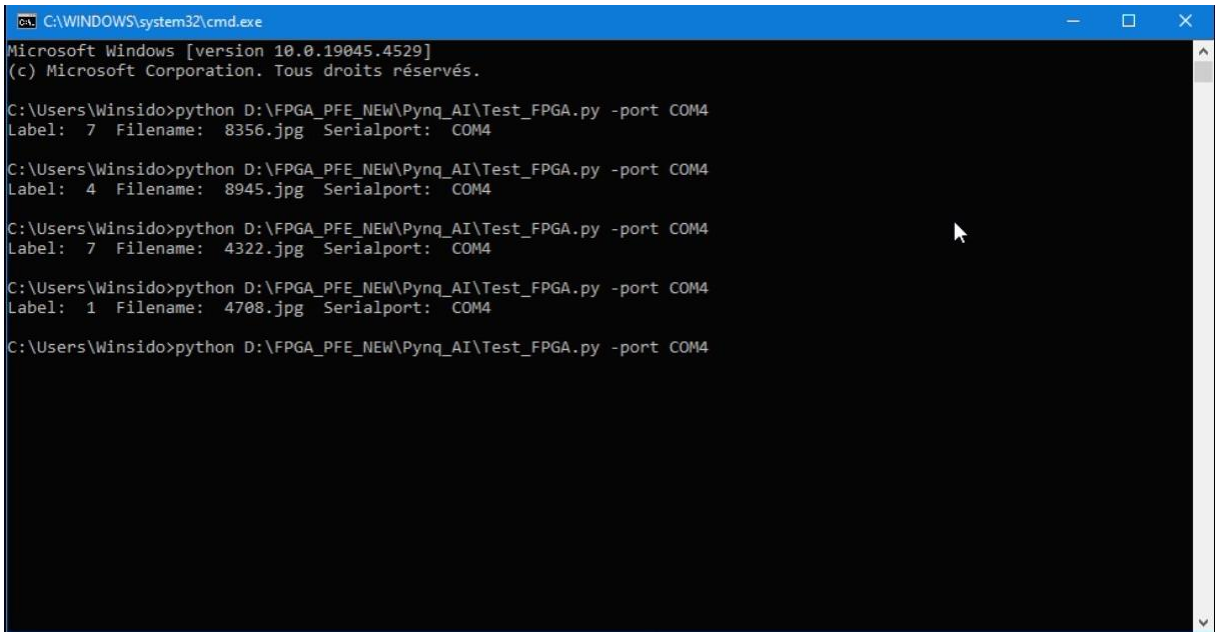


Figure 4.46 : *résultat sur pc pour numéro 1*

Figure 4.47 : *résultat sur PYNQ pour numéro 1*

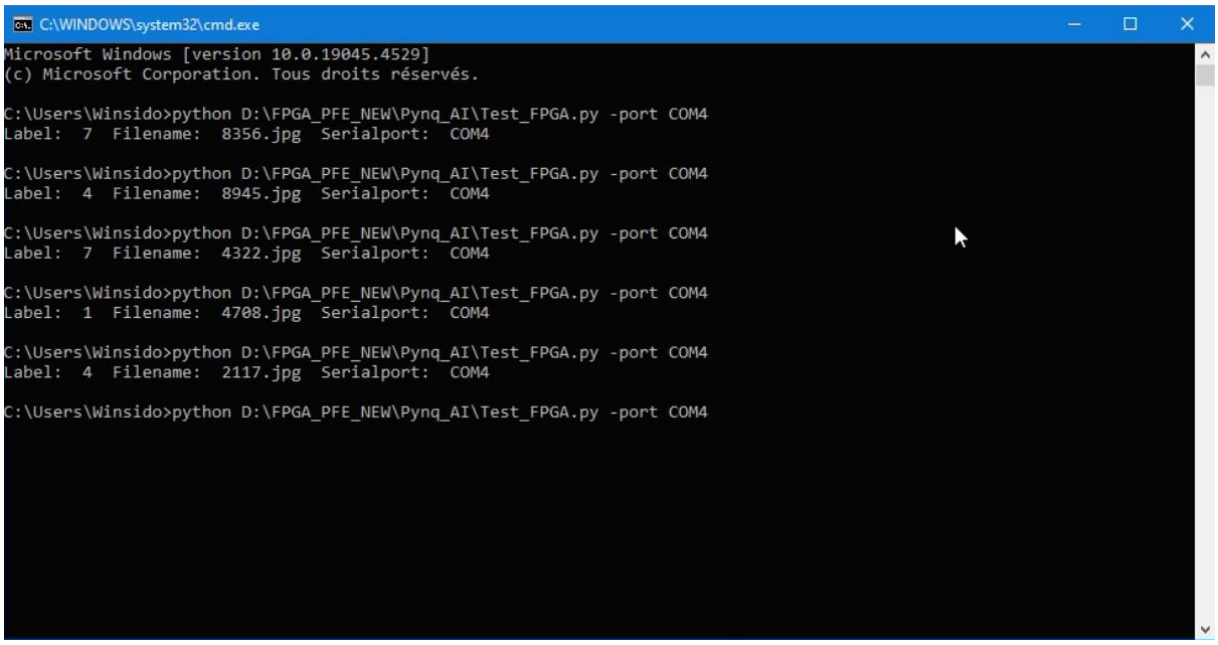


Figure 4.48 : *résultat sur pc pour numéro 4*

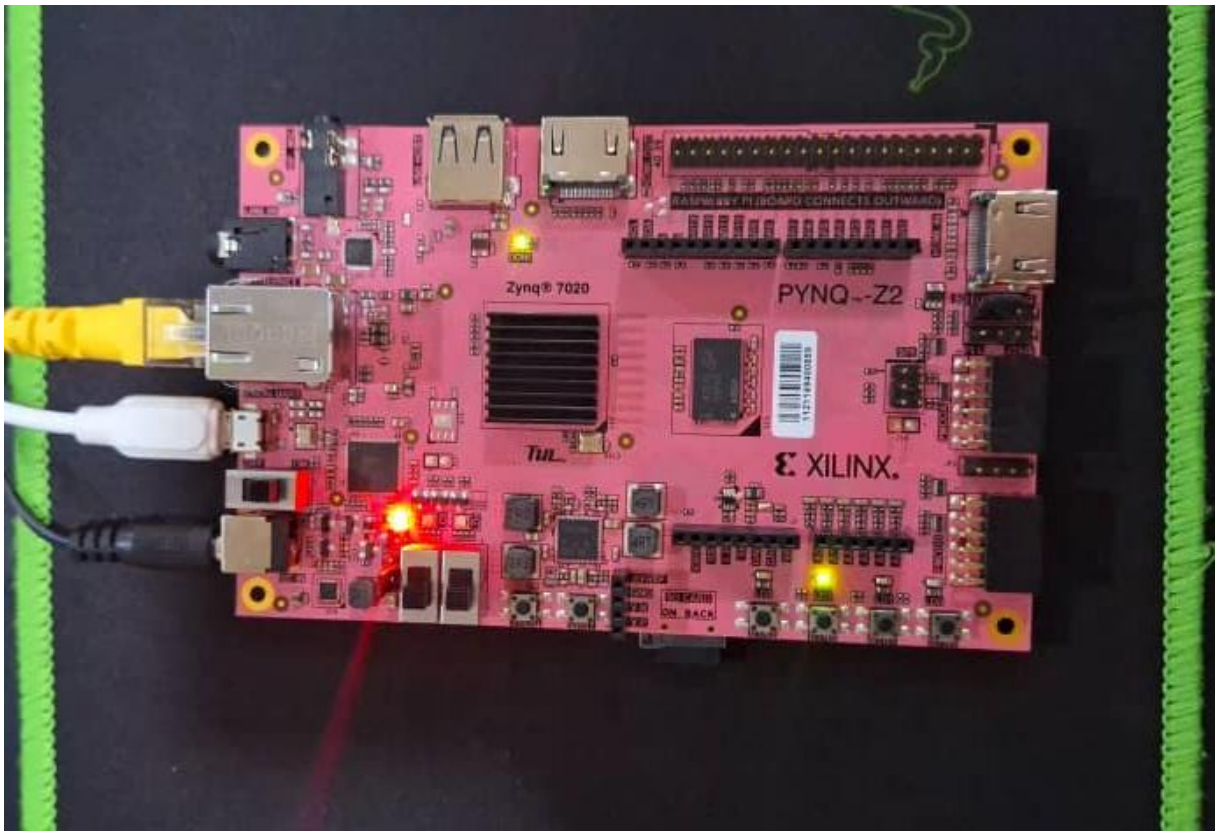


Figure 4.49 : *résultat sur PYNQ pour numéro 4*

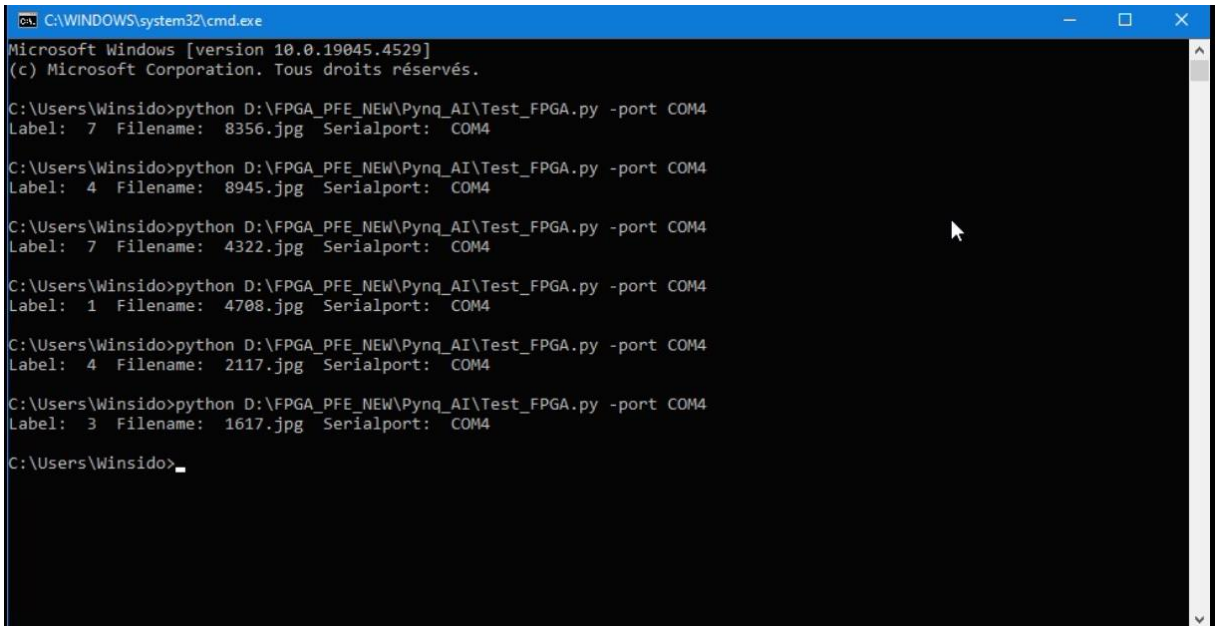


Figure 4.50 : *résultat sur PYNQ pour numéro 3*

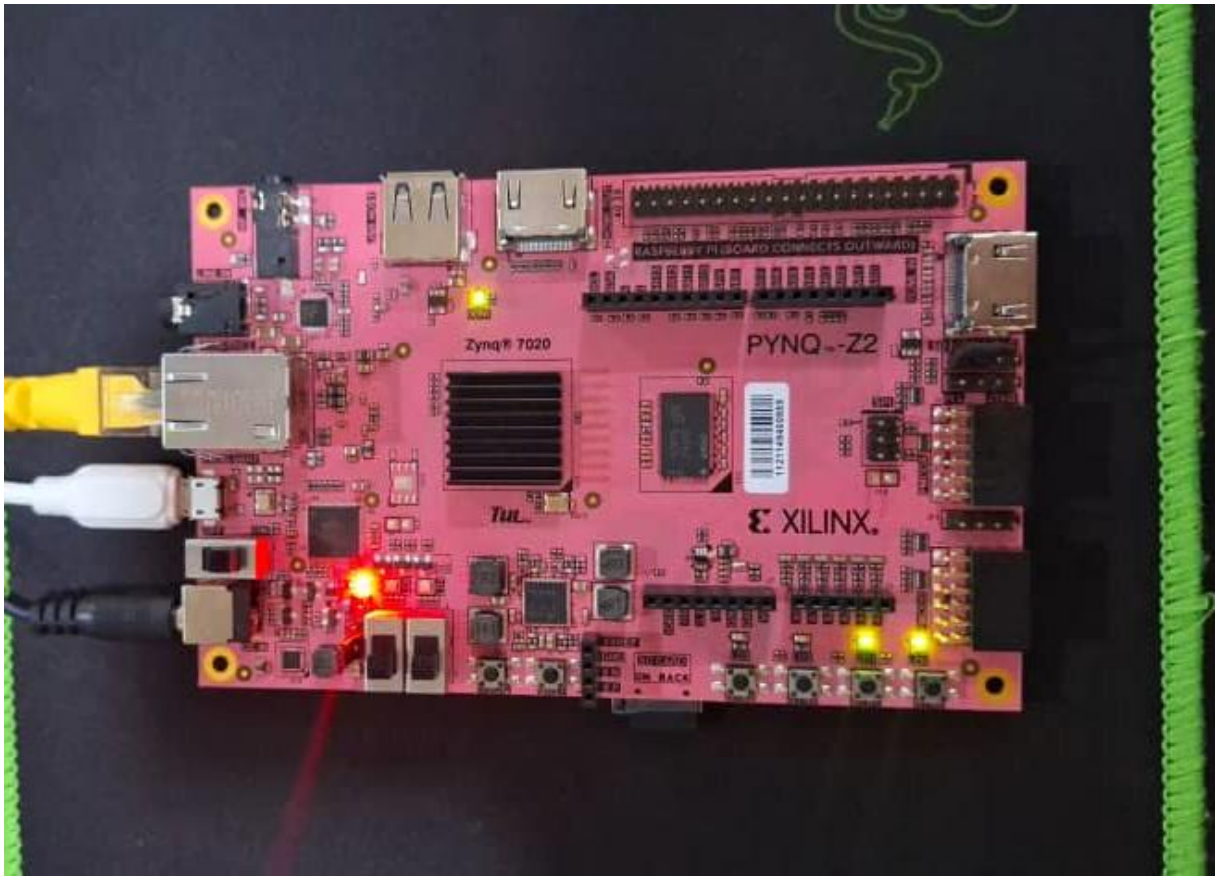


Figure 4.51 : *résultat sur PYNQ pour numéro 3*

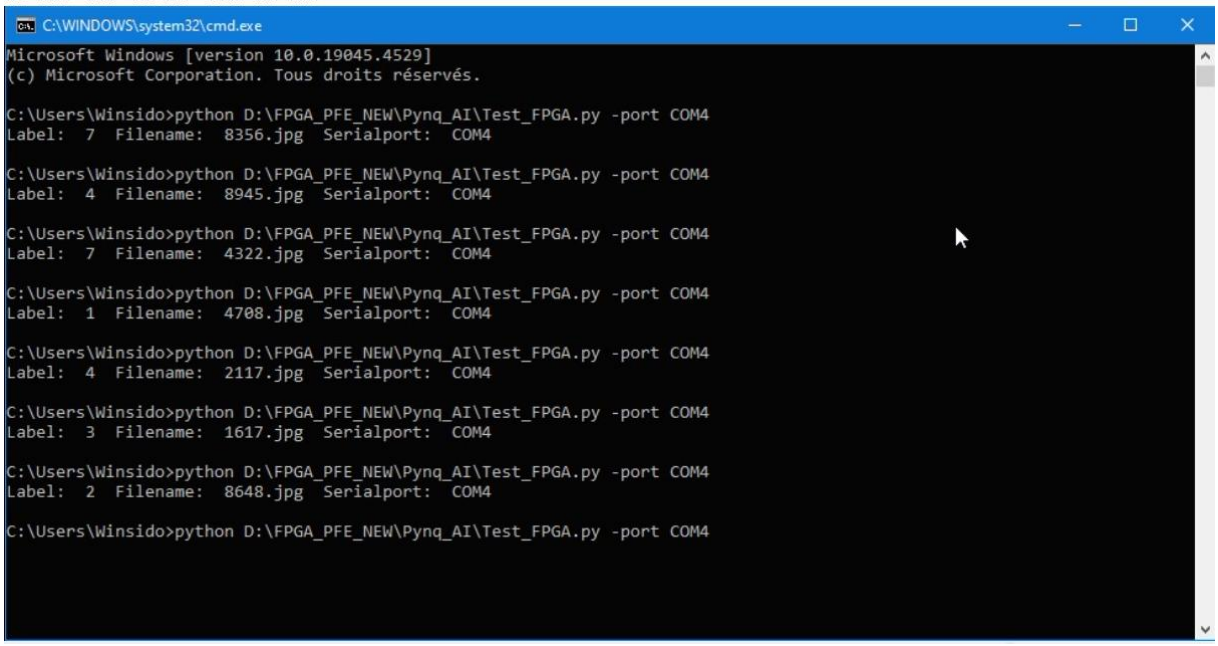


Figure 4.52 : *résultat sur PYNQ pour numéro 2*

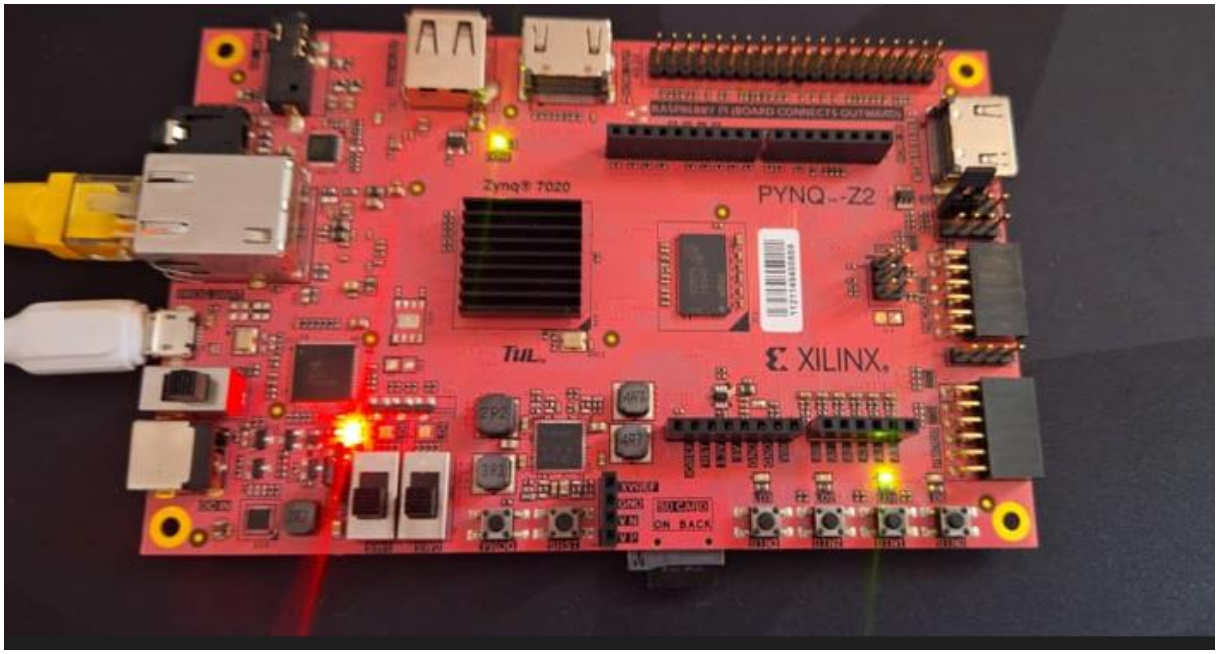


Figure 4.53 : *résultat sur PYNQ pour numéro 2*

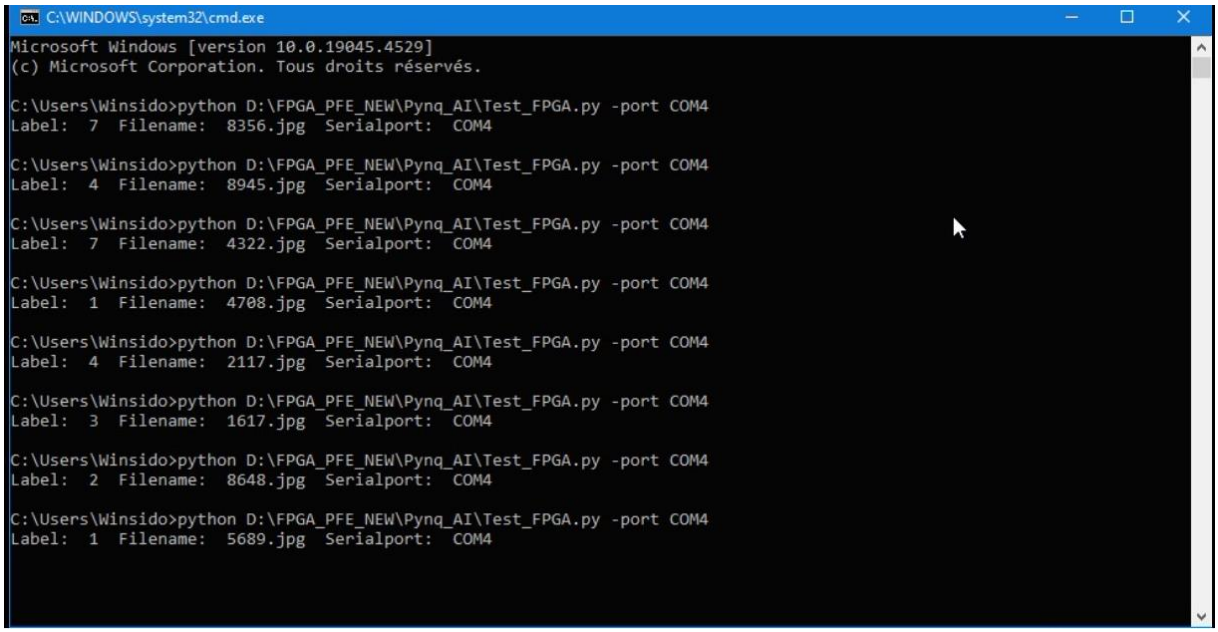


Figure 4.54 : *résultat sur pc pour numéro 1*

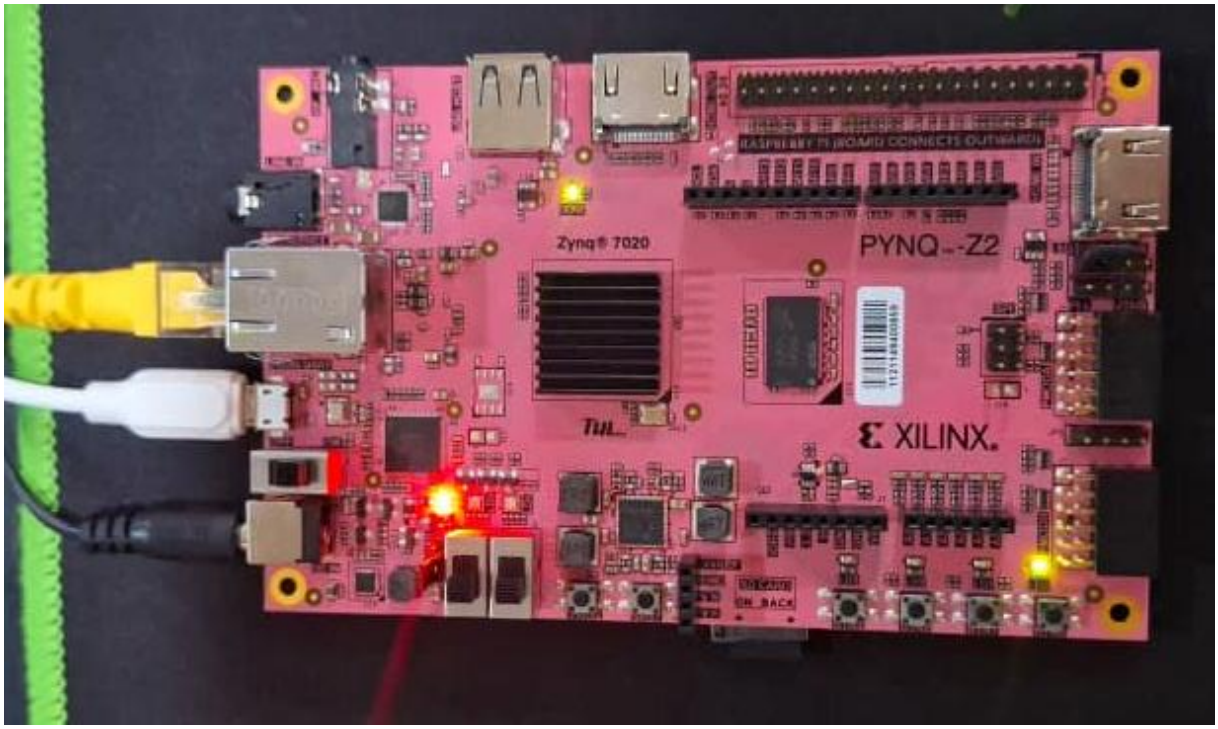


Figure 4.55 : *résultat sur PYNQ pour numéro 1*

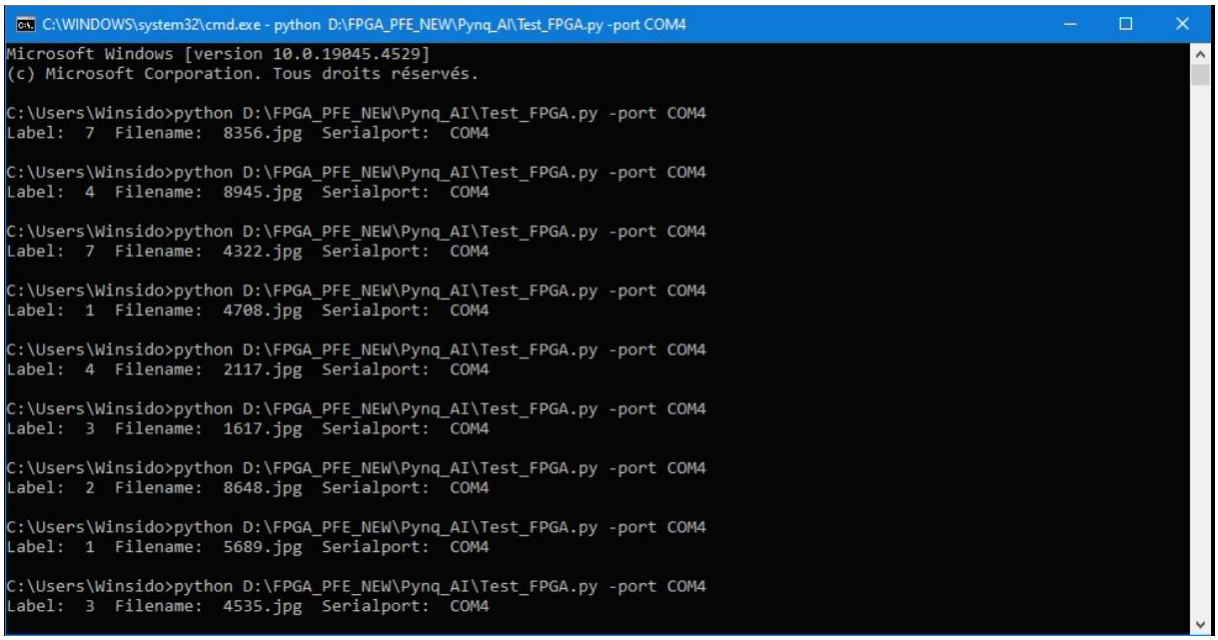


Figure 4.56 : *résultat sur pc pour numéro 3*

Figure 4.57 : *résultat sur PYNQ pour numéro 3*

- **Essai d'Implémentation de U-net pour la segmentation de l'arbre rétinien :**

On premier lieu nous avons essai d'implémenter sur Jupyter notre code python.

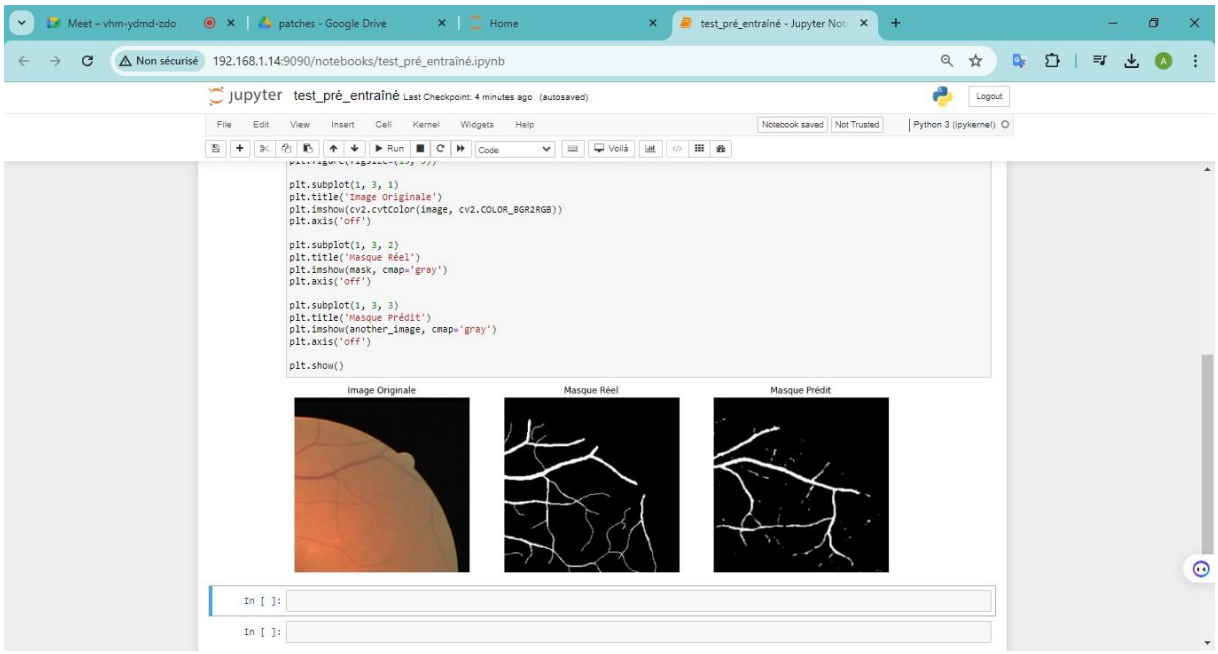


Figure 4.58 : *code d'implémentation sur Jupyter*

Nous avons effectué le test sur une carte PYNQ Z2 avec une seule image et un masque, et cela a été réalisable.

Cependant, lorsque nous avons essayé d'implémenter toute notre base de données, la carte PYNQ Z2 a cessé de fonctionner. Malgré nos efforts pour alléger le code et les bibliothèques en installant des versions légères comme TensorFlow Lite avec succès, mais le problème toujours persiste.

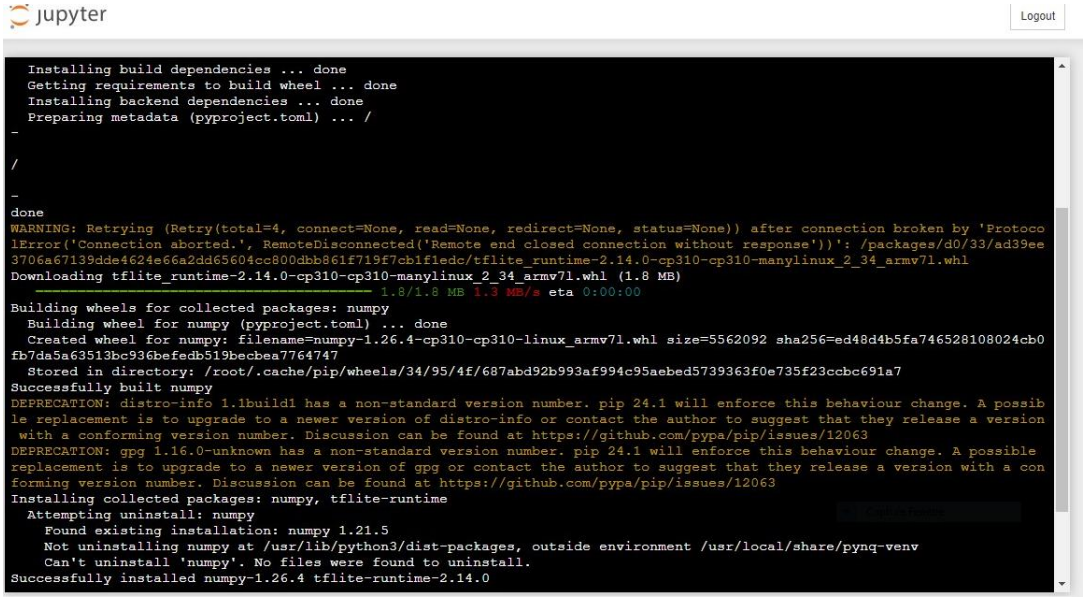


Figure 4.59 : *installation Tensorflow lite*

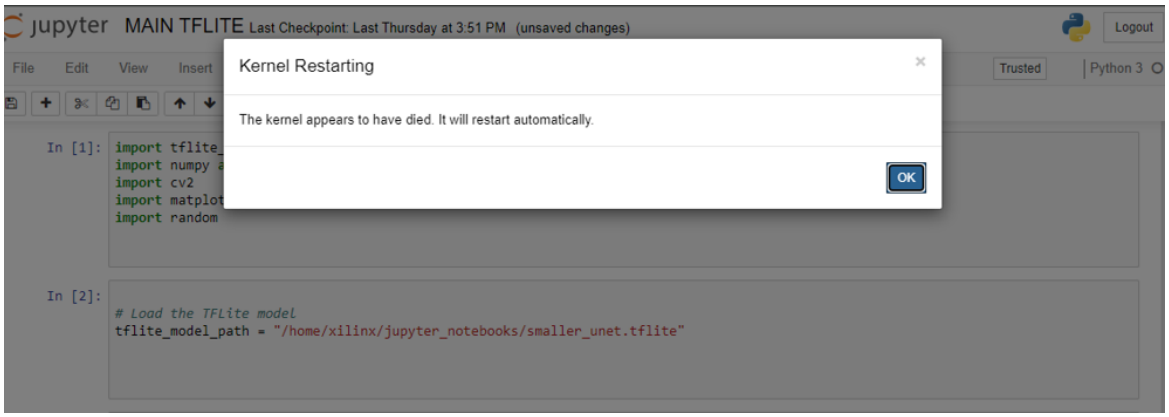


Figure 4.60 : *message d'erreur*

Après plusieurs tentatives infructueuses, nous en avons conclu qu'il nous faut une carte plus puissante pour pouvoir traiter toute la base de données de manière efficace.

IV.7. Conclusion

En conclusion, ce mini-projet démontre avec succès l'implémentation d'un modèle de réseau de neurones convolutifs (CNN) pré-entraîné pour la reconnaissance de chiffres manuscrits en utilisant la base de données MNIST sur une carte FPGA PYNQ Z2 de Xilinx. En exploitant les capacités de traitement parallèle de la FPGA, nous avons efficacement validé l'utilisation d'un CNN pour la classification d'images, avec des résultats affichés à la fois sur un ordinateur et sur les LED intégrées de la carte PYNQ Z2 sous forme binaire. Cette approche met en évidence non seulement la puissance des FPGA dans le domaine de l'apprentissage automatique, mais aussi leur potentiel pour des solutions embarquées nécessitant des performances rapides et précises.

Pour aller plus loin dans notre projet initial, l'implémentation de U-net pour la segmentation de l'arbre rétinien nécessiterait une carte FPGA plus avancée, comme celles de la série Xilinx Zynq UltraScale+ ou Versal AI Core Series, spécialement conçues pour le traitement vidéo et les communications. Ces plateformes permettraient une exploitation optimale des capacités requises pour une segmentation précise et rapide des données d'imagerie médicale complexes. Cela ouvrirait la voie à des applications innovantes dans les domaines de la santé et de la médecine.

Cette évolution vers une FPGA plus puissante assurerait une gestion plus efficace des ressources et une exécution plus rapide des tâches complexes, répondant ainsi aux exigences croissantes en matière de traitement d'images et de calcul intensif pour les applications embarquées avancées.

Conclusion Générale

Conclusion Générale

En conclusion, ce mémoire a exploré de manière approfondie l'application de l'architecture U-Net pour la segmentation de l'arbre rétinien, en mettant en lumière son rôle crucial dans l'amélioration du diagnostic et de la gestion de la rétinopathie diabétique. À travers l'intégration de techniques avancées de Deep Learning et l'utilisation de bibliothèques tels que TensorFlow et Keras, nous avons développé des modèles précis et efficaces pour analyser les images rétiniennes et identifier avec précision les structures anatomiques critiques.

L'étude a mis en évidence les avantages significatifs de l'approche basée sur l'IA dans le domaine de l'ophtalmologie, notamment en permettant une détection précoce des anomalies rétiniennes et une évaluation objective des progrès de la maladie. En améliorant la précision diagnostique, ces avancées technologiques offrent des perspectives prometteuses pour personnaliser les traitements et améliorer les résultats cliniques des patients diabétiques.

Par ailleurs, ce projet a souligné l'importance de l'optimisation continue des modèles d'IA, en explorant des stratégies telles que le prétraitement des données, l'augmentation des données et l'ajustement des hyper paramètres pour maximiser la performance des algorithmes de segmentation. Ces efforts ont été essentiels pour garantir la fiabilité et la généralisation des modèles dans des environnements cliniques réels.

Enfin, cette recherche a ouvert la voie à de futures explorations dans le domaine de l'IA et de l'imagerie médicale, encourageant le développement de nouvelles méthodes et technologies pour relever les défis complexes posés par les maladies rétiniennes. En continuant à intégrer les avancées technologiques et les connaissances biomédicales, nous pouvons aspirer à transformer les soins ophtalmologiques et à améliorer la qualité de vie des patients à travers le monde.

Références Bibliographiques

Références Bibliographiques

[1] https://www.institut-vision.org/comprendre-la-vision

[2] https://www.futura-sciences.com/sante/definitions/corps-humain-retine-4322/

[3] https://www.futura-sciences.com/sante/dossiers/medecine-oeil-vision-dela-vision-667/page/5/

[4] https://fr.wikipedia.org/wiki/%C5%92il_humain#Anatomie_et_physiologie_de_l'%C5%93il_humain

[5] https://hal.science/hal-03363238/document

[6] Dupont, J., & Martin, M. (2020). *Retinal Pathology: Definition and Associated Pathologies*. In **Clinical Ophthalmology Manual** (3rd ed., pp. 45-47). International Medical Editions.

[7] https://icrcat.com/fr/traitements-et-tests-de-diagnostic/photocoagulation-au-laser/

[8] https://hal.science/hal-02960791

[9] https://fr.wikipedia.org/wiki/Classification

[10] https://www.ibm.com/fr-fr/topics/image-segmentation

[11] https://docs.clarifai.com/tutorials/image-classification-detection-segmentation/

[12] https://fr.wikipedia.org/wiki/Traitement_d%27images

[13] https://di.univ-blida.dz/jspui/handle/123456789/12630

[14] https://hal.science/hal-02960791

[15] https://www.ibm.com/fr-fr/topics/neural-networks

[16] https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif

[17] https://paperswithcode.com/method/max-pooling

[18] https://en.wikipedia.org/wiki/Batch_normalization

[19] https://medium.com/@vaibhav1403/fully-connected-layer-f13275337c7c

[20] https://blent.ai/blog/a/cnn-comment-ca-marche

[21] https://paperswithcode.com/dataset/drive

[22] Luis Perez, Jason Wang - The Effectiveness of Data Augmentation in Image

[23] Classification using Deep Learning – publié le 13 Décembre 2017

[24] Anubhav Tewari Architecture U-Net (opengenus.org)

[25] https://fr.wikipedia.org/wiki/Apprentissage_supervis%C3%A9

[26] Jason Brownlee - Understand the Impact of Learning Rate on Neural Network

[27] Performance – publié le 25 Janvier 2019 in Deep Learning Performanc

[28] https://deepai.org/machine-learning-glossary-and-terms/epoch

[29] Qiang Yang, Sinno Jialin Pan, "A Survey on Transfer Learning", IEEE Transactions on

[30] Knowledge & Data Engineering, vol. 22, no. , pp. 1345–1359, October 2010,

[31] doi:10.1109/TKDE.2009.191

[32] https://2btrading.tn/accueil/9296-carte-de-developpement-pynq-z2.html

[33] https://pynq.tue.nl/general/pynq/

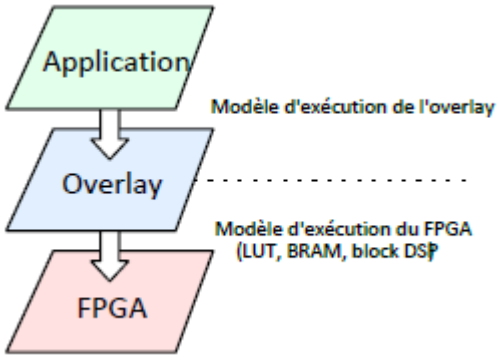
[34] https://win32diskimager.download/

[35] http://www.pynq.io/board.html

Annexes

Les overlays

Le concept d'overlay est relativement récent, et il n'a pas encore de définition précise et arrêtée. La définition la plus générique d'un overlay que l'on puisse donner est la suivante : Un overlay est un design reconfigurable implémenté sur FPGA (qui est lui-même reconfigurable). En d'autres termes, un overlay est une couche d'abstraction matérielle placée entre l'application le FPGA hôte, isolant celle-ci de ce dernier.

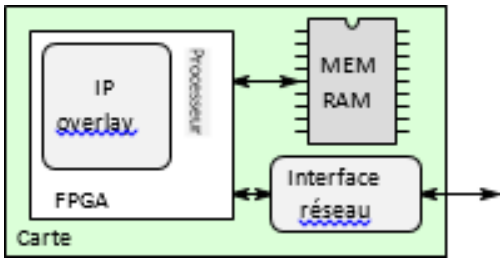


L'overlay isole l'application du FPGA sous-jacent. Il apporte son propre modèle d'exécution.

Comme illustré ci-dessus, l'overlay apporte son propre modèle d'exécution, qui peut être radicalement différent de celui du FPGA sous-jacent. L'application ne voit que le modèle d'exécution de l'overlay et est isolée du FPGA hôte. Le modèle d'exécution de l'overlay détermine sa granularité. Les overlays permettent d'explorer l'espace entre la facilité d'usage des processeurs et les performances des FPGA. Les overlays sont donc de bons candidats pourêtre utilisés en tant qu'accélérateurs de calcul reconfigurables.

Intégration des overlays

La gestion bas niveau de l'overlay est réalisée aux niveaux matériel et logiciel : un ensemble de contrôleurs matériels vient s'interfacer avec l'overlay pour permettre sa gestion par un logiciel que nous appelons hyperviseur. L'overlay et ses contrôleurs matériels sont rassemblés en une IP. L'hyperviseur étant logiciel, un processeur est nécessaire pour l'exécuter.

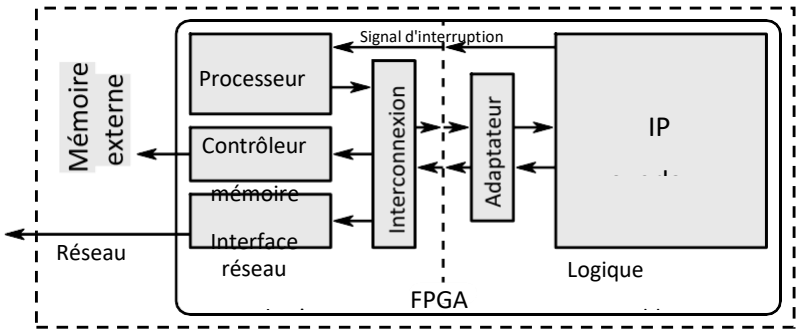


Exemple d'intégration de l'overlay, l'hyperviseur est exécuté sur le processeur dans le FPGA

Dans le cas de la carte Pynq Z2, le FPGA est utilisé seul c'est à dire en mode "standalone", c'est-à-dire qu'il est utilisé seul sans processeur et qu'il est directement relié au réseau. Dans ce cas, l'hyperviseur doit être exécuté sur le FPGA sur un processeur softcore implémenté via les ressources reconfigurables du FPGA. Ce cas est illustré figure ci-dessus.

Intégration de l'IP overlay avec processeur physique

Lorsqu'un processeur physique est présent dans la plateforme, il peut être soit externe au FPGA, soit embarqué de manière fixe dans le FPGA avec un ensemble de périphériques. Si le processeur est embarqué dans le FPGA (cas de la carte Pynq Z2), comme illustré figure 4.6, alors il a un accès direct à la matrice reconfigurable du FPGA, et l'intégration de l'IP overlay ne demande qu'un adaptateur entre les interfaces Wishbone maître et esclave de l'IP et les ports de connexions à la partie fixe, ainsi que de connecter le signal d'interruption généré par l'IP au contrôleur d'interruption du processeur dans la partie fixe. Par exemple, dans le cas de la famille de FPGA Zynq de Xilinx d'où la carte PYNQ Z2, l'intégration de l'IP demande un adaptateur Wishbone → AXI et un adaptateur AXI → Wishbone. Le processeur peut ainsi avoir accès aux registres de configuration de l'IP, et l'IP et le processeur partagent l'accès à une même mémoire RAM.



Intégration de l'overlay dans un FPGA comportant un processeur implémenté sur le silicium : cas de la carte PYNQ Z2.

Overlay de base

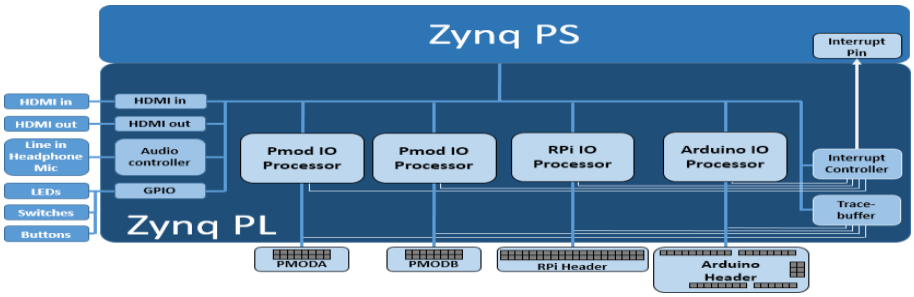
Le but de la conception de l'overlay de base est de permettre à PYNQ d'utiliser des périphériques sur une carte prête à l'emploi. La conception inclut une adresse IP matérielle pour

Contrôler les périphériques sur la carte cible et connecte ces blocs IP au Zynq PS. Si l'overlay de base est disponible pour une carte, les périphériques peuvent être utilisés à partir de l'environnement Python immédiatement après le démarrage du système.

Les périphériques de la carte incluent généralement des périphériques GPIO (voyants, commutateurs, boutons), vidéo, audio et autres interfaces personnalisées.

Comme l'overlay de base inclut IP pour les périphériques sur une carte, elle peut également être utilisée comme une référence pour créer de nouvelles overlay personnalisées.

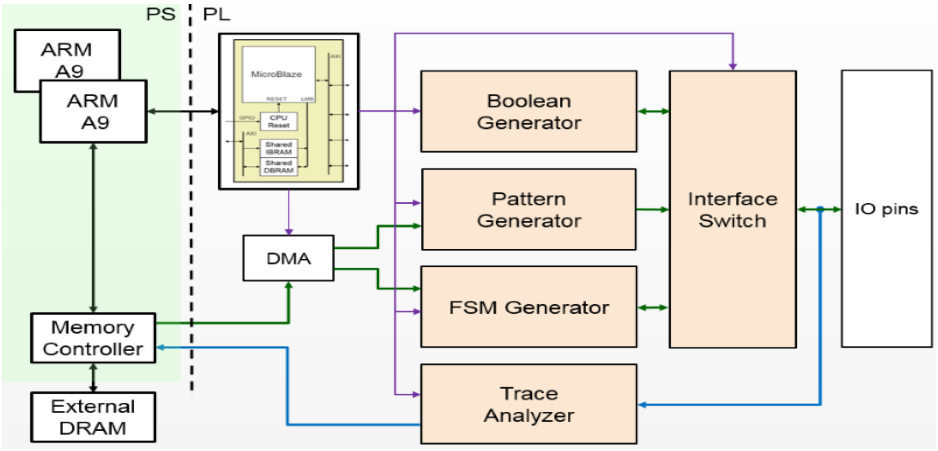
Dans le cas des interfaces à usage général, par exemple les interfaces Pmod ou Arduino, l'overlay de base peut inclure un PYNQ MicroBlaze. Un PYNQ MicroBlaze permet de contrôler des appareils avec différentes interfaces et protocoles sur le même port sans nécessiter de modification de la conception de la logique programmable.



Overlay de base

Overlay logictools

L'overlay logictools se compose de blocs matériels programmables à connecter à des circuits logiques numériques externes. Des machines à états finis, des fonctions logiques booléennes et des modèles numériques peuvent être générés à partir de Python. Un commutateur programmable connecte les entrées et les sorties des blocs matériels aux broches d'E / S externes. L'overlay logictools peut également avoir un analyseur de trace pour capturer les données de l'interface IO pour l'analyse et le débogage.



Overlay logictool