

UNIVERSITÉ MOHAMMED V DE RABAT
Faculté des Sciences



Université Mohammed V
Faculté des Sciences
Rabat

Département d’Informatique

Filière Licence Fondamentale
en Sciences Mathématiques et Informatique

PROJET DE FIN D’ÉTUDES

Intitulé :

**DÉTECTION DU CANCER DE LA PEAU MELANOMA À
L'AIDE D'APPRENTISSAGE PROFOND**

Présenté par :
NEFZI SALMA

TAOUSSI ASSAAD

soutenu le 21 Juin 2024 devant le Jury

M. Yahya benkaouz Professeur à la Faculté des Sciences - Rabat *Président*
M. Rziza Mohammed Professeur à la Faculté des Sciences - Rabat *Encadrant*

Remerciements

Au terme de ce travail, nous tenons à exprimer notre profonde gratitude et nos sincères remerciements à notre encadrant, Professeur Rziza Mohammed. Son soutien indéfectible, ses conseils avisés et la qualité de son suivi ont été déterminants pour la réussite de notre projet de fin d'études. Il nous a offert non seulement son expertise, mais également son temps et sa patience, nous guidant à chaque étape avec une attention particulière.

Nos plus vifs remerciements s'adressent également à tous les membres du jury pour avoir pris le temps d'évaluer notre travail.

Un merci spécial à nos familles et amis qui nous ont soutenus moralement tout au long de cette période exigeante. Leur compréhension et leur encouragement ont été des piliers sur lesquels nous avons pu compter.

Enfin, nous souhaitons exprimer notre profonde gratitude à toutes les personnes qui ont, de près ou de loin, contribué à la réalisation de ce travail. Que ce soit par des encouragements, des suggestions constructives ou une aide ponctuelle, chaque geste a été précieux et nous a aidés à mener à bien ce projet.

Résumé

Ce rapport de fin d'études présente une analyse approfondie du "Détection du cancer de la peau mélanome à l'aide d'apprentissage profond", présente une analyse approfondie de l'application des réseaux de neurones convolutifs (CNN) pour le dépistage précoce du mélanome. L'objectif principal de cette recherche était d'explorer l'efficacité d'un modèle pré-entraîné DenseNet201 sur plus de 10000 images pour détecter les mélanomes avec une précision comparable à celle des dermatologues. Le rapport souligne les avantages de cette technologie, notamment l'accessibilité et la facilité d'utilisation, tout en abordant ses limites telles que les faux positifs et négatifs. Enfin, il discute des perspectives d'avenir pour améliorer la précision et l'efficacité des outils de détection basés sur l'IA .

Mots Clés : Intelligence artificielle, Apprentissage profond, Réseaux de neurones convolutifs, Tensorflow, Classification

Abstract

This final report presents an in-depth analysis of "Melanoma Skin Cancer Detection Using Deep Learning" and explores the application of Convolutional Neural Networks (CNN) for the early screening of melanoma. The primary objective of this research was to investigate the effectiveness of a pre-trained DenseNet201 model on over 10,000 images to detect melanomas with an accuracy comparable to that of dermatologists. The report highlights the benefits of this technology, including accessibility and ease of use, while also addressing its limitations such as false positives and negatives. Finally, it discusses future prospects for improving the accuracy and efficiency of AI-based detection tools.

Keywords : Artificial Intelligence, Deep Learning, Convolutional Neural Network, Tensorflow, Classification

Table des matières

Remerciements	2
Résumé	3
Abstract	4
Liste des Figures	7
1 Généralité	1
1.1 Introduction	1
1.2 Cancer de la peau	2
1.2.1 Types de cancer de la peau	2
1.2.2 Dépistage du cancer de la peau	3
1.2.3 Cancer de la peau bénin et malin	3
1.3 Historique et Avancées Clés	3
1.3.1 Méthodes Traditionnelles de Diagnostic du Mélanome	3
1.3.2 Premières Applications de l'Apprentissage Profond	4
1.3.3 Évolution et Tendances Récentes dans la Détection du Mélanome	5
1.4 conclusion	5
2 L'intelligence artificielle	6
2.1 Définition de L'intelligence artificielle	6
2.2 Apprentissage automatique	7
2.2.1 Apprentissage supervisé	7
2.2.2 Apprentissage non supervisé	8
2.2.3 Apprentissage par renforcement	8
2.2.4 Apprentissage profond	8
2.3 Réseau neuronal convolutif	9
2.3.1 pixels	10
2.3.2 Couche de convolution	11
2.3.3 Couche pooling	11
2.3.4 Appartissement	12
2.3.5 Couches de neurons	12
2.4 Conclusion	13
3 Application et Réalisation	14
3.1 Python	14
3.2 Google colab	14
3.3 Bibliothèques	15
3.3.1 TensorFlow	15
3.3.2 Keras	15
3.3.3 NumPy	15
3.3.4 Scikit-learn	16

3.4	Base de données	16
3.5	Partie code	18
3.5.1	Importation des bibliothèques	18
3.5.2	Augmenter les données d'entraînement et de test pour le modèle afin d'éviter le surapprentissage et normaliser les répertoires.	18
3.5.3	Montrer les images dans le répertoire d'entraînement.	20
3.5.4	Montrer les images dans le répertoire de test.	21
3.5.5	Construire un modèle.	22
3.5.6	Entraîner le modèle.	23
3.5.7	Tester le modèle.	25
3.5.8	Matrice de confusion.	25
3.5.9	Prédiction du modèle.	26
3.6	conclusion	27

Liste des Figures

1.1	Exemple d'image Mélanome	2
1.2	Exemple d'image Carcinome	2
1.3	Image du Cancer de la peau bénin et malin	3
1.4	Méthode Traditionnelle	4
1.5	Méthode Moderne	5
2.1	Relation entre IA,ML et DL	6
2.2	Types de modèles Machine Learning	7
2.3	Apprentissage automatique et apprentissage profond	9
2.4	CNN	9
2.5	les pixels	10
2.6	matrice des pixels	11
2.7	Types de pooling	12
2.8	Representation de flattening	12
2.9	Le réseau formé à la suite des étapes précédentes	13
3.1	La base de données	16
3.2	le dossier d'entraînement et test	17
3.3	Belign	18
3.4	Malign	18
3.5	Importation des bibliothèques	18
3.6	18
3.7	définition des constantes	19
3.8	Répartition des données d'entraînement et de validation dans des répertoires distincts	19
3.9	Génération d'images d'entraînement avec augmentation	20
3.10	Configuration générateurs, évaluation	20
3.11	Affichage d'exemples d'images à partir du répertoire d'entraînement	21
3.12	Exemples d'images organisés par classe	21
3.13	Affichage d'exemples d'images par classe à partir du répertoire de test	22
3.14	le modèle DenseNet201	22
3.15	les couches de convolution	23
3.16	Le modèle	23
3.17	entraînement de modèle	24
3.18	courbe d'accuracy et de loss	24
3.19	evaluate	25
3.20	des prédictions sur l'ensemble de test	25
3.21	matrice de confusion	26
3.22	test du modèle	26
3.23	prédiction du modèle	27

Introduction Générale

Le cancer de la peau est une maladie qui peut toucher n'importe qui, mais le dépistage précoce est crucial pour un traitement efficace. Cependant, le dépistage de cette maladie peut être difficile et nécessite souvent une expertise médicale spécialisée. Une nouvelle application innovante a été développée pour aider à détecter les différents modèles de cancer de la peau à un stade précoce. Dans ce rapport, nous allons examiner cette application et son utilité dans la lutte contre le cancer de la peau.

Nous commencerons par expliquer comment fonctionne cette application et les différents modèles de cancer de la peau qu'elle peut détecter, tels que le carcinome basocellulaire, le carcinome épidermoïde et le mélanome. Nous discuterons également de la précision de cette application et de ses limites actuelles.

Ensuite, nous examinerons les avantages de l'utilisation de cette application, tels que la facilité d'utilisation et la disponibilité pour un grand nombre de personnes, même dans des régions éloignées ou à faible revenu.

Nous discuterons également des inconvénients potentiels, tels que les faux positifs ou négatifs et les risques de diagnostic erroné. Nous aborderons également les implications pour la santé publique de cette application, en soulignant comment elle peut aider à augmenter le taux de dépistage précoce du cancer de la peau et à réduire le taux de mortalité associé à cette maladie.

Enfin, nous conclurons en soulignant l'importance de la détection précoce du cancer de la peau et de l'utilisation de technologies innovantes pour améliorer la santé et le bien-être des personnes du monde entier. Nous discuterons également des perspectives d'avenir pour cette application et des innovations futures qui pourraient améliorer sa précision et son efficacité.

Chapitre 1

Généralité

1.1 Introduction

Les tumeurs malignes de la peau sont les cancers les plus fréquents. Le cancer de la peau est très fréquent chez les personnes qui travaillent ou font du sport en extérieur et chez les adeptes des bains de soleil. Les personnes à la peau claire sont particulièrement sensibles à la plupart des formes de cancer de la peau, car elles produisent moins de mélanine. La mélanine, le pigment protecteur de la couche supérieure de la peau (épiderme), permet de protéger la peau de la lumière ultraviolette (UV). Cependant, le cancer de la peau peut aussi se développer chez des personnes à peau mate ou qui ne se sont pas exposées au soleil de façon importante. Les cancers de la peau peuvent également se développer plusieurs années après une radiothérapie ou une exposition à des substances provoquant un cancer.

objectif :

L'objectif principal de ce rapport est de :

- Présenter les avancées récentes dans l'utilisation de l'apprentissage profond pour la détection du cancer de la peau.

- Évaluer l'efficacité des modèles d'apprentissage profond par rapport aux méthodes de diagnostic traditionnelles.

- Explorer les défis et les limitations de l'intégration de ces technologies dans la pratique clinique quotidienne

- Discuter des perspectives futures et des possibilités d'amélioration des systèmes de détection basés sur l'apprentissage profond.

Problématique :

La problématique centrale de ce rapport est la suivante : Dans quelle mesure les technologies d'apprentissage profond peuvent-elles améliorer la précision et l'efficacité de la détection du cancer de la peau par rapport aux méthodes traditionnelles, et quels sont les obstacles à surmonter pour leur intégration réussie dans le domaine médical ?

Cette problématique soulève plusieurs questions sous-jacentes :

- Quels types de réseaux de neurones sont les plus efficaces pour la classification des images de la peau ?

- Comment garantir la qualité et la diversité des données utilisées pour entraîner ces modèles ?

- Quels sont les risques associés à une confiance excessive dans les systèmes automatisés ?

- Quelles sont les régulations nécessaires pour l'utilisation de l'IA en médecine ?

1.2 Cancer de la peau

Ce que l'on nomme cancer de la peau correspond à une multiplication anormale des cellules cutanées. Ce type de cancer, difficile à dépister et pouvant être grave, apparaît à n'importe quel âge. Il touche principalement les zones de peau (visage, bras...) qui sont les plus exposées aux ultraviolets.

1.2.1 Types de cancer de la peau

Il existe plusieurs types de cancer de la peau, les plus courants sont le carcinome basocellulaire, le carcinome épidermoïde et le mélanome. Le carcinome basocellulaire est le type de cancer de la peau le plus courant, mais il est généralement le moins dangereux, tandis que le mélanome est le type de cancer de la peau le plus dangereux et nécessite souvent un traitement agressif. Il y'a d'autres types de cancers de la peau, mais ils sont beaucoup moins courants.



FIGURE 1.1 – Exemple d'image Mélanome



FIGURE 1.2 – Exemple d'image Carcinome

1.2.2 Dépistage du cancer de la peau

Le dermatologue pratique des examens cliniques de la peau tous les jours à son cabinet. Concrètement, il réalise un examen visuel complet de votre peau pour repérer les taches ou grains de beauté pouvant faire suspecter un cancer. Il peut s'aider d'une sorte de loupe éclairante et très grossissante qui permet de voir à travers la première épaisseur de l'épiderme : une démoscopie. Si vous présentez de nombreux grains de beauté, il peut aussi les prendre en photo. S'il identifie une lésion suspecte, le dermatologue vous propose alors soit de la surveiller, soit de la retirer sous anesthésie locale et de la faire analyser pour confirmer, ou non, son diagnostic. Cette intervention peut être réalisée à son cabinet ou à l'hôpital. Elle ne nécessite pas d'hospitalisation.

1.2.3 Cancer de la peau bénin et malin

Le cancer de la peau est un problème de santé publique majeur, avec une incidence croissante dans de nombreux pays. Les cancers de la peau peuvent être bénignes ou malignes, ce qui signifie qu'ils peuvent être sans danger pour la santé ou qu'ils peuvent présenter un risque de propagation à d'autres parties du corps. Les types de cancer de la peau bénignes comprennent le kyste épidermique, le nævus, le lipome, le papillome et le fibrome, tandis que les types de cancer de la peau malignes comprennent le carcinome basocellulaire, le carcinome épidermoïde et le mélanome.



FIGURE 1.3 – Image du Cancer de la peau bénin et malin

1.3 Historique et Avancées Clés

1.3.1 Méthodes Traditionnelles de Diagnostic du Mélanome

Le diagnostic du mélanome, un type de cancer de la peau potentiellement mortel, a historiquement reposé sur des méthodes cliniques traditionnelles :

Examen Visuel et Dermatoscopie :

- **Examen Visuel** : Les dermatologues utilisent des critères cliniques tels que la forme, la couleur, la taille et les modifications des lésions pour identifier les mélanomes.
- **Dermatoscopie** : Un dermatoscope, permettant une visualisation plus détaillée des structures cutanées, améliore la précision du diagnostic en révélant des caractéristiques non visibles à l'œil nu.

Critères ABCDE :

- **A (Asymmetry)** : Asymétrie des lésions.
- **B (Border)** : Bords irréguliers.
- **C (Color)** : Couleur non uniforme.
- **D (Diameter)** : Diamètre supérieur à 6 mm.
- **E (Evolution)** : Changements de taille, forme ou couleur.

Biopsie : La biopsie excisionnelle ou incisionnelle est utilisée pour confirmer le diagnostic en examinant histopathologiquement les tissus suspects.

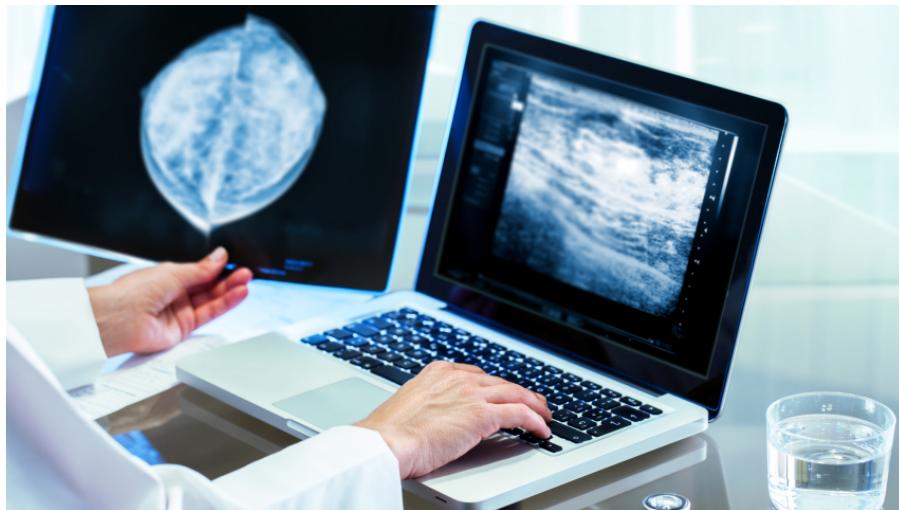


FIGURE 1.4 – Méthode Traditionnelle

1.3.2 Premières Applications de l’Apprentissage Profond

Étude de Stanford (2017) : L’étude de 2017 par l’Université de Stanford a marqué un tournant dans l’application de l’apprentissage profond pour le diagnostic du mélanome.

a. Objectifs et Méthodologie :

Objectif : Développer un modèle de réseau de neurones convolutifs (CNN) capable de classifier les lésions cutanées, y compris les mélanomes, avec une précision équivalente à celle des dermatologues.

Méthodologie : Utilisation de plus de 129 000 images cliniques représentant plus de 2 000 maladies cutanées différentes. Le modèle ResNet, pré-entraîné sur ImageNet, a été ajusté sur ces images.

b. Résultats :

Précision : Le modèle a démontré une précision comparable à celle des dermatologues dans la classification des mélanomes et autres lésions cutanées malignes.

Impact : Cette étude a établi la faisabilité et l’efficacité des CNN dans le diagnostic du mélanome, ouvrant la voie à une adoption plus large de ces techniques.

1.3.3 Évolution et Tendances Récentes dans la Détection du Mélanome

Depuis l'étude pionnière de 2017, plusieurs avancées et tendances notables ont émergé dans ce domaine :

Architectures Modernes : Des architectures plus sophistiquées comme DenseNet, EfficientNet et Inception ont été développées, offrant de meilleures performances et une meilleure efficacité.

Apprentissage par Transfert : Utilisation de modèles pré-entraînés sur de grandes bases de données générales (comme ImageNet), puis affinés avec des images spécifiques de mélanomes pour améliorer la précision.



FIGURE 1.5 – Méthode Moderne

1.4 conclusion

Ce chapitre a fourni une vue d'ensemble essentielle sur les tumeurs malignes de la peau, en détaillant leur prévalence, les facteurs de risque, et les types de cancers cutanés. Il a souligné l'importance de la détection précoce et de la prévention, ainsi que le rôle protecteur de la mélanine contre les rayons UV. Les objectifs principaux de l'étude ont été présentés, notamment l'évaluation des technologies d'apprentissage profond pour la détection du cancer de la peau, posant ainsi les fondations pour les chapitres suivants

Chapitre 2

L'intelligence artificielle

2.1 Définition de L'intelligence artificielle

L'intelligence artificielle (IA) est un domaine scientifique qui comprend diverses méthodes scientifiques, telles que la création de machines et de programmes informatiques en exploitant le maximum d'informations pouvant être initiées.

Certaines fonctions de l'intelligence humaine, dont la compréhension de certains langages, le raisonnement logique... L'objectif principal de l'intelligence artificielle est de développer des machines et des systèmes afin qu'ils puissent accomplir différentes tâches imposées à l'intelligence humaine. En effet, l'IA affecte les activités dans de nombreux domaines et secteurs, mais n'est pas largement utilisée, notamment les technologies de l'information et les logiciels, la médecine et la santé, les services bancaires, l'éducation, etc.

En résumé, l'intelligence artificielle (IA) est le processus d'imitation de l'intelligence humaine et repose sur la création et l'application d'algorithmes exécutés dans un environnement informatique dynamique.

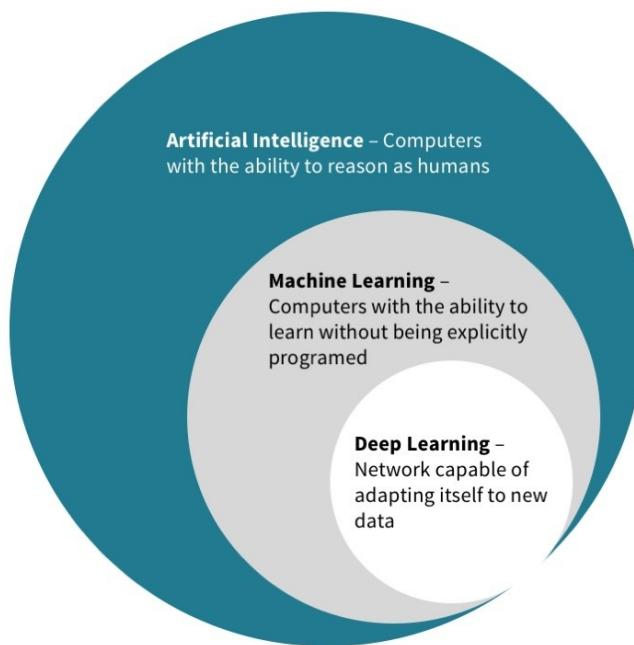


FIGURE 2.1 – Relation entre IA,ML et DL

2.2 Apprentissage automatique

Le Machine Learning, une branche de l'intelligence artificielle (IA), consiste à créer des algorithmes et des modèles qui apprennent à partir de données pour prendre des décisions ou faire des prédictions sans être explicitement programmés. Son principe fondamental est que les machines peuvent détecter des schémas et des relations dans les données en utilisant des algorithmes et des modèles statistiques.

Il existe différents types de modèles en Machine Learning, chacun utilisant des techniques algorithmiques spécifiques. Selon les données et les objectifs, on peut choisir entre quatre modèles principaux : supervisé, non supervisé, par renforcement, ou par apprentissage profond. Chaque modèle peut utiliser une ou plusieurs techniques algorithmiques selon les besoins. [7]

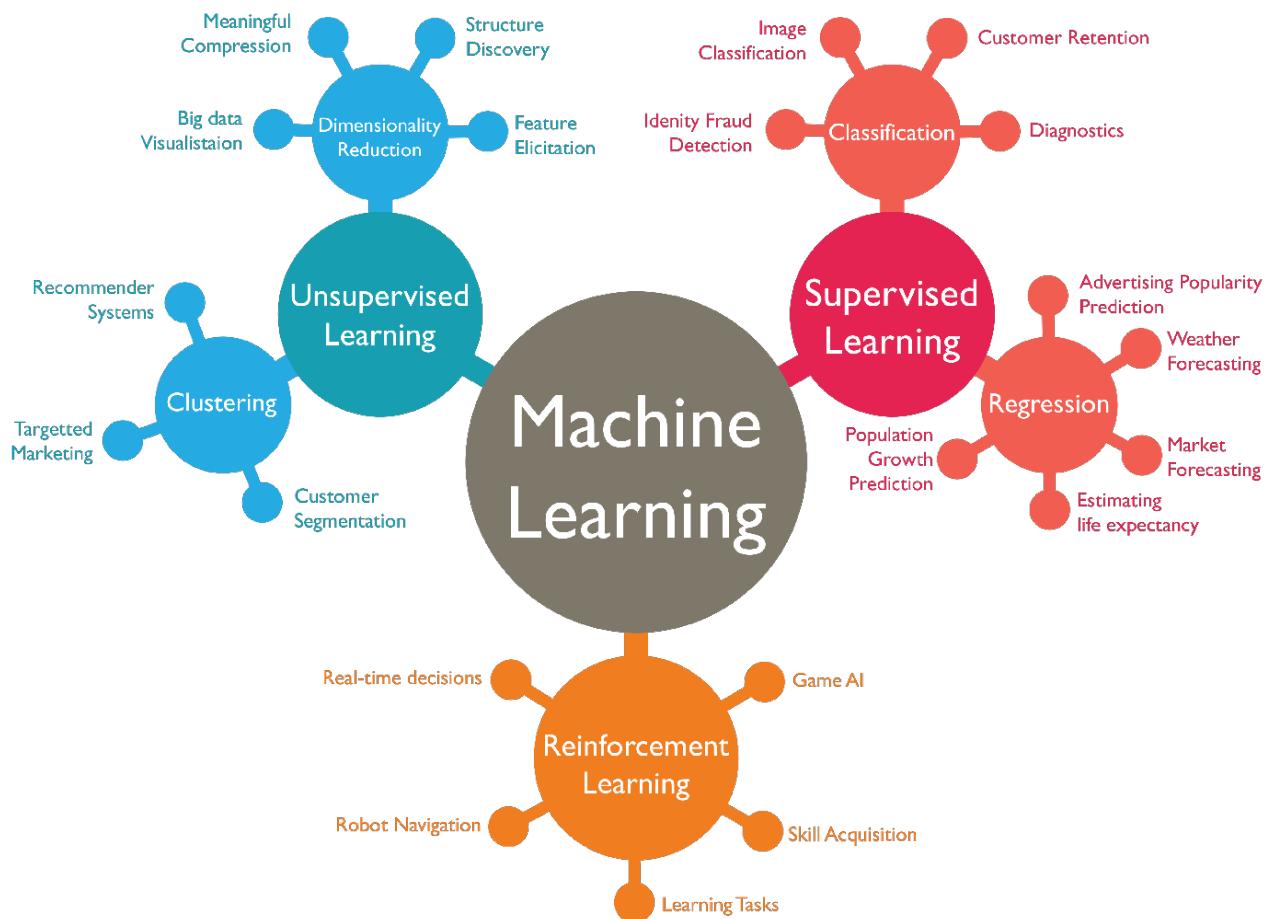


FIGURE 2.2 – Types de modèles Machine Learning
[8]

2.2.1 Apprentissage supervisé

Le premier type de Machine Learning s'appelle l'apprentissage supervisé.

Dans ce type d'apprentissage, la machine apprend en regardant des exemples. Les modèles d'apprentissage supervisé utilisent des paires de données "entrée" et "sortie", où la sortie est déjà étiquetée avec la bonne réponse. À l'aide d'un algorithme, le système étudie ces exemples au fil du temps et cherche des similitudes, des différences et d'autres schémas logiques jusqu'à ce qu'il puisse répondre à des questions par lui-même. C'est un peu comme quand on donne à un enfant des problèmes et les réponses correctes, puis on lui demande de montrer comment il a résolu les problèmes.

Les modèles d'apprentissage supervisé sont utilisés dans plusieurs applications quotidiennes, comme les suggestions de produits sur les sites de vente en ligne ou les applications de navigation

2.2.2 Apprentissage non supervisé

Le deuxième modèle de Machine Learning est l'apprentissage non supervisé. Contrairement au modèle supervisé, il n'y a pas de réponse donnée à la machine. Celle-ci étudie les données d'entrée, la plupart du temps non étiquetées et non structurées, pour rechercher des motifs et des corrélations en utilisant toutes les données pertinentes disponibles. L'apprentissage non supervisé est similaire à la façon dont les humains observent le monde : nous utilisons notre intuition et notre expérience pour regrouper des éléments. Plus nous avons d'expérience dans un domaine, plus nous sommes capables de le classer et de le comprendre avec précision. Pour les machines, cette "expérience" se traduit par la quantité de données auxquelles elles ont accès.

2.2.3 Apprentissage par renforcement

Le troisième modèle de Machine Learning est l'apprentissage par renforcement. Contrairement à l'apprentissage supervisé où la machine reçoit une réponse claire et apprend en trouvant des corrélations entre ces réponses, l'apprentissage par renforcement ne fournit pas de réponses précises. Au lieu de cela, il propose des actions possibles, des règles et des situations finales potentielles. Si l'objectif de l'algorithme est fixe ou binaire, les machines peuvent apprendre par l'exemple. Mais lorsque le résultat souhaité peut varier, le système doit apprendre à partir de l'expérience et des récompenses. Dans les modèles d'apprentissage par renforcement, la "récompense" est une valeur numérique programmée dans l'algorithme, que le système cherche à maximiser.

2.2.4 Apprentissage profond

Le quatrième modèle de Machine Learning est l'apprentissage profond. Il s'agit d'une approche avancée inspirée du fonctionnement du cerveau humain. Son élément central est l'utilisation de réseaux de neurones artificiels profonds, qui sont des structures avec plusieurs couches de neurones. Ces réseaux peuvent apprendre de manière hiérarchique en découvrant des caractéristiques complexes dans les données. Grâce à leur architecture en profondeur, ils peuvent représenter des modèles et des relations très sophistiqués, ce qui les rend idéaux pour des tâches comme la reconnaissance d'images, la traduction automatique, la compréhension du langage naturel, et bien d'autres encore.

Le deep learning est une avancée majeure en intelligence artificielle car il peut traiter des données non structurées et résoudre des problèmes complexes de manière plus précise et efficace.

Alors quel est le fonctionnement du Deep Learning ? Et quel type de modèle est le plus adapté au Deep Learning ?

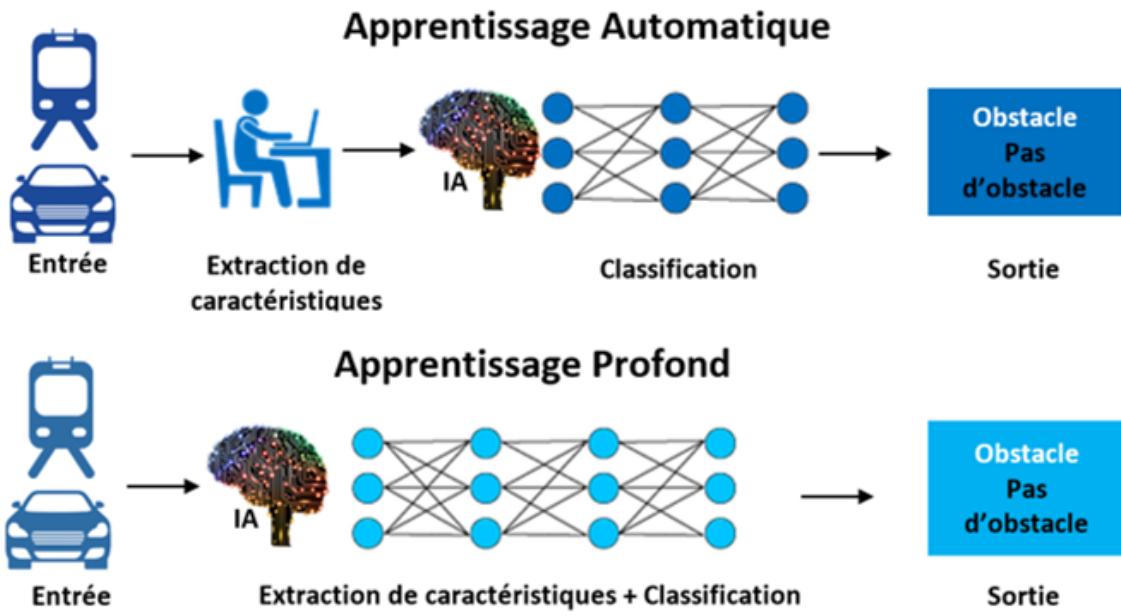


FIGURE 2.3 – Apprentissage automatique et apprentissage profond

2.3 Réseau neuronal convolutif

Un Convolutional Neural Network (CNN), ou réseau de neurones convolutif, est une architecture spécifique de réseau de neurones qui a été conçue pour traiter des données structurées, telles que des images et des vidéos. Inspiré par le fonctionnement du cortex visuel humain, un CNN utilise des opérations de convolution pour extraire des caractéristiques visuelles significatives à partir des données d'entrée. Les couches de convolution sont combinées avec des couches de pooling pour réduire la dimensionnalité des caractéristiques extraites. Les CNN sont particulièrement adaptés à la reconnaissance d'objets et de motifs visuels, grâce à leur capacité à apprendre des hiérarchies de caractéristiques à différentes échelles. Ils ont révolutionné le domaine de la vision par ordinateur, permettant des avancées majeures dans la détection d'objets, la segmentation d'images et la classification d'images à grande échelle. [11]

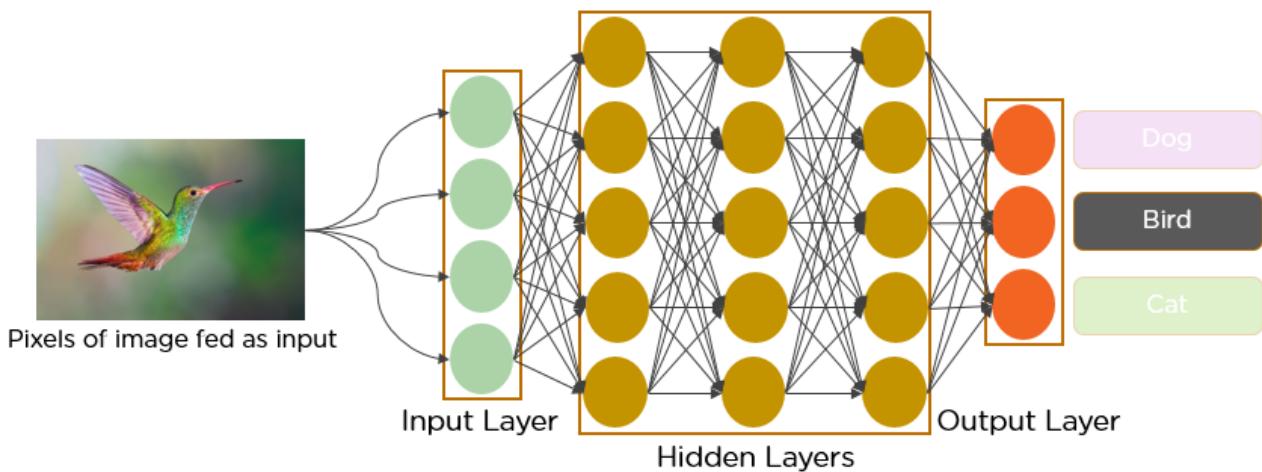


FIGURE 2.4 – CNN

2.3.1 pixels

Les pixels sont les éléments constitutifs fondamentaux des images numériques. Les pixels sont des blocs ou des carreaux de taille fixe. Chaque pixel est représenté par un nombre qui est déterminé en fonction des niveaux de couleurs RGB. ce dernier représente les trois composantes de couleur de base nommé :le rouge (R), le vert (G) et le bleu (B). En adaptant les valeurs de ces composantes pour chaque pixel, on peut créer différentes couleurs et nuances, ce qui permet de donner forme à l'image complète. Ainsi, l'image est formée par une grille de pixels, où chaque pixel est associé à un nombre correspondant à ses niveaux de couleurs RGB (exemple : noir = 0, Blanc = 255). Dès que l'on connaît le nombre à chaque pixel, ces nombres sont stockés dans un tableau à 2D (matrice).



FIGURE 2.5 – les pixels

La taille de chaque pixel est déterminée par sa résolution, qui est mesurée en points par pouce (DPI). Plus le DPI est élevé, plus chaque pixel individuel sera petit et précis. Les pixels peuvent également être regroupés en blocs ou en tuiles pour créer des éléments plus grands dans une image. En combinant des pixels ensemble pour former ces éléments plus grands, une image peut être créée avec plus de détails et de clarté que ce qui serait autrement possible avec des pixels individuels seuls.



```

0 2 15 0 0 11 10 0 0 0 0 0 9 9 0 0 0
0 0 0 4 60 157 236 255 255 177 95 61 32 0 0 29
0 10 16 119 238 255 244 245 243 250 249 255 222 103 10 0
0 14 170 255 255 244 254 255 253 245 255 249 253 251 124 1
2 98 255 228 255 251 254 211 141 116 122 215 251 238 255 49
13 217 243 255 155 33 226 52 2 0 10 13 232 255 255 36
16 229 252 254 49 12 0 0 7 7 0 70 237 252 235 62
6 141 245 255 212 25 11 9 3 0 115 236 243 255 137 0
0 87 252 250 248 215 60 0 1 121 252 255 248 144 6 0
0 13 113 255 255 245 255 182 181 248 252 242 208 36 0 19
1 0 5 117 251 255 241 255 247 255 241 162 17 0 7 0
0 0 0 4 58 251 255 246 254 253 255 120 11 0 1 0
0 0 4 97 255 255 255 248 252 255 244 255 182 10 0 4
0 22 206 252 246 251 241 100 24 113 255 245 255 194 9 0
0 111 255 242 255 158 24 0 0 6 39 255 232 230 56 0
0 218 251 250 137 7 11 0 0 0 2 62 255 250 125 3
0 173 255 255 101 9 20 0 13 3 13 182 251 245 61 0
0 107 251 241 255 230 98 55 19 118 217 248 253 255 52 4
0 18 146 250 255 247 255 255 255 249 255 240 255 129 0 5
0 0 23 113 215 255 250 248 255 255 248 248 118 14 12 0
0 0 6 1 0 52 153 233 255 252 147 37 0 0 4 1
0 0 5 5 0 0 0 0 0 14 1 0 6 6 0 0

```

FIGURE 2.6 – matrice des pixels

2.3.2 Couche de convolution

Pour obtenir une matrice de filtre à partir de la matrice de pixels M , on utilise généralement une petite matrice appelée "noyau" ou "masque". L'élément central du noyau est important (il est préférable d'utiliser une matrice de taille impair pour avoir un élément centré). Lorsque le filtre est appliqué à la matrice de pixels, chaque valeur de la matrice est multipliée par les coefficients du filtre, suivie d'une sommation des résultats. Le noyau est déplacé sur la matrice de pixels en effectuant ces multiplications et additions pour chaque position du noyau. Le résultat de cette opération est une nouvelle matrice appelée "image filtrée". Chaque élément de la matrice M a un élément correspondant (en vis-à-vis) dans la matrice de convolution $M * A$.

2.3.3 Couche pooling

La couche de Pooling est une opération souvent placée entre deux couches de convolution. Elle prend en entrée les cartes de caractéristiques générées par la couche de convolution et son objectif est de réduire la taille des images tout en préservant leurs caractéristiques les plus importantes. Les types de Pooling les plus couramment utilisés sont le max-pooling, qui sélectionne la valeur maximale dans chaque fenêtre de filtre, et l'average pooling, qui conserve la valeur moyenne de la fenêtre de filtre à chaque pas. En sortie de cette couche de Pooling, on obtient le même nombre de cartes de caractéristiques qu'en entrée, mais elles sont considérablement compressées.

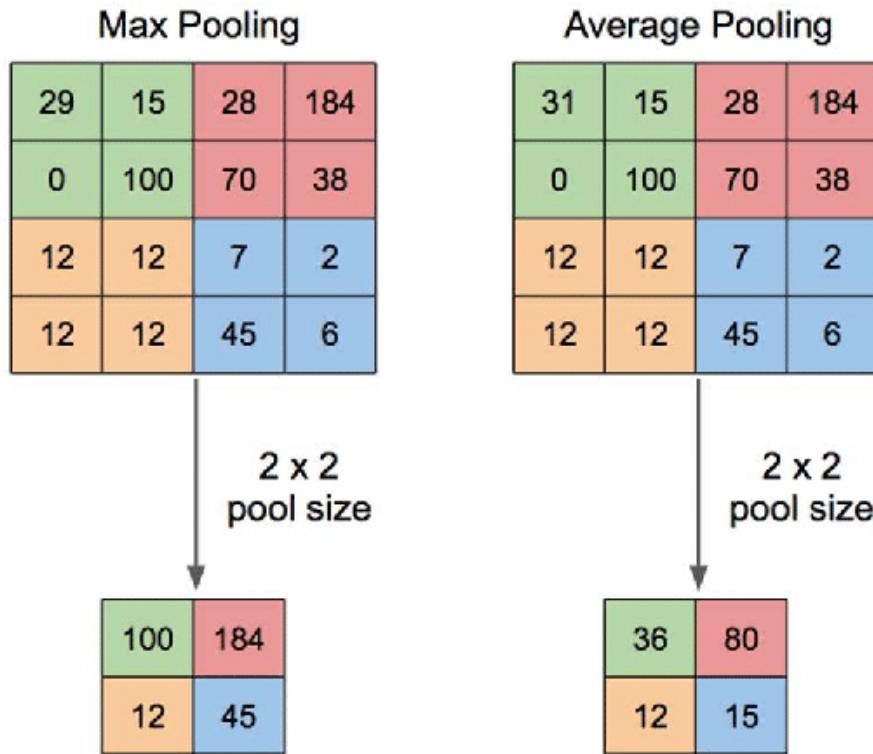


FIGURE 2.7 – Types de pooling

2.3.4 Applatissement

Cette étape est relativement simple. Après avoir terminé les étapes précédentes, nous devrions maintenant disposer d'une carte d'entités regroupées. Comme son nom l'indique, à cette étape, nous allons littéralement aplatisir notre carte de caractéristiques regroupées en une seule colonne. La raison pour laquelle nous transformons la couche de pooling en un vecteur unidimensionnel est que ce vecteur sera ensuite introduit dans un réseau neuronal artificiel. Autrement dit, ce vecteur deviendra la couche d'entrée d'un réseau neuronal artificiel qui sera connecté au réseau neuronal convolutif.

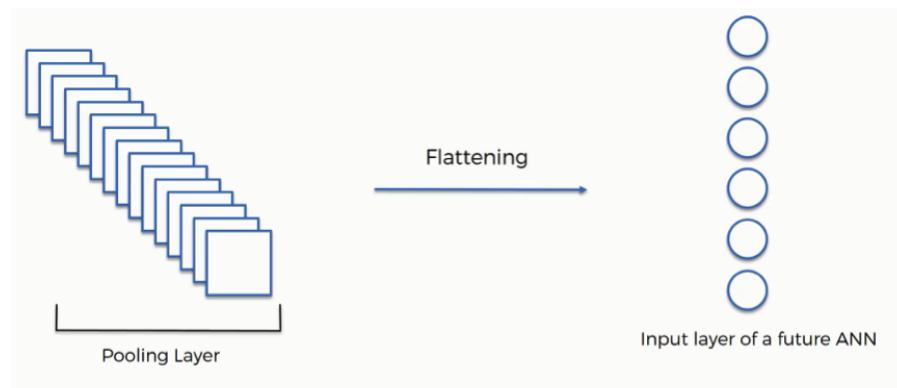


FIGURE 2.8 – Representation de flattening

2.3.5 Couches de neurons

Après l'aplatissement (flattening), nous obtenons des vecteurs qui représentent l'entrée. Ces vecteurs sont tous connectés aux neurones ; chaque neurone est connecté à tous les éléments du vecteur, et il est également connecté à la sortie Pour notre projet, la sortie est un booléen : si

il est déjà dans la base de données, la sortie est Vrai ('1'), sinon elle est Faux ('0').

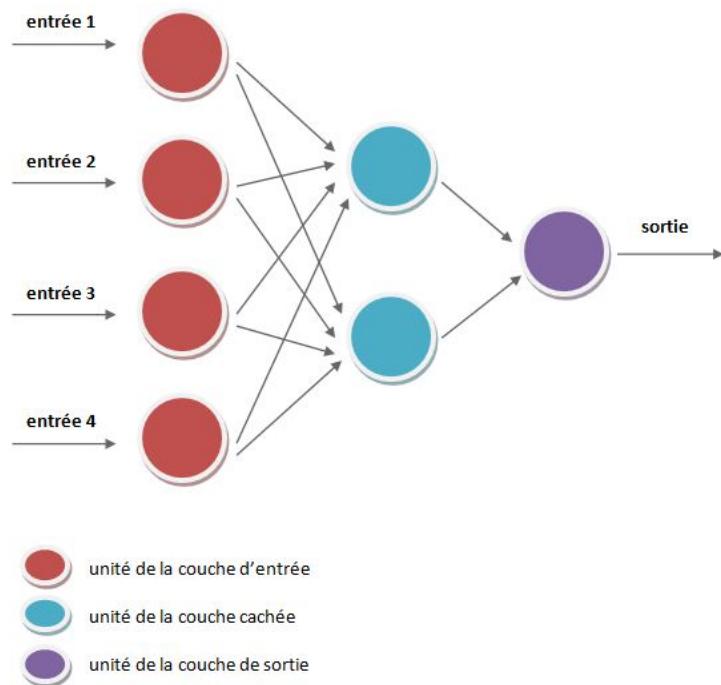


FIGURE 2.9 – Le réseau formé à la suite des étapes précédentes

2.4 Conclusion

En résumé, l'IA et les CNNs travaillent ensemble pour analyser et comprendre les données visuelles avec une précision inégalée. L'IA fournit la base théorique et les algorithmes nécessaires, tandis que les CNNs apportent des techniques avancées spécifiques pour le traitement des images, transformant notre approche de l'analyse visuelle et offrant des applications pratiques.

Chapitre 3

Application et Réalisation

3.1 Python

Python est un langage de programmation gratuit créé par le développeur Guido van Rossum en 1991. Il est considéré comme un langage interprété, ce qui signifie qu'il n'a pas besoin d'être transformé en un autre langage pour fonctionner. Un programme spécial, appelé "interpréteur", peut exécuter le code Python sur n'importe quel ordinateur. Cela signifie que vous pouvez voir les résultats de vos modifications de code rapidement. Cependant, cela peut rendre Python plus lent que d'autres langages, comme le C, qui sont compilés. Python est considéré comme un langage de haut niveau, ce qui signifie qu'il permet aux programmeurs de se concentrer sur ce qu'ils veulent accomplir plutôt que sur les détails techniques de la programmation. Cela rend l'écriture de programmes plus rapide et plus facile comparé à d'autres langages. C'est pourquoi Python est souvent recommandé pour les débutants en programmation.



3.2 Google colab

Google Colab, est une plateforme gratuite de Google qui permet d'écrire et d'exécuter du code Python directement dans un navigateur web. Conçue pour faciliter le travail collaboratif, Colab offre un accès facile à des environnements de calcul puissants, y compris des GPU et des TPU, ce qui le rend idéal pour les tâches de machine learning et d'analyse de données. Intégré avec Google Drive, il permet de sauvegarder, partager et collaborer sur des notebooks Jupyter sans nécessiter de configuration locale, rendant ainsi la programmation accessible à tous.



3.3 Bibliothèques

3.3.1 TensorFlow

TensorFlow est une bibliothèque open-source d'apprentissage automatique développée par Google Brain Team. Elle permet de créer des modèles d'apprentissage automatique, y compris des réseaux de neurones profonds, et de les entraîner pour effectuer diverses tâches telles que la classification, la reconnaissance d'images, la traduction, et bien plus encore. TensorFlow offre une flexibilité et une extensibilité considérables, ainsi qu'une large communauté de développeurs qui contribuent à son écosystème. Grâce à son architecture modulaire, TensorFlow est utilisé dans de nombreux domaines, notamment la recherche, l'industrie, la santé et l'automobile, pour créer des applications intelligentes et résoudre des problèmes complexes.



3.3.2 Keras

Keras est une bibliothèque puissante et accessible qui permet aux chercheurs et aux développeurs de créer et de déployer des modèles d'apprentissage profond avec une grande efficacité. Sa simplicité et sa flexibilité en font un outil de choix pour ceux qui souhaitent explorer le domaine de l'intelligence artificielle et de l'apprentissage automatique.



3.3.3 NumPy

NumPy est une bibliothèque pour langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux. Plus précisément, cette bibliothèque logicielle libre et open source fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes.



3.3.4 Scikit-learn

Scikit-learn est une bibliothèque de machine learning en Python, fournissant des outils pour l'apprentissage supervisé et non supervisé, la validation de modèles, le prétraitement des données et l'évaluation des performances. Elle propose une large gamme d'algorithmes et de méthodes pour résoudre divers problèmes d'apprentissage automatique, le tout dans une interface cohérente et facile à utiliser.



3.4 Base de données

La base de données utilisée dans ce projet est organisée en deux dossiers principaux : un pour l'ensemble de test et un pour l'ensemble d'entraînement. Chacun de ces dossiers est subdivisé en sous-dossiers correspondant aux différentes classes que le modèle vise à classifier, à savoir "maligne" et "benigne". Cette structure de dossier reflète fidèlement la réalité clinique, où les images médicales sont souvent catégorisées en fonction des diagnostics.

A screenshot of a Windows File Explorer window. The title bar says "Téléchargements > archive >". The toolbar includes "Organiser", "Nouveau", "Ouvrir", and "Sélectionner". A search bar says "Rechercher dans : archive". The main area shows a list of two items: "test" and "train".

Nom	Modifié le	Type
test	20/04/2024 18:16	Dossier de fichiers
train	20/04/2024 18:22	Dossier de fichiers

FIGURE 3.1 – La base de données

Dans le dossier de test, nous trouvons un total de 1000 images pour chaque classe, ce qui garantit une représentation équilibrée et équitable de chaque catégorie lors de l'évaluation du modèle. Cette équilibre est essentiel pour une évaluation juste de la performance du modèle, car il garantit que le modèle est testé de manière égale sur les deux classes, sans biais en faveur d'une classe particulière.

En revanche, le dossier d'entraînement contient un nombre plus important d'images pour chaque classe. Avec 6268 images pour la classe "maligne" et 5590 images pour la classe "benigne", cet ensemble de données offre une ample variété pour permettre au modèle d'apprendre de manière exhaustive les caractéristiques distinctives de chaque classe. Cette diversité dans l'ensemble d'entraînement est essentielle pour renforcer la capacité du modèle à généraliser à de nouvelles données et à reconnaître efficacement les différents motifs présents dans les images médicales.

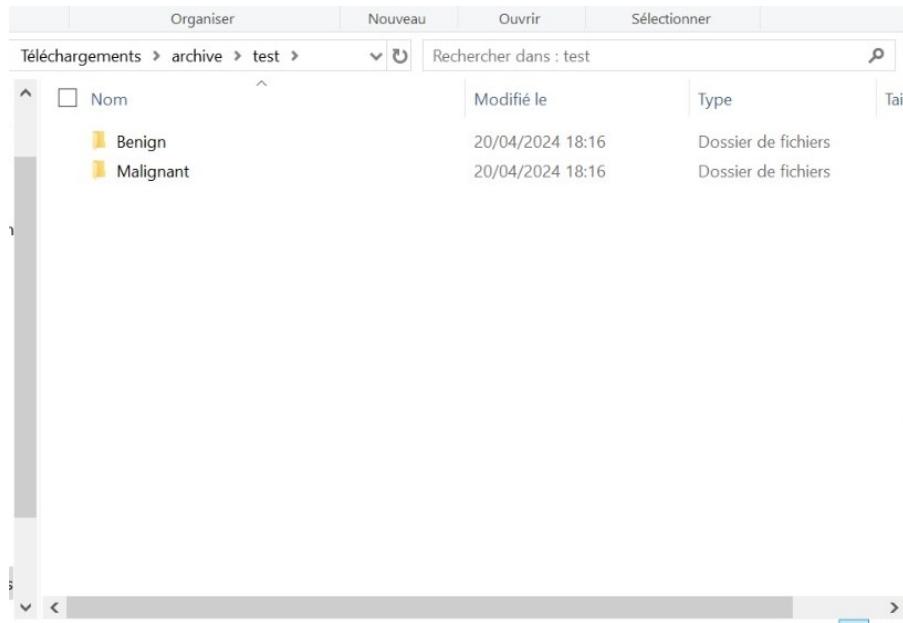


FIGURE 3.2 – le dossier d’entraînement et test

Ensuite, on a fait une augmentation des données a été mise en œuvre pour enrichir l’ensemble d’entraînement et améliorer la capacité du modèle à généraliser à de nouvelles images. Cette augmentation a été réalisée en utilisant la classe ‘ImageDataGenerator’ de Keras, qui permet d’appliquer une variété de transformations aux images, telles que la rotation, le décalage, le cisaillement et le zoom, ainsi que des flips horizontaux. Ces transformations ont été appliquées de manière aléatoire à chaque lot d’images pendant l’entraînement, ce qui a permis de créer de multiples variations des images d’origine, augmentant ainsi la diversité des exemples présentés au modèle.

Pour mettre en œuvre cette augmentation, les images ont été préalablement organisées dans des sous-dossiers correspondant à leurs classes respectives dans les ensembles d’entraînement et de validation. Une fois cette structure mise en place, les générateurs de données ont été configurés pour charger les images à partir de ces dossiers, en appliquant simultanément les transformations spécifiées par l’augmentation des données. Enfin, les générateurs ont été utilisés pour créer des flux de données itératifs qui fournissent des lots d’images augmentées au modèle lors de l’entraînement et de l’évaluation.

L’utilisation de cette technique d’augmentation des données a permis d’augmenter la quantité d’exemples disponibles pour l’entraînement, ce qui a amélioré la robustesse et la capacité de généralisation du modèle. Les résultats de cette augmentation se reflètent dans les statistiques d’entraînement, où l’ensemble d’entraînement comprend désormais un total de 8385 images réparties entre les deux classes, tandis que l’ensemble de validation en compte 3594. De plus, l’ensemble de test comprend 2020 images.



FIGURE 3.3 – Benign

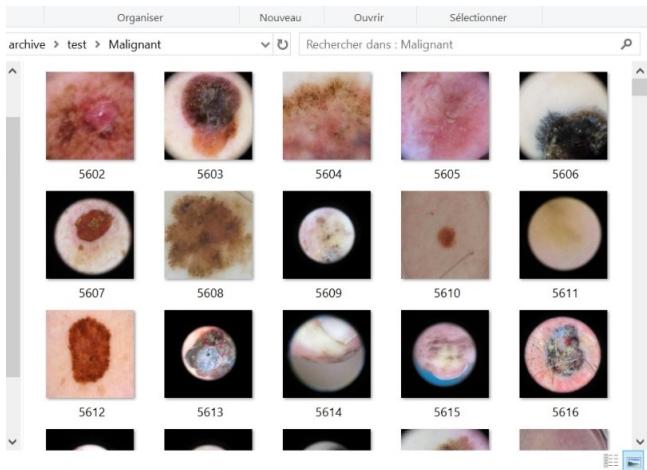


FIGURE 3.4 – Malignant

3.5 Partie code

3.5.1 Importation des bibliothèques

Le code commence par l'importation des bibliothèques nécessaires, notamment TensorFlow et Keras, pour construire et entraîner un modèle de réseau de neurones convolutionnel. Il utilise l'architecture DenseNet201 pré-entraînée, ainsi que des couches supplémentaires et la régularisation L2 pour améliorer les performances du modèle et éviter le surapprentissage. Le script inclut également des outils pour diviser les ensembles de données en ensembles d'entraînement et de test, augmenter les images pour améliorer la robustesse du modèle, et visualiser les résultats à l'aide de Matplotlib.

```
import tensorflow as tf
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras import layers, models
from tensorflow.keras.regularizers import l2
import os
import shutil
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

Python

FIGURE 3.5 – Importation des bibliothèques

On a utilisé Google Colab pour monter Google Drive dans l'environnement d'exécution. Cela permet d'accéder aux fichiers stockés sur Google Drive directement depuis Google Colab, facilitant ainsi la gestion et l'utilisation des données nécessaires pour le projet.

```
>     from google.colab import drive
drive.mount('/content/drive')
[4] ... Mounted at /content/drive
```

Python

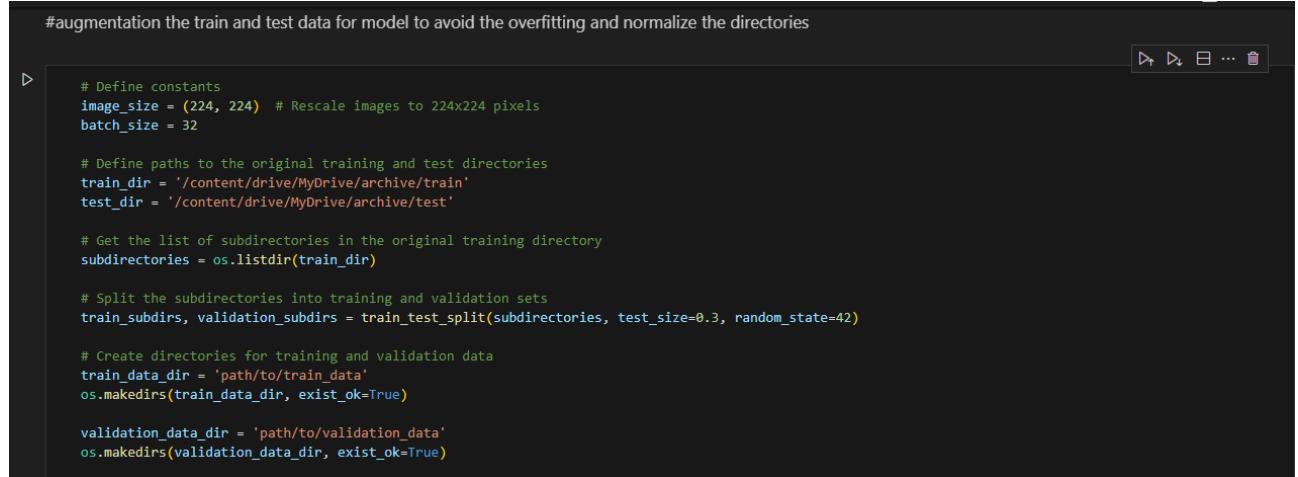
FIGURE 3.6

3.5.2 Augmenter les données d'entraînement et de test pour le modèle afin d'éviter le surapprentissage et normaliser les répertoires.

Pour éviter le surapprentissage et normaliser les répertoires de données, nous avons utilisé diverses techniques d'augmentation des données d'entraînement et de test pour notre modèle.

Nous avons d'abord défini des constantes, comme la taille des images à 224x224 pixels et une taille de lot de 32. Ensuite, nous avons spécifié les chemins des répertoires d'entraînement

et de test originaux situés sur Google Drive. Les sous-répertoires du répertoire d'entraînement ont été listés et divisés en ensembles d'entraînement et de validation en utilisant train-test-split avec une répartition de 70/30.



```
#augmentation the train and test data for model to avoid the overfitting and normalize the directories

# Define constants
image_size = (224, 224) # Rescale images to 224x224 pixels
batch_size = 32

# Define paths to the original training and test directories
train_dir = '/content/drive/MyDrive/archive/train'
test_dir = '/content/drive/MyDrive/archive/test'

# Get the list of subdirectories in the original training directory
subdirectories = os.listdir(train_dir)

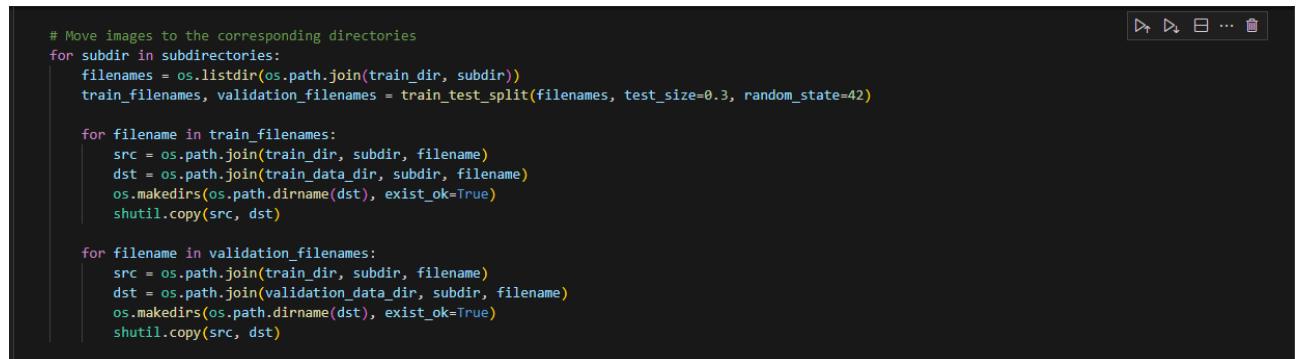
# Split the subdirectories into training and validation sets
train_subdirs, validation_subdirs = train_test_split(subdirectories, test_size=0.3, random_state=42)

# Create directories for training and validation data
train_data_dir = 'path/to/train_data'
os.makedirs(train_data_dir, exist_ok=True)

validation_data_dir = 'path/to/validation_data'
os.makedirs(validation_data_dir, exist_ok=True)
```

FIGURE 3.7 – définition des constantes

Des répertoires distincts pour les données d'entraînement et de validation ont été créés, puis les images ont été déplacées dans ces répertoires correspondants. Pour chaque sous-répertoire, les fichiers ont été répartis entre les ensembles d'entraînement et de validation, et copiés dans les répertoires appropriés.



```
# Move images to the corresponding directories
for subdir in subdirectories:
    filenames = os.listdir(os.path.join(train_dir, subdir))
    train_filenames, validation_filenames = train_test_split(filenames, test_size=0.3, random_state=42)

    for filename in train_filenames:
        src = os.path.join(train_dir, subdir, filename)
        dst = os.path.join(train_data_dir, subdir, filename)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        shutil.copy(src, dst)

    for filename in validation_filenames:
        src = os.path.join(train_dir, subdir, filename)
        dst = os.path.join(validation_data_dir, subdir, filename)
        os.makedirs(os.path.dirname(dst), exist_ok=True)
        shutil.copy(src, dst)
```

FIGURE 3.8 – Répartition des données d'entraînement et de validation dans des répertoires distincts

Ensuite, nous avons défini des générateurs de données avec augmentation pour l'entraînement, en appliquant diverses transformations comme la rotation, le décalage, le cisaillement, le zoom et le retournement horizontal pour augmenter la diversité des images d'entraînement. tandis que les données de validation restent inchangées, simplement en redimensionnant les images.

```

# Define data generators with augmentation for training and validation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

```

FIGURE 3.9 – Génération d’images d’entraînement avec augmentation

Nous avons ensuite créé des générateurs de données à partir de ces répertoires pour l’entraînement et la validation, spécifiant la taille des images, la taille des lots, et le mode de classification (catégoriel). Enfin, nous avons affiché le nombre d’échantillons dans chaque ensemble pour vérifier la répartition, et configuré un générateur de données de test pour évaluer le modèle final.

```

# Print the number of samples in each class for training and validation
print("Number of samples in training set:")
print(train_generator.classes)
print("\nNumber of samples in validation set:")
print(validation_generator.classes)

# Evaluate the model
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

```

[9] ... Found 8385 images belonging to 2 classes.
Found 3594 images belonging to 2 classes.
Number of samples in training set:
[0 0 0 ... 1 1 1]

Number of samples in validation set:
[0 0 0 ... 1 1 1]
Found 2020 images belonging to 2 classes.

FIGURE 3.10 – Configuration générateurs, évaluation.

3.5.3 Montrer les images dans le répertoire d’entraînement.

Ce segment de code vise à organiser les images du répertoire d’entraînement par classe et en affiche des exemples. Il commence par définir le nombre d’images à afficher par classe. Ensuite, il récupère les indices de classe et inverse la correspondance des indices en libellés de classe. Après cela, il parcourt chaque classe pour extraire et afficher les premières images de cette classe à partir du générateur d’entraînement. Ce processus permet de visualiser rapidement les exemples d’images utilisés pour entraîner le modèle.

```

# Define the number of images to plot from each class
num_images_per_class = 5

# Get class indices
class_indices = train_generator.class_indices

# Inverse mapping of class indices to class labels
class_labels = {v: k for k, v in class_indices.items()}

# Plot images from each class
for class_index, class_label in class_labels.items():
    # Find the first num_images_per_class images belonging to the current class
    class_images = [image for image, label in zip(train_generator[0][0], train_generator[0][1]) if label[class_index] == 1][:num_images_per_class]

    # Plot the images
    plt.figure(figsize=(10, 5))
    for i, image in enumerate(class_images):
        plt.subplot(1, num_images_per_class, i + 1)
        plt.imshow(image)
        plt.title(class_label)
        plt.axis('off')
    plt.show()

```

Python

FIGURE 3.11 – Affichage d'exemples d'images à partir du répertoire d'entraînement.

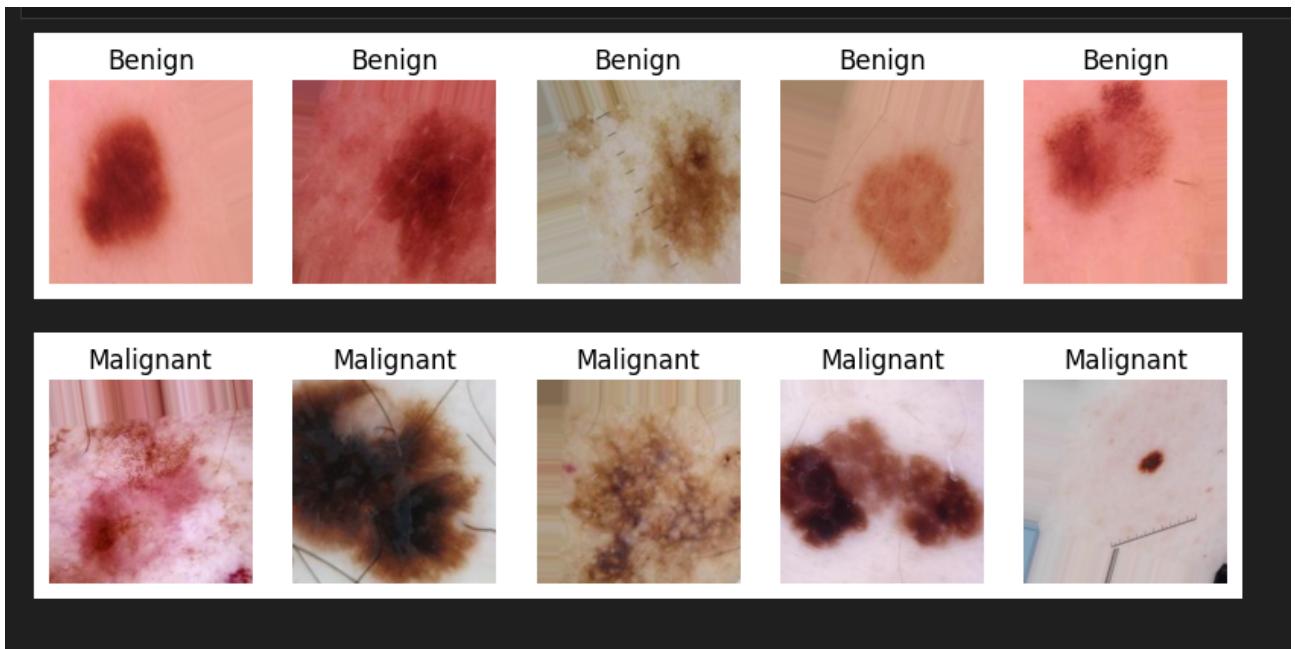


FIGURE 3.12 – Exemples d'images organisés par classe

3.5.4 Montrer les images dans le répertoire de test.

Ce code organise les images du répertoire de test par classe et en affiche des exemples. Il récupère les indices de classe et inverse la correspondance des indices en libellés de classe. Ensuite, il extrait et affiche les premières images de chaque classe à partir du générateur de test.

```

# Define the number of images to plot from each class
num_images_per_class = 5

# Get class indices
class_indices = test_generator.class_indices

class_labels = {v: k for k, v in class_indices.items()}

# Plot images from each class
for class_index, class_label in class_labels.items():
    # Find the first num_images_per_class images belonging to the current class
    class_images = [image for image, label in zip(test_generator[0][0], test_generator[0][1]) if label[class_index] == 1][:num_images_per_class]

    # Plot the images
    plt.figure(figsize=(10, 5))
    for i, image in enumerate(class_images):
        plt.subplot(1, num_images_per_class, i + 1)
        plt.imshow(image)
        plt.title(class_label)
        plt.axis('off')
    plt.show()

```

[5] ...

... <Figure size 1000x500 with 0 Axes>

FIGURE 3.13 – Affichage d'exemples d'images par classe à partir du répertoire de test

3.5.5 Construire un modèle.

Ensuite, nous utilisons le modèle DenseNet201 pré-entraîné sur ImageNet comme base de notre modèle de classification d'images. Nous définissons d'abord la forme d'entrée des images, qui est de 224x224 pixels avec 3 canaux de couleur (RVB). Ensuite, nous chargeons le modèle DenseNet201 en excluant la couche supérieure, et spécifions la forme d'entrée. Les couches de ce modèle de base sont gelées pour empêcher leur réentraînement. Nous créons un modèle séquentiel et ajoutons le modèle de base DenseNet201 comme première couche.

```

# Define the input shape
input_shape = (224, 224, 3)

# Load the DenseNet201 base model
base_model = DenseNet201(weights='imagenet', include_top=False, input_shape=input_shape, pooling=None)

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Define a sequential model that uses the base model
model = models.Sequential()

# Add the base model as the first layer
model.add(base_model)

```

FIGURE 3.14 – le modèle DenseNet201

Nous ajoutons trois couches convolutionnelles avec padding, chacune suivie d'une couche de normalisation par lots et d'une couche de max-pooling avec padding. La première couche convolutionnelle utilise 64 filtres, la deuxième en utilise 128, et la troisième en utilise 256. Après les couches convolutionnelles, nous ajoutons une couche de global average pooling pour réduire les dimensions des données sans perdre d'informations importantes.

```

# Add three convolutional layers with padding
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2), padding='same')) # Add padding here

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2), padding='same')) # Add padding here

model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2), padding='same')) # Add padding here

# Add global average pooling layer
model.add(layers.GlobalAveragePooling2D())

```

FIGURE 3.15 – les couches de convolution

Nous ajoutons ensuite une couche dense avec 128 unités, une activation ReLU, une régularisation L2 pour éviter le surapprentissage, et une couche de dropout avec un taux de 50/100 pour une régularisation supplémentaire. Enfin, nous ajoutons une couche de sortie avec 2 unités, une activation softmax et une régularisation L2 pour obtenir les probabilités des classes de sortie. Le modèle est compilé avec l'optimiseur Adamax et un faible taux d'apprentissage de 0.0001, en utilisant la perte catégorielle croisée (categorical-crossentropy) et en surveillant la précision (accuracy) comme métrique d'évaluation.

```

# Add dense layers with regularization and dropout
model.add(layers.Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(layers.Dropout(0.5))

# Add output layer with L2 regularization
output_layer = layers.Dense(2, activation='softmax', kernel_regularizer=l2(0.001))
model.add(output_layer)

# Compile the model with the specified optimizer and loss function
optimizer = tf.keras.optimizers.Adamax(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])


```

Python

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_ordering_tf_kernels_notop.h5
74836368/74836368 [=====] - 0s 0us/step

```

FIGURE 3.16 – Le modèle

3.5.6 Entrainer le modèle.

Nous définissons d'abord le nombre d'époques d'entraînement à 10. Ensuite, nous entraînons le modèle en utilisant les données de formation et de validation, et nous sauvegardons le modèle entraîné dans Google Drive. Le code utilise la méthode 'model.fit' pour l'entraînement, en spécifiant les générateurs de données d'entraînement et de validation, ainsi que le nombre d'époques. Après l'entraînement, le modèle est sauvegardé dans le répertoire spécifié. Enfin, le code génère des graphiques pour visualiser les courbes de précision et de perte pour les ensembles de formation et de validation.

```

# Define the number of epochs
num_epochs = 10

# Train the model
history = model.fit(
    train_generator,
    epochs=num_epochs,
    validation_data=validation_generator,
    verbose=1
)

model.save('/content/drive/MyDrive/saved_model/my_model')

# Plot training and validation accuracy and loss
plt.figure(figsize=(12, 4))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

```

FIGURE 3.17 – entraînement de modèle

Les courbes montrent que la précision d’entraînement commence autour de 0.7 et augmente régulièrement pour atteindre environ 0.86, tandis que la précision de validation suit une tendance similaire, bien que légèrement inférieure. En parallèle, la courbe de perte d’entraînement diminue progressivement, indiquant une réduction de l’erreur au fil des époques, et la perte de validation montre également une tendance à la baisse, bien que de manière moins régulière. Cela montre que le modèle s’améliore au fur et à mesure de l’entraînement, avec une meilleure performance et une réduction de l’erreur sur les données de validation.

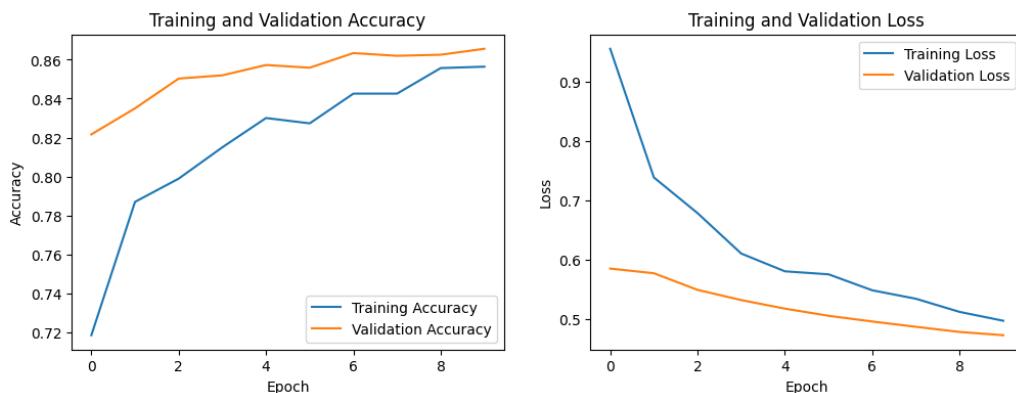


FIGURE 3.18 – courbe d’accuracy et de loss

Nous chargeons le modèle sauvegardé à partir de Google Drive en utilisant la fonction ‘tf.keras.models.load-model’. Ensuite, nous évaluons ce modèle sur un ensemble de données de test en utilisant le générateur de test. La méthode ‘evaluate’ permet de calculer la perte et la précision du modèle sur ces données de test. Les résultats de cette évaluation sont ensuite affichés, indiquant la perte et la précision obtenues sur l’ensemble de test. Cette étape permet de vérifier la performance du modèle sur des données non vues pendant l’entraînement, confirmant ainsi sa capacité de généralisation.

```

# Load the saved model
loaded_model = tf.keras.models.load_model('/content/drive/MyDrive/saved_model/my_model')

# Evaluate the model on the test dataset
test_loss, test_accuracy = loaded_model.evaluate(test_generator)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

```

[10] Python

... 64/64 [=====] - 504s 8s/step - loss: 0.4696 - accuracy: 0.8668
Test Loss: 0.4696390628814697
Test Accuracy: 0.8668316602706999

FIGURE 3.19 – evaluate

3.5.7 Tester le modèle.

Nous utilisons le modèle chargé pour faire des prédictions sur l'ensemble de test en appelant la méthode 'predict' sur le générateur de test. Les classes prédites sont déterminées en prenant l'indice de la probabilité maximale pour chaque prédition, ce qui nous donne les classes prédites pour chaque image de test. Les classes réelles sont extraites directement du générateur de test. Nous définissons ensuite les étiquettes de classe en utilisant les indices des classes du générateur de test.

```

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns
import numpy as np

# Make predictions on the test set
predictions = loaded_model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Plot confusion matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

```

Python

FIGURE 3.20 – des prédictions sur l'ensemble de test

3.5.8 Matrice de confusion.

Nous générerons une matrice de confusion en comparant les classes réelles et les classes prédites à l'aide de la fonction 'confusion-matrix' de Scikit-Learn. Cette matrice est ensuite visualisée sous forme de heatmap en utilisant Seaborn. La matrice de confusion montre la performance du modèle en termes de nombre de prédictions correctes et incorrectes pour chaque classe. Les étiquettes des classes sont affichées sur les axes x et y, et les valeurs de la matrice sont annotées pour une meilleure lisibilité. Ce graphique permet de visualiser où le modèle fait des erreurs et d'évaluer sa capacité à distinguer les différentes classes.

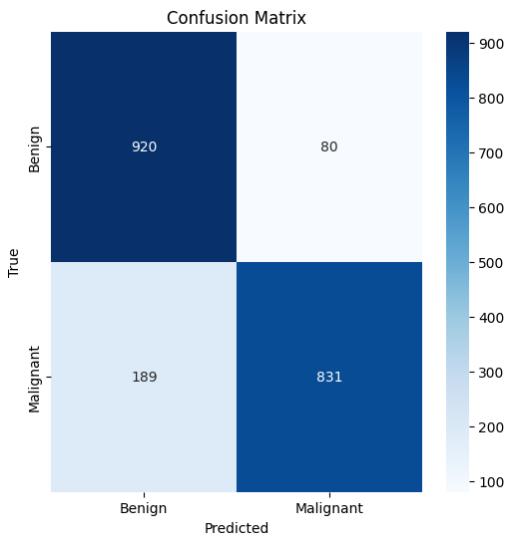


FIGURE 3.21 – matrice de confusion

3.5.9 Prédiction du modèle.

Nous définissons une fonction 'test-single-image' pour tester le modèle sur une seule image. La fonction prend en entrée le chemin de l'image et le modèle chargé. L'image est chargée et redimensionnée à la taille cible spécifiée. Ensuite, l'image est convertie en tableau de numpy et rééchelonnée en divisant par 255 pour normaliser les valeurs des pixels. L'image est ensuite augmentée d'une dimension pour correspondre au format d'entrée attendu par le modèle.

Le modèle prédit la classe de l'image en utilisant la méthode 'predict', et la classe prédictive est déterminée en prenant l'indice de la probabilité maximale. L'étiquette de classe correspondante est récupérée à partir de la liste des étiquettes de classe. L'image est ensuite affichée avec l'étiquette de classe prédictive comme titre.

```
# Test the model on a single image:
def test_single_image(image_path,model):
    img = load_img(image_path, target_size=image_size)
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0 # Rescale the image

    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction[0])
    predicted_label = class_labels[predicted_class]

    plt.imshow(img)
    plt.title(f"Predicted: {predicted_label}")
    plt.axis('off')
    plt.show()

# Example usage:
single_image_path = '/content/drive/MyDrive/archive/test/Benign/7282.jpg'
test_single_image(single_image_path,loaded_model)
```

1/1 [=====] - 10s 10s/step

FIGURE 3.22 – test du modèle

Pour illustrer l'utilisation de cette fonction, un chemin d'image d'exemple est fourni, et la fonction 'test-single-image' est appelée avec ce chemin et le modèle chargé. Cela permet de visualiser l'image testée et de voir la prédiction du modèle.

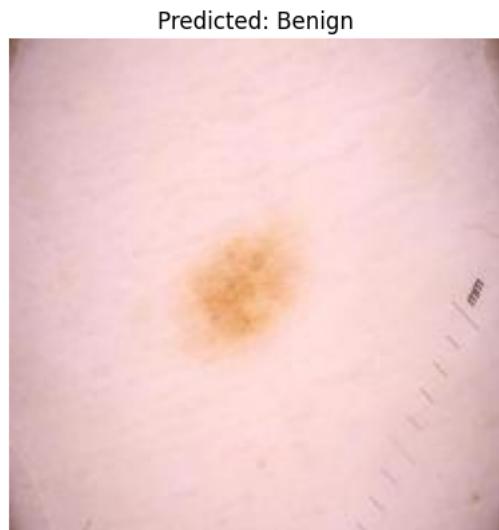


FIGURE 3.23 – prédition du modèle.

3.6 conclusion

En conclusion, ce projet de classification d’images a impliqué l’utilisation d’un modèle DenseNet201 pré-entraîné pour différencier deux classes d’images. En augmentant le volume de données à la fois pour l’ensemble d’entraînement, nous avons pu renforcer la capacité du modèle à généraliser sur de nouvelles données. Après avoir entraîné le modèle, nous l’avons évalué avec succès sur l’ensemble de test, obtenant une précision satisfaisante. L’utilisation de matrices de confusion nous a permis de comprendre où le modèle performait bien et où il rencontrait des difficultés. En outre, la fonction de test sur une seule image offre une évaluation visuelle pratique de la performance du modèle sur des exemples individuels. En somme, ce projet démontre l’efficacité de l’apprentissage automatique pour la classification d’images, avec une attention particulière portée à l’augmentation des données pour améliorer les performances du modèle.

Conclusion Générale

Dans ce projet, nous avons discuté des notions fondamentales de l'apprentissage automatique, des réseaux de neurones, en particulier les réseaux de neurones convolutifs. Nous avons introduit ces réseaux de neurones en présentant les différents types des couches utilisées. Nous avons implémenté un réseau de neurones avec trois couches. Pour ce dernier modèle nous avons effectué des tests qui consistent à changer le nombre d'époques à chaque fois, on examine la précision et la perte pour chacun des tests. L'implémentation a été faite avec le langage de programmation Python et on a utilisé des bibliothèques pour faciliter la tâche de la création de nos modèles et pour l'accélération du training.

Bibliographie

- [1] agence api. Hera-mi veut aller plus loin dans l'aide au diagnostique du cancer. <https://agence-api.ouest-france.fr/article/hera-mi-veut-aller-plus-loin-dans-laide-au-diagnostic-du-cancer-du-sein>.
- [2] dermomedicalcenter. Diagnostic cutane. <https://dermomedicalcenter.com/notre-expertise/diagnostic-cutane/>.
- [3] kaggle. Melanoma cancer image dataset. <<https://www.kaggle.com/datasets/bhaveshmittal/melanoma-cancer-dataset>>.
- [4] Kirmani. machine learning and deep learning project in python. <https://fiverr-res.cloudinary.com/images/t_main1,q_auto,f_auto,q_auto,f_auto/gigs/128782250/original/15cbb5904d950dd0eb5272612cd458e318b2b3c1/do-any-machine-learning-and-deep-learning-project-in-python.png>.
- [5] lemagit. Apprentissage supervisé et non supervisé : les différencier et les combiner. <https://www.lemagit.fr/conseil/Apprentissage-supervise-et-non-supervise-les-differencier-et-les-combiner>.
- [6] maisonabeille dermatologie. Comment prévenir les cancers de la peau ? <https://www.maisonabeille-dermatologie.com/actualites/>.
- [7] MathWork. Introduction aux réseaux neuronaux convolutifs (cnn). <<https://fr.mathworks.com/discovery/convolutional-neural-network.html>>.
- [8] sap. Qu'est-ce que le machine learning ? <<https://www.sap.com/canada-fr/products/artificial-intelligence/what-is-machine-learning.html>>.
- [9] wikipedia. La couche pooling. https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.
- [10] wikipedia. Numpy. [https://fr.wikipedia.org/wiki/Numpy_\(langage\)](https://fr.wikipedia.org/wiki/Numpy_(langage)).
- [11] wikipedia. Python. <[https://fr.wikipedia.org/wiki/Python_\(langage\)](https://fr.wikipedia.org/wiki/Python_(langage))>.
- [12] wikipedia. Réseau neuronal convolutif. https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.
- [13] wikipedia. Réseau neuronal convolutif. https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.
- [14] wikipedia. Scikit-learn. [https://fr.wikipedia.org/wiki/Scikit-learn_\(langage\)](https://fr.wikipedia.org/wiki/Scikit-learn_(langage)).
- [15] wikipedia. Tensorflow. <https://fr.wikipedia.org/wiki/TensorFlow>.