

Réf : PFA1-2023- 05

Rapport de Projet de Fin d'Année

de

Première année en Génie Informatique

Présenté et soutenu publiquement le 10/05/2023

Par

Aziz BENAMOR

Mohammed Nadhir NAJJAR

Développement d'une solution AI de détection d'une pneumonie

Composition du jury

Madame KOUKI Zoulei

Président

Monsieur BOULARES Mehrez

Encadrant

Année universitaire : 2022-2023

Remerciements

Nous souhaitons tout d'abord exprimer notre gratitude envers l'équipe pédagogique de l'école nationale supérieure d'ingénieurs de Tunis, qui a joué un rôle important dans la réussite de notre travail. Chaque membre de l'équipe a apporté sa contribution et nous a aidés à avancer dans notre projet.

Nous profitons de cette occasion pour remercier notre professeur et encadrant, Monsieur Mehrez BOULARES, pour son soutien et ses encouragements tout au long du projet, ainsi que pour sa générosité en matière de formation et d'encadrement. Nous le remercions également pour son aide, son soutien et ses conseils tout au long des différentes étapes de notre travail. Nous tenons également à exprimer notre gratitude envers les membres du jury pour l'honneur qu'ils nous ont fait en acceptant d'évaluer notre travail. Leur participation a été très importante pour nous, et nous les remercions chaleureusement.

Enfin, nous sommes reconnaissants envers tous nos professeurs, qui ont partagé avec nous leur expérience et leurs compétences, et qui nous ont encouragés à persévérer dans notre travail. Leur soutien a été précieux et nous les remercions pour leur contribution à notre formation.

Table des matières

Introduction générale.....	1
1. L'intelligence artificielle (IA) et la classification des images.....	2
2. L'utilisation de l'AI pour la détection de la pneumonie	3
2.1. Utilisation des réseaux de neurones convolutifs (CNN).....	3
2.2. Utilisation du « Transfert Learning ».....	4
2.3. Utilisation de l'apprentissage ensembliste	4
2.4. Autres techniques.....	5
3. Analyse comparative	5
4. Synthèse	7
Chapitre 2. Analyse et préparation des données	9
1. Description des données.....	9
2. Problématique.....	12
3. Prétraitement des données (Data pre-processing)	13
3.1. La re-division du Dataset (Data Split)	13
3.2. Le « Shuffle ».....	16
3.3. Le « Reshape ».....	16
3.4. La normalisation	17
4. Augmentation des données.....	18
Chapitre 3. Modélisation.....	20
1. Choix de l'architecture CNN.....	20
2. La couche de convolution	22
3. La fonction d'activation	23
4. La couche d'agrégation « Pooling ».....	25
5. La couche d'aplatissement « Flattening layer »	26
6. La couche entièrement connectée « Dense layer »	26
7. La couche Dropout.....	27
8. La couche de « Batch normalization »	28
9. Les métriques d'évaluation (Evaluation Metrics)	28
9.1. Précision.....	28
9.2. Recall	29
9.3. F1 score	29
9.4. Accuracy	30

Chapitre 4. Evaluation et déploiement	31
1. Etude technique.....	31
2. Construction des modèles.....	33
2.1. Modèle CNN classique	33
2.2. Modèle par Transfert Learning	35
2.2.1. Choix du modèle pré-entraîné	36
3. Entraînement des modèles.....	38
4. Evaluation des modèles.....	40
5. Déploiement du modèle	43
5.1. Création de l'API Flask :	43
5.2. Interface Utilisateur :	44
5.3. Hébergement :.....	44
Conclusion générale et perspectives	46

Table des figures

Figure 1 : Pourcentage d'utilisation des ensembles de données dans les recherches et nombres d'images radiographiques dans chacun	3
Figure 2: La décomposition initiale des données dans l'environnement Kaggle [37]	10
Figure 3 : La répartition des images en chaque classe	10
Figure 4: La répartition des images en chaque ensemble	11
Figure 5 : Exemple d'image radiographique normale et affectée.....	11
Figure 6 : Code de réarrangement des données.....	14
Figure 7 : Fonction de la répartition des données	14
Figure 8 : L'appel de la fonction	15
Figure 9: La nouvelle répartition des données	15
Figure 10 : Exemple d'architecture d'un simple CNN.....	21
Figure 11 : Choix de la fonction d'activation	24
Figure 12 : Différents types de Pooling	25
Figure 13 : Couche d'aplatissement.....	26
Figure 14 : couche entièrement connectée	27
Figure 15 : Paramètres du bloc-notes Kaggle	31
Figure 16 : Test de périphérique GPU	32
Figure 17 : Paramètres du premier modèle	34
Figure 18 : Architecture du premier modèle	34
Figure 19 : Code du premier modèle.....	35
Figure 20: Paramètres du deuxième modèle	35
Figure 21 : Création du deuxième modèle	36
Figure 22 : Architecture du deuxième modèle	36
Figure 23 : Architecture de EfficientNetB0.....	37
Figure 24 : Comparaison de la précision top-1 d'EfficientNetB0 avec d'autres modèles en fonction du nombre de paramètres	37
Figure 25 : L'entraînement du premier modèle	38
Figure 26 : La fonction de rappel ReduceLROnPlateau.....	38
Figure 27 : Résultat de l'entraînement du premier modèle	39
Figure 28 : Résultat de l'entraînement du deuxième modèle	39
Figure 29 : courbes de perte et de précision de l'entraînement et de la validation pour les deux modèles.....	40

Figure 30 : Evaluation des modèles par précision de chaque ensemble	41
Figure 31 : Matrice de Confusion	42
Figure 32 : Rapport de classification du modèle 2.....	42
Figure 33 : Rapport de classification du modèle 1.....	42
Figure 34 : Sauvegarde des modèles.....	43
Figure 35 : Pneumonia Online Detector (aziz0220.pythonanywhere.com)	44

Table des tableaux

Tableau 1 : Comparaison de différents algorithmes d'apprentissage automatique pour la détection de maladies	6
Tableau 2 : Environnement technique utilisé pour la mise en œuvre de l'approche proposée .	33

Introduction générale

La pneumonie est l'une des infections virales les plus courantes qui a été une menace pour la santé humaine tout au long de l'histoire. Les infections causées par les virus et les bactéries endommagent les poumons. Les symptômes de la pneumonie sont fréquents, tels que la douleur, la toux, l'essoufflement, etc. Selon les statistiques, cette maladie touche environ 14 % des enfants de moins de 5 ans dans le monde [1].

Pour diagnostiquer la pneumonie, les médecins utilisent généralement des examens physiques, des analyses de sang et des radiographies thoraciques. Cependant, ces techniques anciennes de détection de la pneumonie ont montré leurs limites, d'où l'importance de la détection précoce.

Ainsi, l'utilisation de L'intelligence artificielle dans la classification d'images a connu une évolution considérable ces dernières années, grâce à la disponibilité croissante de grandes quantités de données et à l'augmentation de la puissance de calcul. La classification d'images consiste à attribuer une ou plusieurs catégories à une image.

L'objectif de ce projet est de développer une solution basée sur l'intelligence artificielle pour la détection de la pneumonie. Notre travail consiste à explorer les différentes techniques et méthodes de Deep Learning pour la détection de la pneumonie, puis à concevoir et implémenter un modèle de classification performant à l'aide des données d'imagerie médicale disponibles (Chest-X-Ray). Nous évaluerons également les performances de notre solution en utilisant des mesures standard de classification. Finalement, nous allons la mettre en œuvre dans une application Web pour pouvoir l'utiliser.

Chapitre 1. Etat de l'art

Introduction

La reconnaissance de la pneumonie à partir d'images médicales est un domaine de recherche actif en Deep Learning, avec plusieurs approches qui ont été développées et améliorées au fil des années. Dans ce chapitre, nous présenterons l'évolution de l'intelligence artificielle dans la classification des images, ainsi que les différentes applications de l'AI pour la détection de la pneumonie et les méthodes utilisées. Nous allons également faire une analyse comparative des travaux existants dans ce domaine, en mettant en évidence leurs avantages et leurs limites.

1. L'intelligence artificielle (IA) et la classification des images

L'histoire de l'intelligence artificielle dans la classification d'images à commencer aux années 1960 avec les travaux de Hubel et Wiesel sur le cortex visuel des chats. [2] Ce qui a contribué à la conception des réseaux de neurones convolutifs comme le Neocognitron de Fukushima en 1980 ou le LeNet de LeCun en 1989. [3] Cependant, Ces réseaux limités par le manque de données et de puissance de computation.

Ce n'est qu'à partir des années **2010** que la classification d'images a connu un essor spectaculaire, avec l'apparition de grands ensembles de données comme **ImageNet**, qui contient plus de 14 millions d'images annotées dans plus de 20 000 catégories, et l'utilisation de processeurs graphiques (**GPU**) ou d'unités de traitement tensoriel (**TPU**), qui permettent d'accélérer le calcul parallèle. En **2012**, le modèle **AlexNet**, développé par **Krizhevsky et al.**, a remporté le défi ImageNet Large Scale Visual Recognition Challenge (ILSVRC), en réduisant le taux d'erreur de classification à 15,3 %, contre 26 % pour le meilleur modèle concurrent. Ce résultat a marqué un tournant dans le domaine de la vision par ordinateur et a suscité un intérêt croissant pour **les réseaux de neurones convolutifs (CNN)**. Depuis lors, plusieurs modèles plus performants et plus complexes ont été proposés, comme **VGGNet** en 2014, **GoogLeNet** en 2015, **ResNet** en 2016 ou **DenseNet** en 2017. Ces modèles ont permis d'atteindre des taux d'erreur inférieurs à 5 % sur le défi ImageNet, dépassant ainsi les performances humaines.

2. L'utilisation de l'AI pour la détection de la pneumonie

La détection de la pneumonie à partir de radiographies du thorax est un défi majeur pour la santé publique, qui nécessite des solutions innovantes basées sur le Deep Learning.

Le Deep Learning est une technique d'apprentissage automatique qui permet d'analyser des données complexes, comme les images médicales, et d'en extraire des informations pertinentes.

Pendant les dernières années, l'apprentissage en profondeur a été largement appliqué à la détection des pathologies pulmonaires à partir d'images de radiographies thoraciques par de nombreux chercheurs. Ces travaux ont exploité différents jeux de données disponibles. Ces figures en dessous montrent les différents « Datasets » tels que **RSNA Dataset**, **Mendeley Dataset**, **ChestX-ray14** ...etc. et leurs volumes en images de rayons X de la pneumonie et de l'état normal.

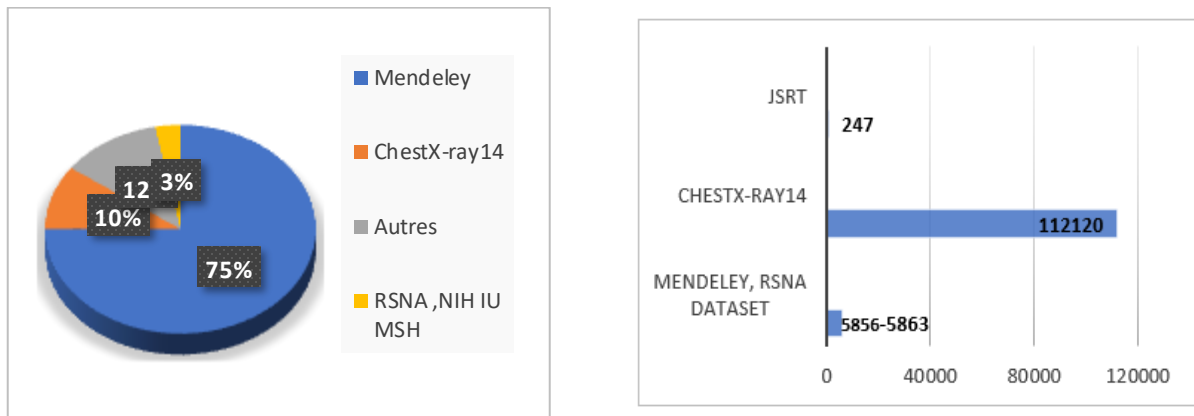


Figure 1 : Pourcentage d'utilisation des ensembles de données dans les recherches et nombres d'images radiographiques dans chacun

2.1. Utilisation des réseaux de neurones convolutifs (CNN)

Pour détecter la pneumonie, plusieurs travaux ont utilisé des réseaux de neurones convolutifs, qui sont des modèles spécialisés dans le traitement des images. Les « CNN » sont capables d'extraire automatiquement les caractéristiques pertinentes des images sans avoir besoin d'une extraction manuelle préalable.

Parmi les travaux existants, nous pouvons citer CheXNet [4] et CovXNet [5]. CheXNet est un réseau de neurones de 121 couches qui a été entraîné sur 100 000 images de radiographies thoraciques provenant du jeu de données ChestX-ray14. Ce modèle a montré une meilleure performance que les radiologues moyens pour la détection de la pneumonie et a atteint une précision de 92,8%. Tandis que, CovXNet est un CNN qui utilise la convolution à profondeur

variable avec des taux de dilatation variables pour extraire plusieurs caractéristiques à partir d'images radiographiques. L'architecture a été entraînée et validée sur 6161 scans de radiographie pulmonaire et a obtenu une précision de 90,2% dans une classification de pneumonie en quatre classes et de 97,4% dans une classification en deux classes.

Ces deux modèles ont dépassé les méthodes antérieures basées sur des caractéristiques manuelles traditionnelles pour la détection de la pneumonie. Cependant, les modèles CNN nécessitent beaucoup de données annotées pour l'entraînement et demandent des ressources de calcul très élevées en raison de leur grand nombre de couches et du temps pris pour la recherche d'une architecture optimale du modèle.

2.2. Utilisation du « Transfert Learning »

Le transfert Learning est une technique d'apprentissage automatique qui consiste à réutiliser un modèle pré-entraîné sur un problème différent. Le transfert Learning permet aux modèles CNN d'être plus performants, moins coûteux et de nécessiter moins de données. Plusieurs architectures de modèles pré-entraînés, comme Xception, VGGNet, InceptionV3 et ResNet ont été employées pour la classification de la pneumonie [6].

Parmi les travaux existants, on peut mentionner, « Jaipurkar et al. [7] » qui ont proposé en 2018 un modèle basé sur « DenseNet » pour classifier les images de radiographie pulmonaire, y compris celles présentant une pneumonie, afin de réduire l'erreur et de réexploiter les caractéristiques. Alhudhaif et al. [8] ont déployé une architecture basée sur DenseNet-201 et ont obtenu des valeurs de précision et de rappel allant jusqu'à 95 % et une précision presque de 90 % sur plus de 6000 images de rayons X.

2.3. Utilisation de l'apprentissage ensembliste

Cette méthode combine deux techniques : la combinaison des modèles CNN pour l'entraînement et la sélection du meilleur modèle. Plusieurs travaux ont utilisé des approches similaires avec des résultats variables. Par exemple, Sirazitdinov et al. [9] ont obtenu un rappel de 80 % avec un ensemble de deux CNNs, RetinaNet et Mask R-CNN. Jaiswal et al. [10] ont amélioré la précision en utilisant Mask RCNN avec ResNet50 et ResNet101, et en appliquant l'augmentation d'image, le dropout et la régularisation L2. Jain et al. [11] ont comparé les performances de VGG-16, VGG-19, Resnet50 et Inception-V3, avec des précisions allant de 71 % à 88 %.

2.4. Autres techniques

On trouve aussi des travaux qui ont exploré d'autres méthodes uniques pour la détection et la classification de la pneumonie à partir d'images radiographiques. Par exemple, Karthik et al. [12] ont employé l'architecture CNN Channel-Shuffled Dual-Branched (CSDB), qui combine différents types de convolutions pour apprendre des caractéristiques spatio-temporelles distinctives, et ont obtenu des scores F1 de 94% à 98% sur plusieurs ensembles de données publics. Wang et al. [13] ont conçu un bloc d'apprentissage résiduel avec attention préalable et l'ont intégré à deux réseaux 3D-ResNet pour atteindre une précision de 93,3% dans la détection de pneumonie en trois classes. Wang et al. [14] ont proposé une approche basée sur les réseaux antagonistes génératifs (GAN) pour synthétiser des images radiographiques de poumons sains et malades, avec une précision de 92,6% dans la classification binaire de la pneumonie.

Après citer tous ces méthodes employées dans les recherches réalisées avec différents ensembles de données. On va essayer de les comparer en donnant les avantages et inconvénients de chacune.

3. Analyse comparative

Références	Année	Algorithme / Méthode	Précision	Ensemble de données
Sarada N, Rao K. [15]	2021	ANN	92%	« Hôpital Sasoo »
Quan et al. [16]		DenseNet et CapsNet,	90,7 %	
Hegedűs I, Danner G, Jelasity M. [17]		Gossip Training	95% LR 96% SVM	« Spambase binary classification »
Rajinikanth V, et al. [18]	2020	DCNN	76%	ChestX-ray14
Ouchicha et al. [19]		Architecture de CNN résiduelle classification de pneumonie en trois classes .	96,7%	2905 images radiographiques
Yi P, Kim T, Lin C [20]		CNN (ResNet-50)	AUC interne : 93,1% AUC externe : 81,5%	NIH IU MSH
Jain et al.		VGG-16, VGG-19, Resnet50 et Inception-V3	de 71 % à 88 %.	
Dey et al. [21]		VGG-19	95 %	
Naqvi SZH, Choudhry MA [22]		Discrete wavelet transform (DWT)	99,70%	COPD dataset
Vyas J, Han M, Li L et al [23]		Blockchain	95,08%	EHR

Huang L, Shea AL, Qian H, et al. [24]	2019	Apprentissage fédéré	95,02%	EHR
Rajaraman S et al. [25]		VGG-16	96,2%	Radiographies du thorax
Liang et Zheng [26]		CNN personnalisé (49 couches de convolution + 2 couches denses)	89,1%	RSNA
O'Quinn et al. [27]		AlexNet	Précision initiale 72%	RSNA
Stephen et al. [28]		CNN personnalisé	Une précision d'entraînement maximale de 95% et une précision de validation de 93,73%	Mendeley
Islam et al. [29]		CNN personnalisé	97,34%	Mendeley
Sirazitdinov et al., Ko et al.		Ensemble de RetinaNet et Mask R-CNN	80%	
Jaiswal et al. [30]		Ensemble de ResNet50 et ResNet101 avec Mask R-CNN		Mendeley
Jaipurkar et al. [31]	2018	DensNet		
Kermany et al.		InceptionV3 avec des poids entraînés sur ImageNet et transfert de connaissances	96,6%	Classification d'images médicales (108,312 images OCT)
Rajpurkar et al.		CNN	Précision moyenne de 92,5%	ChestX-ray14
Afshar et al.		CapsNet	86,56%	Classification de tumeurs cérébrales sur des images d'IRM
Vianna [32]		Fine-tuning de transfert de connaissances avec augmentation de données		
Rajpurkar et al.	2017	CheXNet	Précision moyenne de 79,0%	ChestX-ray14
Qing et al. [33]	2014	CNN personnalisé		Chest-X-Ray8 +400 000 images
Parveen and Sathik [34]	2011	DWT, WFT, WPT, Fuzzy C-means		
Caicedo et al. [35]	2009	SIFT et SVM	67%	JSRT, MC X-ray set, and Shenzhen
Paredes et al. [36]	2002	Méthode traditionnelle k-NN avec des patches d'images médicales et ORB et SVM		1617 images

Tableau 1 : Comparaison de différents algorithmes d'apprentissage automatique pour la détection de maladies

Ce tableau compare les différentes approches de détection de la pneumonie à partir d'images radiographiques entre les années 2002 et 2021. Les algorithmes étudiés sont divers, incluant des réseaux de neurones convolutifs (CNN), des réseaux de neurones artificiels (ANN), des méthodes de classification traditionnelles telles que SVM et k-NN, ainsi que des approches de transfert de connaissances et d'apprentissage ensembliste. Les ensembles de données étudiés sont également variés, allant des images radiographiques d'hôpitaux spécifiques à des ensembles de données plus généraux tels que ChestX-ray14 ou des données Mendeley. On remarque que la précision des modèles varie considérablement, avec des précisions allant de 67% à 99,70%. Les approches d'ensemble et de transfert de connaissances semblent avoir des résultats plus cohérents et plus élevés que les approches autonomes. Les méthodes basées sur l'apprentissage fédéré et la blockchain ont également montré des résultats prometteurs pour préserver la vie privée des patients.

Cependant, chaque approche a ses avantages et ses inconvénients. Par exemple, les méthodes basées sur les réseaux de neurones peuvent avoir des performances impressionnantes, mais peuvent être inefficaces en cas de changement d'image radiographique. Les méthodes de classification traditionnelles peuvent également être moins efficaces pour les ensembles de données en temps réel.

Il convient également de noter que plusieurs études ont utilisé des approches d'augmentation d'image pour améliorer les performances des modèles, ainsi que des techniques de régularisation pour éviter le surapprentissage. En outre, certains modèles ont été conçus pour détecter plusieurs classes pathologiques différentes (Pneumonie bactérienne ou virale, Covid-19), tandis que d'autres se concentrent sur une seule maladie.

4. Synthèse

En somme, les études examinées ont révélé que les réseaux de neurones convolutionnels sont prometteurs pour la détection et la classification de la pneumonie à partir d'images radiographiques, mais des limites telles que la taille des ensembles de données et la complexité des architectures de réseau doivent être prises en compte.

L'apprentissage automatique en médecine présente de grands avantages, mais les défis à relever comprennent la qualité et la diversité des données. En outre, les modèles d'apprentissage automatique ont le potentiel d'améliorer les diagnostics précoces.

Conclusion

Comme nous avons pu le voir dans ce premier chapitre, plusieurs approches d'IA telles que l'apprentissage par transfert, les réseaux neuronaux profonds et la méthode « ensemble » sont largement utilisées pour la détection de la pneumonie à partir d'images médicales. Les perspectives d'amélioration proposées dans ces études sont importantes pour orienter notre propre projet. Dans le chapitre suivant, nous allons commencer par l'analyse et la préparation des données.

Chapitre 2. Analyse et préparation des données

Introduction

L'étape de traitement des données est très importante dans le succès de ce projet car il se base essentiellement sur fournir un grand nombre de données comme « inputs ». L'importance de cette étape réside dans le fait que les données brutes peuvent inclure des erreurs, des valeurs erronées ou même des fausses données. Il sera donc nécessaire de les nettoyer et de bien les préparer pour que la performance du modèle ne soit pas affectée.

Dans le cadre de notre modèle qui accepte des images comme des entrées, si les images sont floues ou contiennent du bruit, cela peut affecter significativement son efficacité.

1. Description des données

Les données que nous avons utilisées pour entraîner notre modèle sont des images radiographiques de la poitrine (Chest X-Ray) importés depuis la plateforme « Kaggle ». Le Dataset contenant 5863 images de type JPEG a déjà été organisé en 3 dossiers (train, test, val) et contient des sous dossiers pour chaque catégorie d'image (Pneumonie/Normal). Ces images ont été sélectionnées à partir du centre médical pour femme et enfants de Guangzhou, en chine. Elles ont été prises dans le cadre des soins cliniques de routine des patients.

Pour l'analyse des images de radiographie pulmonaire, toutes les radiographies ont été examinées pour le contrôle de qualité en retirant tous les scans de qualité médiocre ou illisibles. Les diagnostics pour les images ont ensuite été évalués par deux médecins experts avant d'être autorisés pour l'entraînement du système d'IA. Pour tenir compte de toutes les erreurs d'évaluation, l'ensemble de données d'évaluation a également été vérifié par un troisième expert.

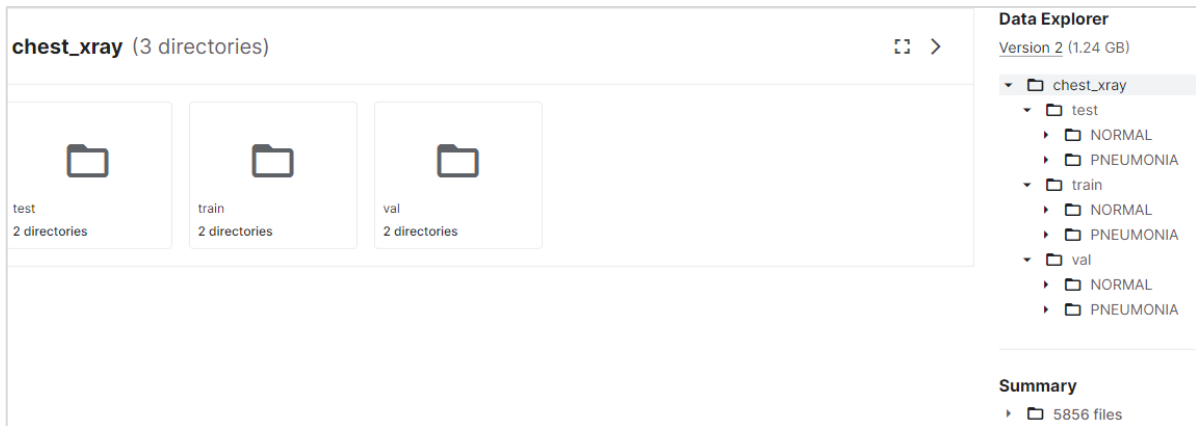


Figure 2: La décomposition initiale des données dans l'environnement Kaggle [37]

La figure ci-dessus montre que le Dataset est initialement décomposé en 3 parties [test, train, val], chaque partie est divisé lui-même en 2 sous parties [Pneumonia, Normal], nous avons donc 5216 images pour le train set, 624 images pour le test set, et seulement 16 images pour le val set.

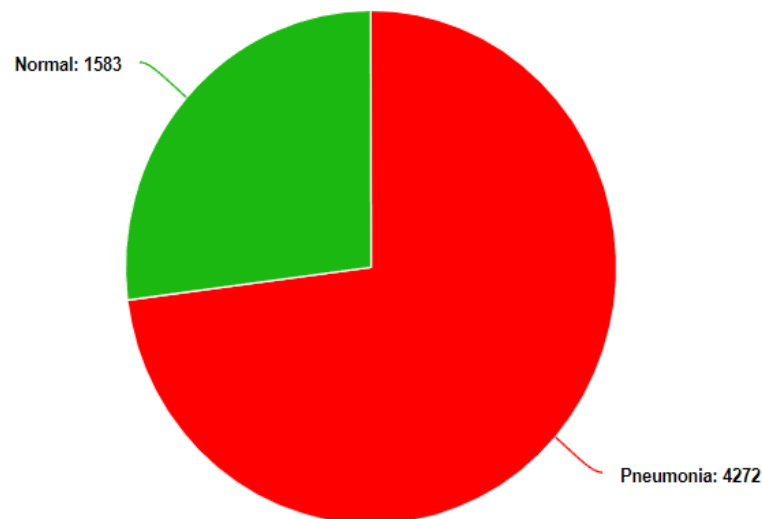


Figure 3 : La répartition des images en chaque classe

Ci-dessus, le diagramme circulaire représentant la répartition des classes sur le Dataset, avec 4272 images pour la classe « Pneumonie » et 1583 images pour la classe « Normal ».

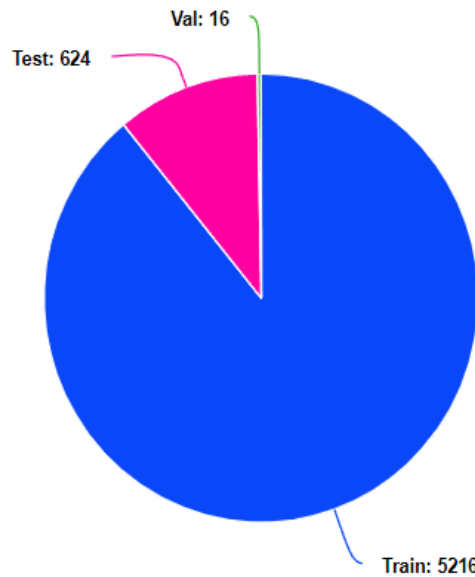


Figure 4: La répartition des images en chaque ensemble

Ci-dessus, la répartition initiale des train, test et val sets dans notre Dataset représenté dans un diagramme circulaire, avec : 5216 images pour le train set qui est représenté avec la couleur bleu ce qui semble suffisamment raisonnable, 624 images pour le test set représenté en couleur rose, et seulement 16 images pour le val test représenté en vert et qui apparaît à peine à cause de sa petite taille, la chose que nous avons considéré un problème à réparer avant de commencer à utiliser ces données.

En effet, lorsque l'ensemble de validation est très petit, cela peut poser un problème. Car le but de cet ensemble est d'évaluer les performances d'un modèle d'apprentissage automatique et de s'assurer que le modèle généralise bien sur les données jamais vues. Si cet ensemble est très petit, l'évaluation du modèle peut ne pas être fiable pour s'assurer de la performance du modèle, ce que nous a incité à penser à une autre division, la chose que nous allons traiter dans le titre suivant.

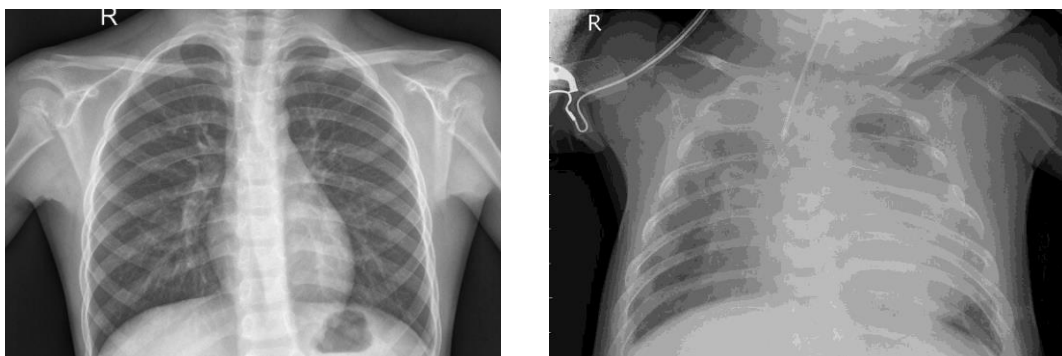


Figure 5 : Exemple d'image radiographique normale et affectée

Ci-dessus un échantillon des images X-Ray de notre Dataset, une qui représente une image d'une poitrine de la classe normale, et l'autre qui représente une poitrine affectée de la Pneumonie.

Comme il est illustré dans les deux figures ci-dessus, la principale différence entre une radiographie d'une poitrine affectée de la pneumonie et une poitrine normale est la présence d'opacités dans les poumons sur l'image de la poitrine affectée. Ces opacités sont causées par l'inflammation et l'accumulation de liquide dans les poumons, ce qui sont les caractéristiques de la pneumonie. En revanche, sur l'image d'une poitrine normale, les poumons apparaissent clairs et sans opacités anormales.

2. Problématique

Les données qu'on a trouvées sur Kaggle présentent certaines problèmes (de format, de qualité, de répartition ...etc.).

En effet, ces données sont de tailles et de dimensions différentes, donc il faut bien fixer leurs dimensions pour certaines raisons :

D'abord, les modèles d'apprentissage sont conçus pour fonctionner avec des données d'entrée de taille fixe, et si les dimensions des images varient d'une image à une autre cela peut ralentir l'entraînement et le rendre plus difficile.

Aussi, la fixation des dimensions des images permet de réduire la complexité du modèle et d'améliorer l'efficacité du processus d'entraînement. En ayant des images de taille fixe, le modèle peut être optimisé pour cette taille spécifique, ce qui permet une utilisation plus efficace des ressources informatiques.

A côté de ceci, nous avons remarqué que la répartition des images dans les dossiers n'est pas optimale, donc il est important d'avoir une bonne répartition des données entre les ensembles d'entraînement, de validation et de test avant de commencer tout processus pour plusieurs raisons :

Si l'ensemble d'entraînement (train set) est mal réparti, on risque de se heurter aux problèmes de sur ajustement ou sous ajustement (Overfitting - Underfitting).

L'ensemble de validation est utilisé pour évaluer les performances du modèle pendant l'entraînement. Il est généralement utilisé pour régler les hyperparamètres afin d'optimiser les

performances du modèle. Si cet ensemble est mal reparti, cela peut entrainer une évaluation peu fiable.

L'ensemble du test est utilisé pour évaluer les performances du modèle sur de nouvelles données. Si cet ensemble est mal reparti, cela peut entrainer des performances médiocres sur les nouvelles données, qui sont considéré très important pour mesurer l'utilité réelle du modèle final.

Un autre problème à traiter, est que les images de notre Dataset ne sont pas normalisées, ce qui peut poser des problèmes lors de l'entraînement. En effet, les valeurs des pixels non normalisées peuvent varier considérablement d'une image a une autre, ce qui peut ralentir la convergence. Ils peuvent aussi engendrer des biais dans les données, ce qui peut affecter la capacité du modèle à généraliser sur les nouvelles données.

Ainsi, comment pouvons-nous procéder pour trouver des solutions à tous les problèmes mentionnés afin de préparer les données d'une manière efficace pour garantir des résultats précis et améliorer la performance du modèle ?

3. Prétraitement des données (Data pre-processing)

Afin de bien préparer les données pour l'entraînement, nous avons appliqué quatre opérations sur les images de notre Dataset, notamment : la division du Dataset, le « Shuffle », le « Reshape » et la « normalisation »

3.1. La re-division du Dataset (Data Split)

Comme indiqué précédemment, bien que le Dataset ait déjà été divisé, nous avons décidé de créer une autre division qui serait plus adaptée à notre problématique.

Nous avons donc trié les images selon leurs label (Pneumonie/Normal) dans deux dossiers distincts. Ensuite, nous avons implémenté une fonction qui permet de diviser un Dataset passé en argument en trois ensembles (train, test, val) selon des pourcentages spécifiés en paramètres, comme indiqué clairement dans la figure ci-dessous. Les répartitions précédentes et nouvelles sont également présentées dans la figure.

```

data_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray'
# Define the directory paths for the normal and abnormal data
normal_dirs = ['train/NORMAL', 'test/NORMAL', 'val/NORMAL']
abnormal_dirs = ['train/PNEUMONIA', 'test/PNEUMONIA', 'val/PNEUMONIA']
# Create a new directory to store the transformed data
os.makedirs(os.path.join('/kaggle/working/new_data'), exist_ok=True)
# Loop through the normal and abnormal directories, and copy the images to the new directory
for directory in normal_dirs + abnormal_dirs:
    src_dir = os.path.join(data_dir, directory)
    filenames = os.listdir(src_dir)
    for filename in filenames:
        src_path = os.path.join(src_dir, filename)
        dst_path = os.path.join('/kaggle/working/new_data', directory.split('/')[1], filename)
        os.makedirs(os.path.dirname(dst_path), exist_ok=True)
        shutil.copy(src_path, dst_path)

```

Figure 6 : Code de réarrangement des données

La figure ci-dessous montre une portion de code qui explique comment nous avons réarranger les données dans 2 dossiers au lieu de l'ancienne division sur 3 dossiers avec 2 sous dossiers pour chacun.

```

def get_dataset_partitions_tf(ds , train_split=0.8 , val_split=0.1 , test_split=0.1 , shuffle=True, shuffle_size=1000):
    ds_size = len(ds)
    if shuffle :
        ds = ds.shuffle(shuffle_size , seed = 12)

    train_ds = ds.take(int(ds_size*train_split))
    val_ds = ds.take(int(ds_size*val_split))
    test_ds = ds.skip(len(val_ds)+len(train_ds))

    return train_ds , val_ds , test_ds

```

Figure 7 : Fonction de la répartition des données

Comme il est illustré dans la figure ci-dessus, la fonction `get_dataset_partitions_tf()` accepte 5 arguments : le Dataset à diviser, les pourcentages de chaque classe, une valeur booléenne pour appliquer le Shuffle ou non, et le Shuffle size.

```
train_ds , val_ds , test_ds = get_dataset_partitions_tf(dataset)
```

Figure 8 : L'appel de la fonction

Dans cette pièce de code, nous affectons aux 3 variables `train_ds`, `val_ds`, `test_ds`, les 3 valeurs retournées par la fonction `get_dataset_partition_tf()`.

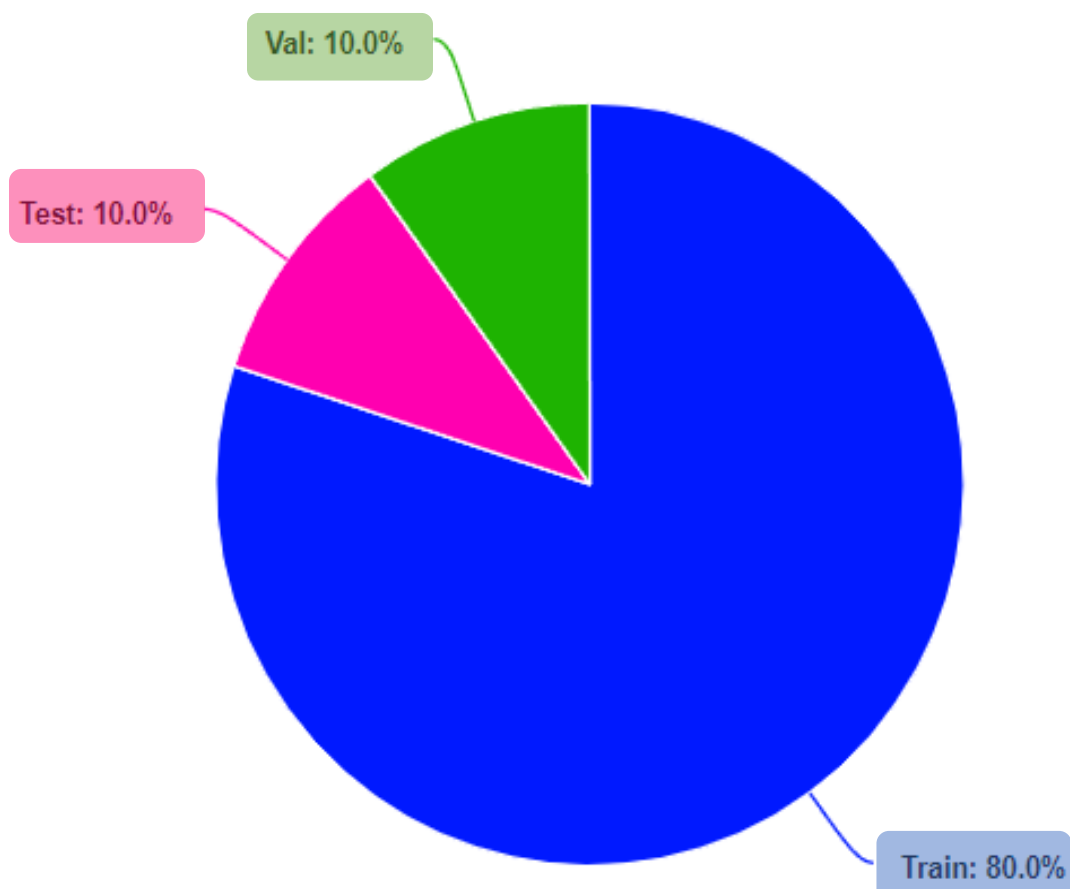


Figure 9: La nouvelle répartition des données

Après tous processus, cette figure montre la nouvelle répartition des données que nous avons considéré plus équilibrée.

3.2. Le « Shuffle »

C'est une opération qui permet de mélanger l'ordre des images dans un ensemble des données pour éviter que le modèle ne mémorise l'ordre des images pour ne pas affecter la qualité de son entraînement et par suite sa performance.

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Explication des méthodes utilisées :

- La méthode `cache()` permet de mettre en cache les données de chaque partie pour éviter de les recharger à chaque itération ceci va accélérer le processus.
- La méthode `prefetch()` permet de charger les données en arrière-plan dans le but de réduire le temps d'attente et minimiser l'usage de la mémoire.
- La méthode `Shuffle` est utilisée pour réarranger aléatoirement les images.

3.3. Le « Reshape »

C'est une opération qui permet de redimensionner les images pour leur fixer une seule taille, ce qui est nécessaire pour l'entraînement du modèle.

```
layers.experimental.preprocessing.Resizing(IMAGE_SIZE , IMAGE_SIZE),
```

Cette figure illustre le code que nous avons utilisé pour créer une couche qui permet de redéfinir les dimensions de chaque image, le paramètre `IMAGE_SIZE` est une constante définie à 224.

```

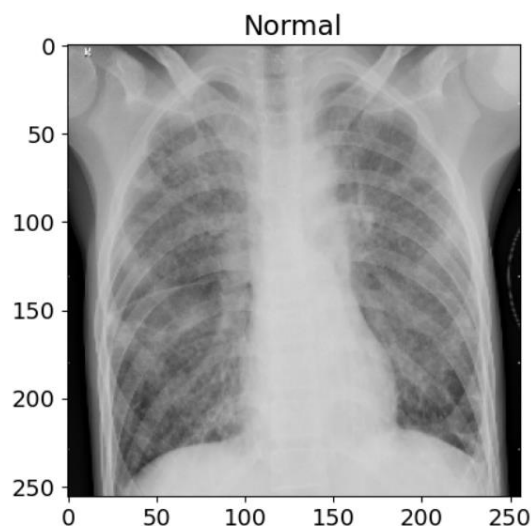
class_names = ["Pneumonia", "Normal"]

# Load the first image from train_ds
for images_batch, labels_batch in train_ds.take(1):
    image = images_batch[0].numpy().astype("uint8")
    label = labels_batch[0].numpy()

# Plot the image using plt
plt.figure(figsize=(5, 5))
plt.imshow(image, cmap='gray')
plt.title(class_names[label])

```

Dans cette figure, nous avons utilisé les méthodes « plt.imshow », « plt.figure », et numpy des bibliothèques Matplotlib et Numpy pour afficher la première image du Train Set du 1^{er} batch.



Cette figure montre que les dimensions des images du Dataset ont été fixées de 224x224 afin de limiter le taux de calcul et accélérer la procédure de l'entraînement en limitant le nombre total des inputs.

3.4. La normalisation

C'est une opération qui permet de standardiser les valeurs des pixels dans les images dans le but de réduire le taux de calculs nécessaires pour l'entraînement, de plus, les CNN convergent

plus rapidement sur des données comprises entre [0,1] que sur des données comprises entre [0,255].

```
layers.experimental.preprocessing.Rescaling(1.0/255)
```

4. Augmentation des données

Dans cette dernière étape de préparation des données nous avons appliqué l'augmentation des données « Data Augmentation ». Cette technique permet d'augmenter la taille de l'ensemble de données en appliquant un ensemble des modifications et des transformations aléatoires sur les images existantes. Pour cela, nous avons utilisé la bibliothèque Keras pour appliquer 2 transformations différentes : la rotation aléatoire et l'inversion horizontale ou verticale. Cette opération est assurée par une couche que nous allons l'ajouter ensuite lors de la définition de l'architecture de notre modèle.

Cette étape permettra à notre modèle d'apprendre des caractéristiques génériques plutôt que des caractéristiques spécifiques à notre ensemble de données, ce qui contribuera à améliorer sa performance, de plus, ceci va aider à éviter le problème de « Overfitting ».

```
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Cette couche séquentielle doit être implémentée en tant que première couche du modèle afin de générer plusieurs échantillons pour chaque image d'entrée. Pour un grand nombre d'images, lors de l'entraînement nous avons rencontré un problème de pénurie de mémoire vive (RAM).

Donc on a opté pour une autre méthode, c'est l'utilisation de « ImageDataClassification » :

```
def data_augmentation(img):
    resized = smart_resize(img, IMG_SIZE)
    scaled = per_image_standardization(resized)
    flipped = random_flip_left_right(scaled)
    rotated = rot90(flipped, k=uniform(shape=[], minval=0, maxval=4, dtype=int32))
    return img #the new generated image after after 4layers of mods
#this function replace the data_augmentation sequential layer
data_gen= ImageDataGenerator(preprocessing_function= data_augmentation)

train_gen = data_gen.flow_from_dataframe(train_ds, x_col= 'filepaths', y_col= 'labels', target_size= IMG_SIZE, class_mode= 'binary',
                                         color_mode= 'rgb', shuffle= True, batch_size= BATCH_SIZE)

valid_gen = data_gen.flow_from_dataframe(val_ds, x_col= 'filepaths', y_col= 'labels', target_size= IMG_SIZE, class_mode= 'binary',
                                         color_mode= 'rgb', shuffle= True, batch_size= BATCH_SIZE)

test_gen = data_gen.flow_from_dataframe(test_ds, x_col= 'filepaths', y_col= 'labels', target_size= IMG_SIZE, class_mode= 'binary',
                                        color_mode= 'rgb', shuffle= False, batch_size= test_batch_size)

Found 4684 validated image filenames belonging to 2 classes.
Found 703 validated image filenames belonging to 2 classes.
Found 469 validated image filenames belonging to 2 classes.
```

Nous avons défini une fonction data augmentation pour appliquer une augmentation de données à chaque image d'entrée. Cette fonction redimensionne l'image en utilisant la fonction smart_resize, normalise les valeurs de pixel en utilisant la fonction per_image_standardization, effectue une transformation de retournement horizontal aléatoire en utilisant la fonction random_flip_left_right, et effectue une rotation aléatoire en utilisant la fonction rot90. Cette fonction génère ainsi plusieurs images à partir de chaque image d'entrée.

Nous avons remplacé la couche séquentielle de data augmentation par l'objet ImageDataGenerator qui permet de générer dynamiquement des lots d'images avec augmentation de données lors de l'entraînement. Nous avons ainsi créé trois objets ImageDataGenerator pour les ensembles de données d'entraînement, de validation et de test en utilisant la fonction flow_from_dataframe.

Au final, nous avons obtenu 4684 images d'entraînement, 703 images de validation et 469 images de test avec augmentation de données à chaque époque d'entraînement. Cette méthode nous a permis d'augmenter efficacement la quantité de données d'entrée sans avoir à stocker toutes les images en mémoire vive.

Conclusion

Dans ce chapitre, nous avons présenté les données qu'on a utilisé, expliqué les opérations de prétraitement qu'on a appliqué telle que la re-division, le Shuffle, le Reshape et la normalisation. Nous avons aussi expliqué l'importance de l'augmentation des données dans l'amélioration de la qualité de notre Dataset. Dans le chapitre suivant, Nous nous intéresserons aux réseaux de neurones convolutionnels et ses différentes couches.

Chapitre 3. Modélisation

Introduction

Comme nous sommes face à un problème de classification d'images, plusieurs architectures peuvent être utilisés, notamment l'ANN, le DNN, le CNN, « Random Forest » ... etc.

Dans ce chapitre, on va expliquer notre choix de l'architecture CNN, puis on va décrire les différentes couches qui composent un CNN, telles que la couche de convolution, la fonction d'activations, le Pooling, le Flattening, les couches entièrement connectées et la façon de régler les hyperparamètres du modèle.

1. Choix de l'architecture CNN

Les CNN sont une architecture de réseau de neurones qui sont conçus pour traiter des images en 2D. Contrairement à un réseau de neurones classique qui utilise des couches entièrement connectées, les CNN utilisent des couches de convolution qui peuvent extraire des caractéristiques à partir des images en effectuant des opérations matricielles appelées convolutions.

Dans notre situation, les images X-Ray sont souvent grandes et de haute résolution, donc il sera difficile à les traiter avec des techniques d'apprentissage traditionnelles. Par contre, les CNN sont conçus pour gérer efficacement ces grandes images.

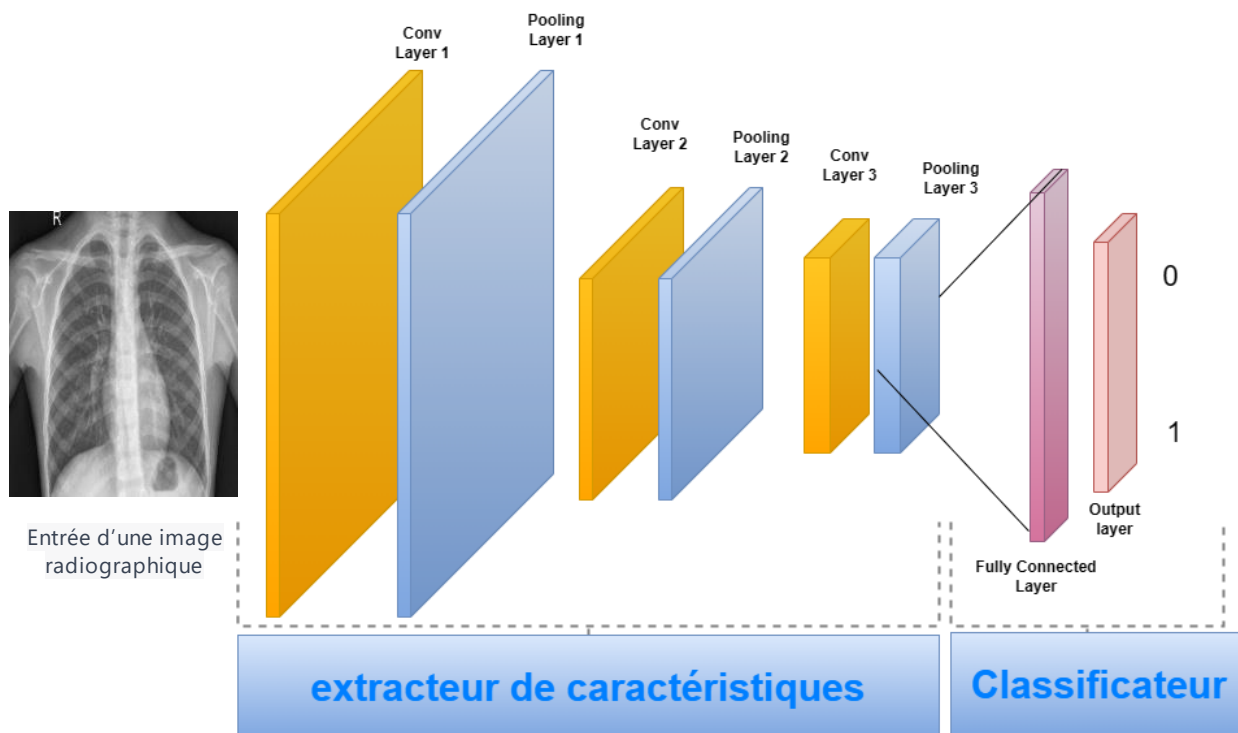


Figure 10 : Exemple d'architecture d'un simple CNN

Les CNN fonctionnent en apprenant à reconnaître des caractéristiques pertinentes dans une image à travers une série de couches de convolutions et une couche entièrement connectée.

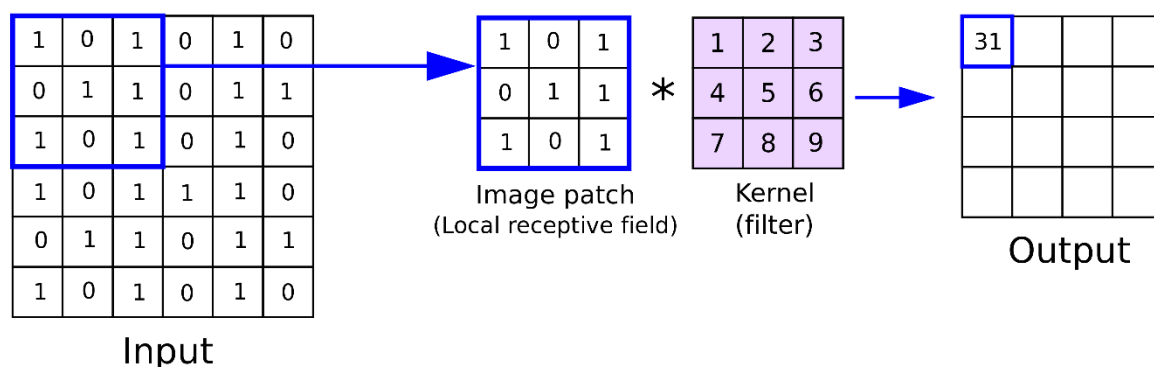
D'abord, les couches de convolutions extraient des caractéristiques de l'image d'entrée en appliquant un ensemble de filtres. Chaque filtre permet d'extraire une caractéristique différente, telle que les bords, les textures et les autres motifs, et produit ensuite une matrice de caractéristiques. Ensuite, les couches de regroupement ou « Pooling » réduisent les dimensionnalités des cartes, cela permet de réduire le nombre de paramètres et de calculs nécessaire. Enfin, les couches entièrement connectées utilisent les caractéristiques extraites pour faire une prédiction sur l'image donnée. Ces couches connectent chaque neurone d'une couche à chaque neurone de la couche suivante permettant au réseau d'apprendre des relations complexes entre les « Features ».

Au cours de l'entraînement, le CNN ajuste les poids des filtres et des neurones dans les couches pour minimiser le taux d'erreur (Loss) entre les prédictions et les étiquettes des classes (Labels) associée aux images. En ce qui suit nous allons traiter chaque étape d'une manière explicite.

2. La couche de convolution

Chaque image passée comme input est transformée en une matrice avec des dimensions égales à celles de l'image, où chaque valeur de la matrice représente l'intensité de couleur d'un pixel spécifique.

La convolution est une technique fondamentale utilisée dans les réseaux de neurones convolutifs (CNN) pour extraire des caractéristiques importantes [Special Features] des images. Elle consiste à appliquer un filtre, appelé aussi Kernel, qui est une petite matrice de poids qui est utilisé pour scanner une image d'entrée et effectuer une opération mathématique appelé « convolution ». Le but du filtre est d'extraire les caractéristiques importantes de l'image passée en input.



Le filtre à appliquer dépend du type du problème, des formes et des localisations des motifs qu'on cherche à identifier...etc.

Pendant le processus de convolution, le filtre est glissé sur l'image, pixel par pixel, et un produit scalaire est calculé entre les valeurs dans le filtre et les valeurs de pixel correspondantes dans la matrice de l'image. Le résultat de cette opération est une nouvelle matrice des valeurs, qui représente la sortie de la couche de convolution. Puis il se produit une sommation du résultat afin d'avoir finalement une seule valeur. Ce processus est répété sur toute la matrice avec un pas fixé dès le début, permettant de créer enfin une matrice de dimension plus petits que la matrice de l'image, cette matrice présente les zones de l'image qui contiennent des formes spécifiques (ou des motifs).

3. La fonction d'activation

Sur chaque sortie de couche de convolution, on va appliquer une fonction d'activation.

La fonction d'activation est une fonction mathématique, son rôle majeur est d'introduire la non-linéarité dans le modèle.

Sans fonction d'activation, on n'aura à la sortie du modèle que des combinaisons linéaires des entrées et des poids (Weights), ce qui ne va pas nous servir dans la plupart des problèmes du monde réel, plus précisément dans notre situation, ou le modèle devrait capturer des relations non linéaires entre les caractéristiques.

Dans notre cas, nous avons utilisé les fonctions « ReLU » et « Sigmoid » et « Softmax » qui sont deux fonctions d'activation qui sont largement utilisées dans les CNN et dans les problèmes de classification binaire.

On a utilisé la fonction « Sigmoid » dans la dernière couche pour estimer la probabilité d'appartenance d'une entrée à une des deux classes, tandis que la fonction « ReLU » a été incluse dans les couches cachées [Hidden layers] pour éviter le problème de la disparition du gradient.

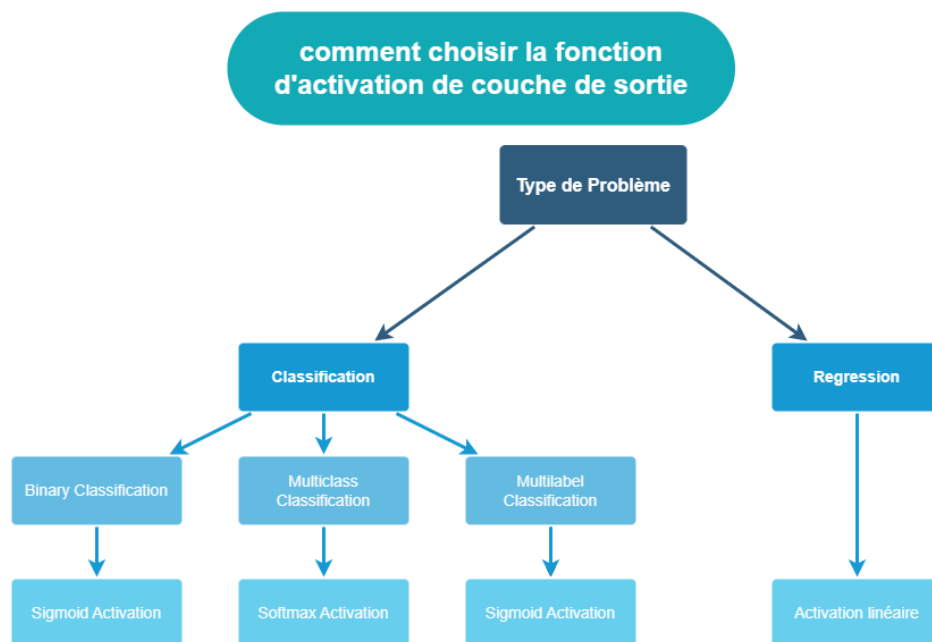
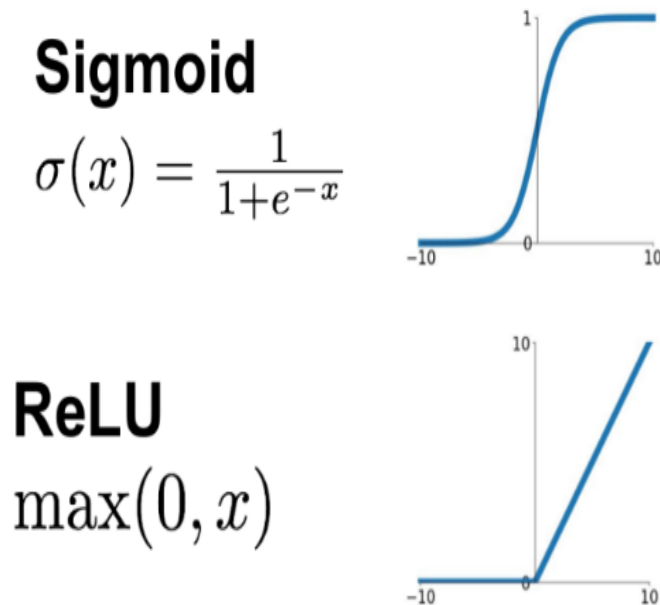


Figure 11 : Choix de la fonction d'activation

La figure ci-dessus montre comment la fonction d'activation dépend du type du problème. Dans notre cas nous avons recourt à la fonction sigmoïde puisqu'il s'agit d'un problème de classification binaire.

4. La couche d'agrégation « Pooling »

L'étape suivante est le « Pooling », son rôle est de réduire la taille de la matrice de l'image mais en conservant les pixels qui représentent les caractéristiques les plus importantes.

Cette opération consiste à diviser la matrice en des régions, et la réduire en une seule valeur dans le but de réduire le nombre de paramètres et de calculs nécessaires pour entrainer le modèle.

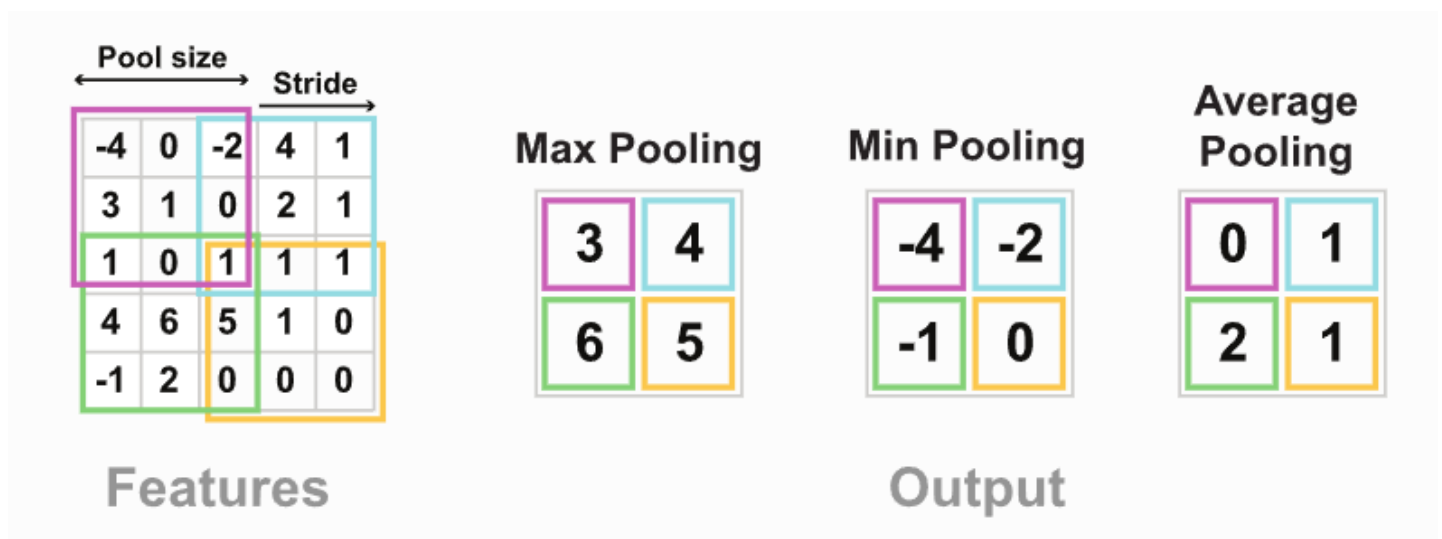


Figure 12 : Différents types de Pooling

La figure ci-dessous montre qu'il y a plusieurs types de « Pooling » notamment le « Max Pooling » et c'est celui-ci qu'on a utilisé, il en existe d'autres comme le « Average Pooling », « Min Pooling ».

5. La couche d'aplatissement « Flattening layer »

Après avoir soumis l'entrée à plusieurs couples de couches (couche de convolution, couche de Max Pooling) le résultat sera après soumis à une autre opération fondamentale nommée « Flattening » ou l'aplatissement, cette opération a pour but de convertir la matrice multidimensionnelle résultante en un vecteur unidimensionnel. Dans la figure ci-dessous, cette couche est marquée en vert.

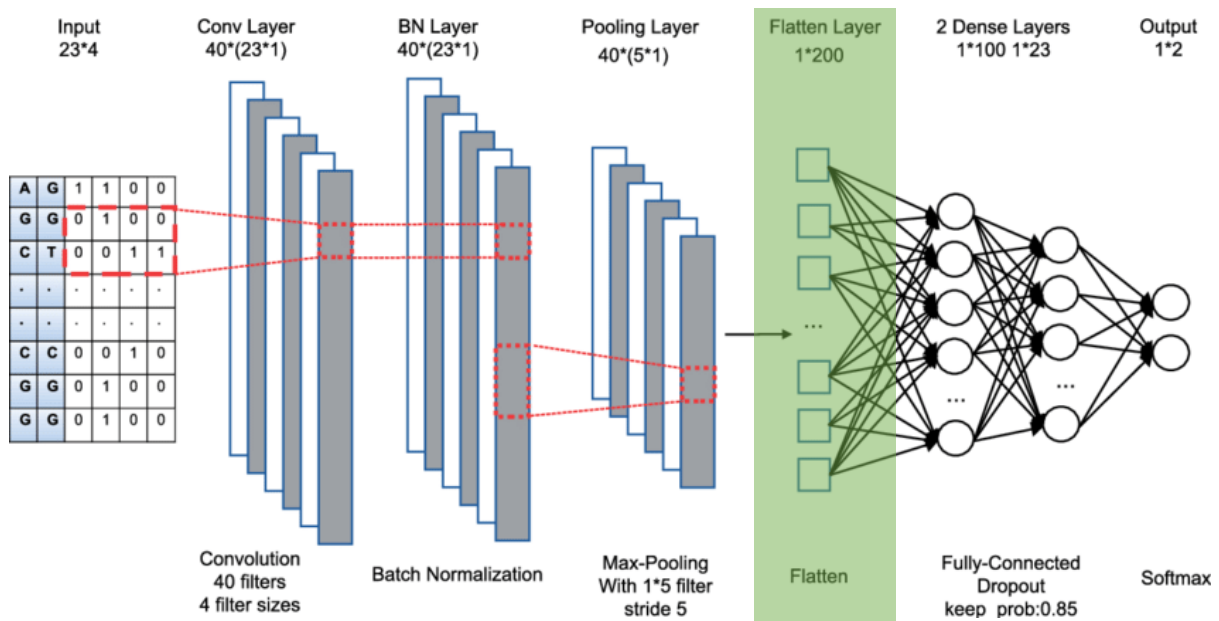


Figure 13 : Couche d'aplatissement

6. La couche entièrement connectée « Dense layer »

Finalement, le vecteur unidimensionnel sera passé à travers une couche entièrement connectée [Fully Connected Layer], où chaque neurone est connecté à tous les autres neurones de la couche précédentes, et chaque connexion est associée à un poids qui représente l'importance de l'entrée correspondante à la sortie.

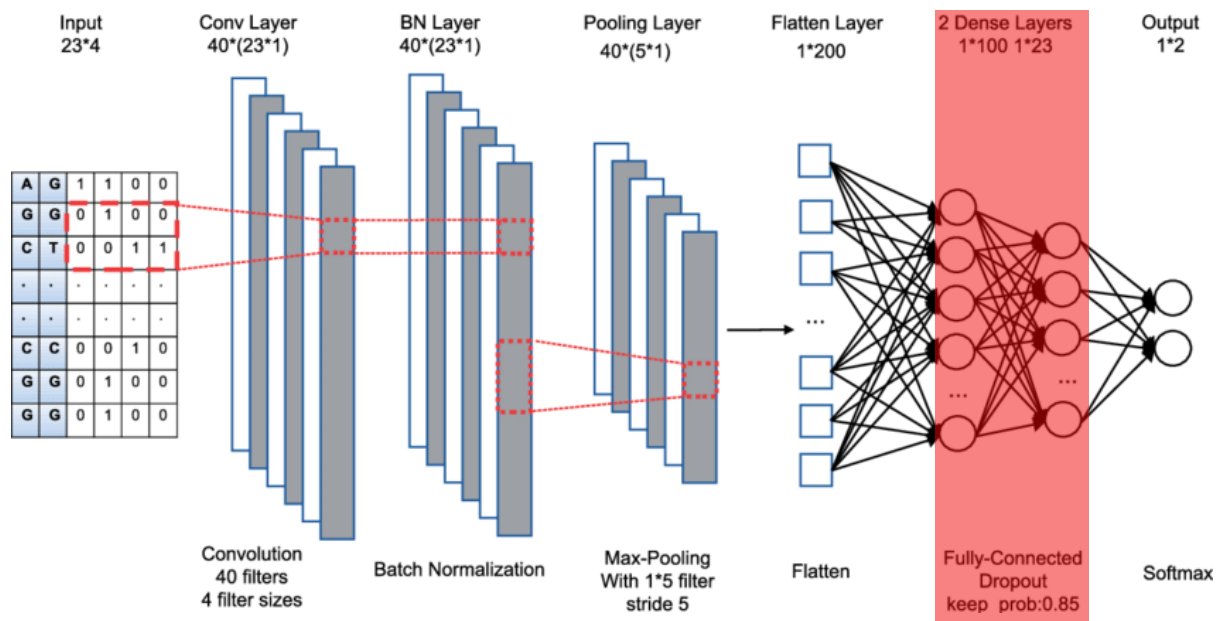


Figure 14 : couche entièrement connectée

La figure ci-dessous montre l'utilisation de cette couche pour produire une sortie finale représentée par une distribution des probabilités sur chacune de deux classes : Normal et Pneumonie.

7. La couche Dropout

La couche Dropout est une technique de régularisation couramment utilisée dans les réseaux de neurones afin de prévenir le surapprentissage (Overfitting). Elle consiste à supprimer aléatoirement un pourcentage de neurones dans la couche pendant la phase d'entraînement.

En pratique, à chaque « Epoch », chaque neurone dans la couche Dropout est désactivé avec une probabilité donnée. Cette approche garantit que le signal qui traverse le réseau suit des chemins différents à chaque étape, empêchant ainsi les neurones de devenir trop co-dépendants et permettant au modèle d'obtenir une meilleure généralisation.

8. La couche de « Batch normalization »

La couche « Batch Normalization » est une technique largement utilisée dans les réseaux de neurones profonds pour améliorer la stabilité et la rapidité de l'apprentissage. Elle consiste à normaliser les activations de chaque couche en se basant sur les statistiques des batches d'entraînement. Dans chaque mini-batch d'entraînement, les activations de chaque couche sont centrées et réduites en se basant sur la moyenne et l'écart-type des activations dans le mini-batch. Cette normalisation permet de rendre les activations de chaque couche plus stables et moins susceptibles de diverger pendant l'apprentissage, améliorant ainsi la vitesse et la stabilité de l'apprentissage. [38]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1, \dots, x_m\}$; Parameters to be learned: γ, β	
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$	// scale and shift

9. Les métriques d'évaluation (Evaluation Metrics)

Dans le domaine de l'apprentissage en profondeur (Deep Learning), les métriques sont des unités utilisées pour mesurer la qualité de la performance d'un modèle de réseau de neurones. Ces métriques permettent d'évaluer la précision et la performance d'un modèle de réseau de neurones par rapport à des données d'entraînement ou de validation.

9.1. Précision

Dans un problème de classification binaire déséquilibré comme dans notre cas, la précision est calculée en divisant le nombre de vrais positifs par la somme du nombre de vrais positifs. Cette mesure donne une valeur comprise entre 0.0 pour aucune précision et 1.0 pour une précision parfaite. La précision est une métrique importante pour

l'évaluation de la performance du modèle, par contre elle peut être parfois trompeuse, dans ce cas il faut considérer utiliser d'autres métriques.

$$precision = \frac{TP}{TP + FP}$$

9.2. Recall

Le recall est une mesure qui quantifie le nombre de prédictions positives correctes effectuées parmi toutes les prédictions positives qui auraient pu être faites. Contrairement à la précision qui ne commente que sur les prédictions positives correctes parmi toutes les prédictions positives, le recall fournit une indication des prédictions positives manquées. [39]

$$recall = \frac{TP}{TP + FN}$$

9.3. F1 score

La métrique F1 est une mesure de la performance d'un modèle qui combine la précision et le recall en une seule valeur. Elle est définie comme la moyenne harmonique de la précision et du recall, où la précision mesure la proportion de prédictions positives correctes parmi toutes les prédictions positives, et le recall mesure la proportion de prédictions positives correctes parmi toutes les instances positives réelles. La métrique F1 est donc une mesure équilibrée qui prend en compte à la fois la précision et le recall, et elle est souvent utilisée pour évaluer des modèles de classification binaire.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

9.4. Accuracy

L'accuracy est une métrique fondamentale pour évaluer la performance d'un modèle de Deep Learning. Elle mesure la proportion de prédictions correctes parmi toutes les prédictions effectuées, ce qui permet de déterminer à quel point le modèle est précis dans sa classification. Cependant, l'accuracy ne prend pas en compte les erreurs de classification pour chaque classe, elle ne donne donc pas une vision complète de la qualité du modèle.

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Conclusion

Après avoir justifié le choix du réseau de neurone convolutif, présenté ses différentes notions, tels que la couche de convolution, Pooling, Flattening, Dense layer...etc, le prochain chapitre sera concentré sur la mise en œuvre pratique de ces composants, notamment leur déploiement et leur évaluation.

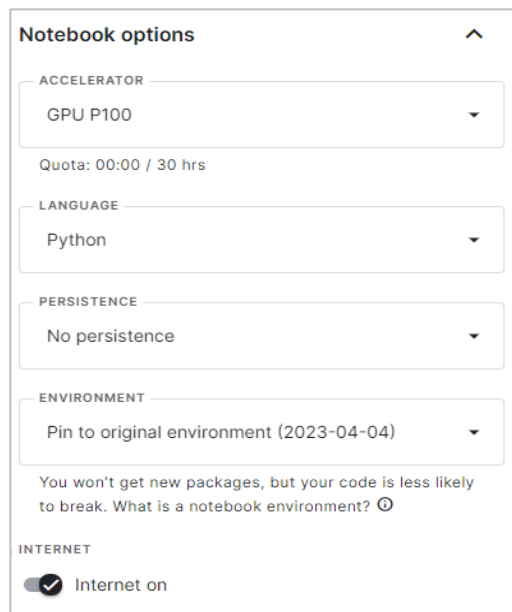
Chapitre 4. Evaluation et déploiement

Introduction

La phase d'évaluation et déploiement est une étape clé qui permet de valider les résultats obtenus et de mettre en œuvre. Dans ce chapitre, nous allons présenter les différentes étapes que nous avons suivie dans cette dernière phase. Nous ferons d'abord une étude technique dans laquelle nous présenterons les outils et des données utilisés, puis nous allons décrire la construction, la compilation, l'entraînement et l'évaluation du modèle. Ensuite, nous allons expliquer le déploiement de notre modèle sur une plateforme web. Nous allons aussi effectuer une comparaison de l'efficacité de notre solution avec celles d'autres solutions existantes et nous allons finir par une conclusion sur les principaux résultats.

1. Etude technique

Ce projet a été réalisé en utilisant principalement le « **Kaggle Notebook** » qui est une plateforme en ligne qui permet d'exécuter du code Python. Nous avons choisi d'utiliser le GPU dans notre projet afin d'accélérer le temps d'apprentissage de notre modèle et d'améliorer ses performances. Dans ce contexte, nous avons opté pour le **NVIDIA TESLA P100 GPU**, qui offre des performances élevées et permet de traiter les tâches les plus complexes avec une grande efficacité.



The image shows the 'Notebook options' panel in the Kaggle interface. It contains several settings:

- ACCELERATOR:** Set to 'GPU P100'. Below it, the quota is shown as 'Quota: 00:00 / 30 hrs'.
- LANGUAGE:** Set to 'Python'.
- PERSISTENCE:** Set to 'No persistence'.
- ENVIRONMENT:** Set to 'Pin to original environment (2023-04-04)'. Below this, a note states: 'You won't get new packages, but your code is less likely to break. What is a notebook environment? ⓘ'.
- INTERNET:** A toggle switch is turned on, labeled 'Internet on'.

Figure 15 : Paramètres du bloc-notes Kaggle

```






▶ device_name = tf.test.gpu_device_name()
if not device_name:
    print("GPU device not found")
else:
    print('Found GPU at: {}'.format(device_name))

```

Found GPU at: /device:GPU:0

Figure 16 : Test de périphérique GPU

Dans le tableau ci-dessous, nous avons mentionner les logiciels et bibliothèques utilisés pour implémenter l'approche proposée sont tous présentés dans le tableau en dessous :

Logiciel	Logo	Description
Kaggle Notebook		Kaggle Notebook est une plateforme en ligne qui permet de créer, partager et exécuter des codes en Python. Il s'agit d'un IDE basé sur le cloud, qui offre une grande flexibilité pour travailler sur des projets de data science en équipe ou en solo.
Jupyter Notebook		Jupyter Notebook est une application client-serveur créée par l'organisation à but non lucratif Project Jupyter pour l'écriture de code en Python, R et d'autres langages de programmation.
Python 3.11.2		Python est le langage de programmation de haut niveau utilisé pour développer des applications dans divers domaines, y compris l'apprentissage automatique.
Scikit-learn 1.2.2		Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique [40]. Elle comprend des outils pour la classification, la régularisation, la réduction de la dimensionnalité, le clustering, la préparation de données et bien plus encore.
TensorFlow v2.12.0		TensorFlow est une bibliothèque open source de Machine Learning développée par Google [41] . Elle permet de construire des modèles d'apprentissage profond pour la classification, la reconnaissance d'image, la détection d'objet, la prédiction de séries chronologiques et bien plus encore.

Keras		Keras est une bibliothèque open source de réseaux de neurones destinée à l'apprentissage profond, qui est intégrée à TensorFlow. Elle permet de construire des modèles d'apprentissage profond à l'aide de couches préconfigurées pour divers types de tâches, telles que la classification d'image, la prédiction de séries chronologiques, etc.
Pillow (PIL Fork) 9.5.0		Pillow est une bibliothèque de traitement d'image open source qui ajoute des fonctionnalités de traitement d'image à Python.
Flask		Flask est un micro Framework open-source de développement web en Python. Il permet de développer des applications web rapidement et facilement.
PythonAnywhere by ANACONDA		PythonAnywhere est une plateforme de développement et d'hébergement web en ligne pour les applications Python. Elle permet d'héberger des applications web et de travailler sur des projets Python à distance.

Tableau 2 : Environnement technique utilisé pour la mise en œuvre de l'approche proposée

2. Construction des modèles

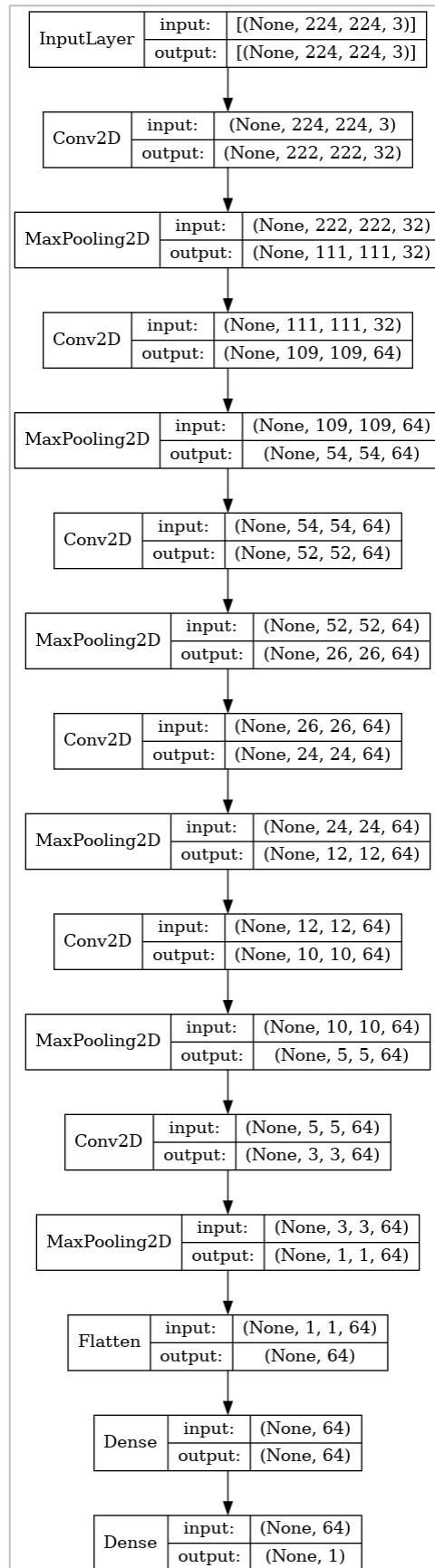
Pour construire un modèle, il faut commencer par définir les couches qui composent le réseau de neurones convolutionnels, ainsi que les fonctions d'activation et la fonction de coût.

Dans notre cas, nous avons décidé de construire deux modèles distincts en utilisant deux techniques différentes afin de comparer leurs efficacités. Le premier modèle est construit à partir d'un CNN classique, par contre, pour le deuxième modèle on a utilisé la méthode de « Transfer Learning » qui introduit le modèle pré-entraîné « EfficientNetB0 ».

2.1. Modèle CNN classique

Ce modèle est construit en utilisant des différentes couches pour extraire les caractéristiques des images de radiographie thoracique. Le modèle comprend plusieurs paires de couches Conv2D et MaxPooling2D pour extraire progressivement les caractéristiques les plus importantes de l'image. Ensuite, les caractéristiques extraites sont aplaties et passées à travers deux couches Dense pour la classification. Le modèle est ensuite compilé en utilisant la fonction

de perte (Loss Function) "Binary Crossentropy" et l'optimiseur (Optimizer) "Adamax" avec un taux d'apprentissage de 0,001. La performance du modèle est évaluée en utilisant la métrique (Metrics) "Accuracy".



Total params: 171,329
Trainable params: 171,329
Non-trainable params: 0

Figure 17 : Paramètres du premier modèle

Figure 18 : Architecture du premier modèle

Voici une illustration du code que nous avons implémenté pour créer ce modèle :

```
model1 = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3),activation='relu' ),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, kernel_size = (3,3),activation='relu' ),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3),activation='relu' ),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3),activation='relu' ),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3) ,activation='relu' ),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(class_count,activation='sigmoid'),
])

model1.build(input_shape=input_shape) # Build the model

model1.compile(Adamax(lr=0.001), loss= 'binary_crossentropy', metrics= ['accuracy'])

model1.summary()
```

Figure 19 : Code du premier modèle

2.2. Modèle par Transfert Learning

Pour la création de ce modèle, nous avons utilisé un modèle pré-entraîné « EfficientNetB0 » pour extraire les caractéristiques des images de radiographie thoracique. Nous avons ajouté une couche de normalisation par lots, une couche Dense avec une régularisation L1 et L2 et une couche « Dropout » pour réduire le risque de sur-apprentissage (Overfitting). Le modèle est compilé ensuite en utilisant les mêmes paramètres.

```
Total params: 4,382,884
Trainable params: 4,338,301
Non-trainable params: 44,583
```

Figure 20: Paramètres du deuxième modèle

Voici une illustration du code que nous avons écrit pour créer ce modèle :

```
base_model = efficientnet.EfficientNetB0(include_top=False, weights="imagenet", input_shape=input_shape, pooling='max')
# base_model.trainable = False

model2 = Sequential([
    base_model,
    BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001),
    Dense(256, kernel_regularizer=regularizers.l2(l=0.016), activity_regularizer=regularizers.l1(0.006),
        bias_regularizer=regularizers.l1(0.006), activation='relu'),
    Dropout(rate=0.45, seed=123),
    Dense(class_count, activation='sigmoid')
])

model2.compile(Adamax(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

model2.summary()
```

Figure 21 : Création du deuxième modèle

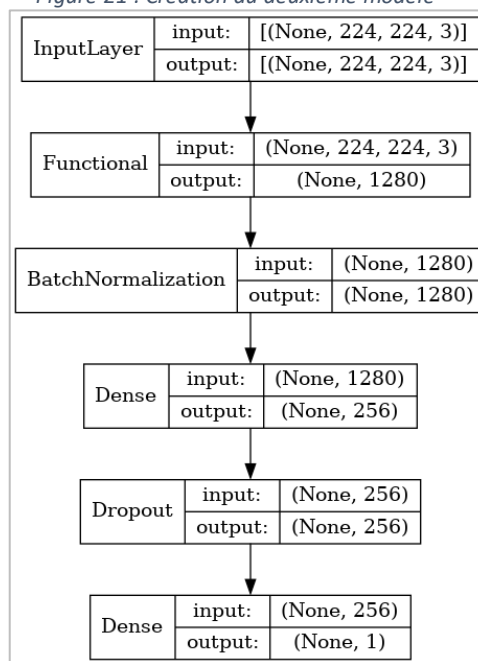


Figure 22 : Architecture du deuxième modèle

2.2.1. Choix du modèle pré-entraîné

« EfficientNet » est un modèle de réseau de neurones relativement nouveau qui a été introduit en 2019. Le but d'EfficientNet était d'améliorer l'efficacité et la précision des réseaux de neurones convolutifs (CNN) en équilibrant la profondeur, la largeur et la résolution du réseau. Pour atteindre cet objectif, EfficientNet utilise une méthode d'échelle novatrice qui uniformise les trois dimensions de la profondeur, la largeur et la résolution en utilisant un coefficient composé qui peut être facilement réglé. Cette méthode est conçue pour équilibrer le compromis entre la précision du modèle et les ressources computationnelles, permettant aux utilisateurs de choisir le modèle de la bonne taille pour leur application spécifique. EfficientNet a atteint des

résultats de pointe sur le jeu de données ImageNet. Le plus grand modèle EfficientNet, EfficientNet-B7, a atteint une précision top-1 de 84,3% sur ImageNet, tout en étant 8,4 fois plus petit et 6,1 fois plus rapide que le meilleur CNN existant, « GPipe ». [41]

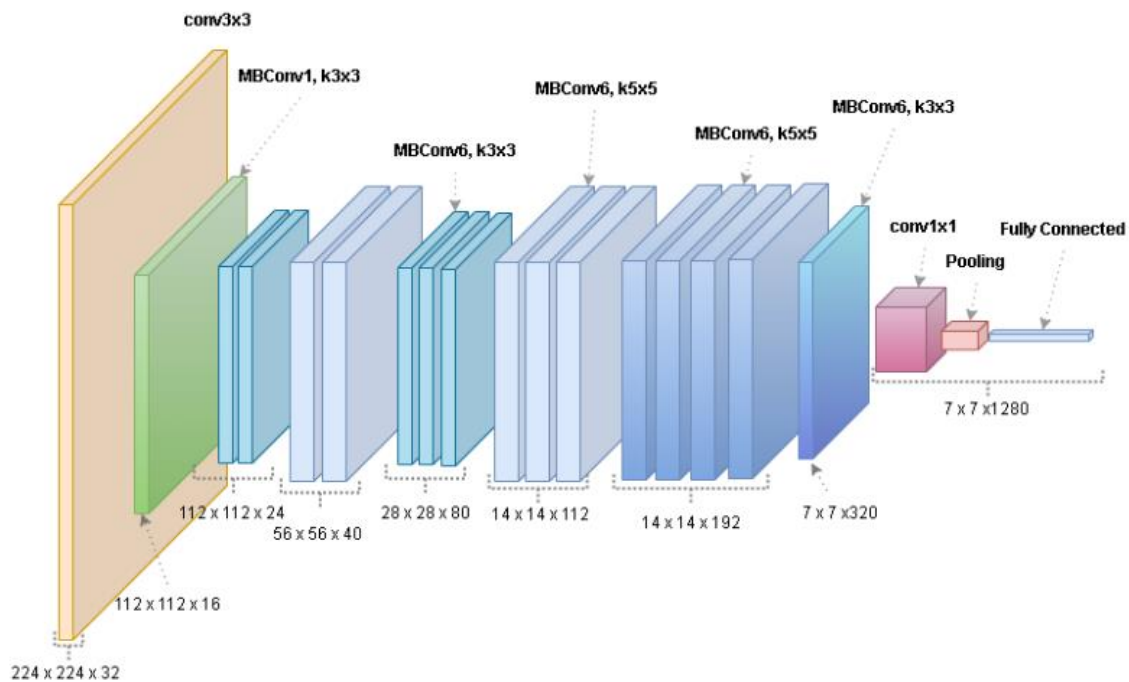


Figure 23 : Architecture de EfficientNetB0

Nous avons décidé de choisir ce modèle car il a été spécifiquement conçu pour être efficace en termes de taille et de vitesse d'entraînement et aussi pour être efficace et à faible complexité par rapport aux autres modèles pré-entraînés et cela le rend très adapté ce type de problème.

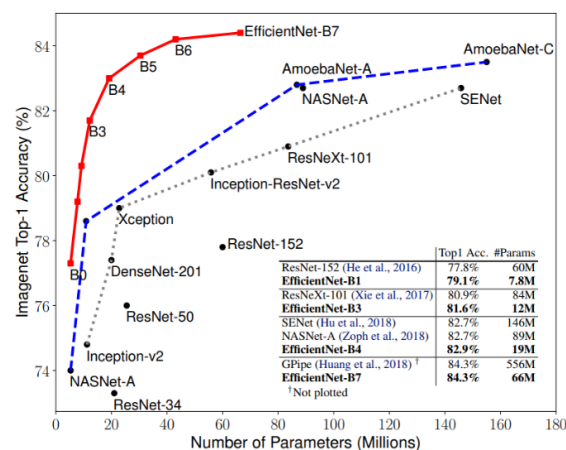


Figure 24 : Comparaison de la précision top-1 d'EfficientNetB0 avec d'autres modèles en fonction du nombre de paramètres

Dans la suite, nous allons détailler les performances de ces deux modèles et les résultats obtenus lors de l'entraînement et de la validation.

3. Entraînement des modèles

```
history1 = model1.fit(x= train_gen, epochs= EPOCHS, verbose= 1, validation_data= valid_gen,  
                     validation_steps= None, shuffle= False, callbacks = [learning_rate_reduction])
```

Figure 25 : L'entraînement du premier modèle

La méthode `fit()` de l'objet `model1` est utilisée pour entraîner le modèle. Elle prend en entrée les données d'entraînement suivantes :

- Le nombre d'époques à entraîner (`EPOCHS=20`)
- Le mode de verbosité (`verbose`)
- Les données de validation (`validation_data`)
- Le nombre d'étapes de validation (`validation_steps`)
- La méthode de mélange (`shuffle`)
- Une liste d'objets de rappel (`callbacks`).

Les données d'entraînement et de validation sont fournies sous forme de générateurs (`train_gen` et `valid_gen`), les données de validation sont utilisées pour évaluer les performances du modèle à chaque époque. Le mode de verbosité est défini sur 1, ce qui permet de suivre la progression de l'entraînement en temps réel. Le nombre d'étapes de validation est défini sur `None`, ce qui signifie que toutes les données de validation sont utilisées pour une seule évaluation. La méthode de mélange (`shuffle`) est définie sur `False`, ce qui signifie que les données ne sont pas mélangées avant chaque époque. Enfin, un objet de rappel (`learning_rate_reduction`) est fourni en tant que liste, pour réduire le taux d'apprentissage lorsque la performance du modèle sur les données de validation cesse de s'améliorer.

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1, factor=0.3, min_lr=0.000001)
```

Figure 26 : La fonction de rappel `ReduceLROnPlateau`

« `ReduceLROnPlateau` » est une fonctionnalité de rappel (callback) de « Keras » qui permet de réduire le taux d'apprentissage (Learning Rate) du modèle lorsque la métrique de validation (monitor) cesse de s'améliorer pendant un certain nombre d'époques consécutives (Patience).

Nous avons utilisé cette fonctionnalité en spécifiant "val_accuracy" comme métrique de validation à surveiller. Ainsi, lorsque la précision de validation cesse de s'améliorer pendant 2 époques consécutives, le taux d'apprentissage sera réduit de 30% (factor=0.3). Cette réduction continue jusqu'à atteindre une limite minimale de 0.000001 (min_lr=0.000001).

L'utilisation de cette fonctionnalité est importante car elle permet d'éviter que le modèle ne reste coincé dans un minimum local ou ne commence à surapprendre (Overfitting). En réduisant le taux d'apprentissage, le modèle est capable d'ajuster les poids du réseau plus délicatement et avec plus de précision, ce qui peut aider à sortir du minimum local et à améliorer la précision du modèle sur les données de validation.

```
Epoch 1/20
293/293 [=====] - 88s 276ms/step - loss: 0.4083 - accuracy: 0.8367 - val_loss: 0.2687 - val_accuracy: 0.8905 - lr: 0.0010
Epoch 2/20
293/293 [=====] - 80s 272ms/step - loss: 0.2115 - accuracy: 0.9206 - val_loss: 0.2027 - val_accuracy: 0.9090 - lr: 0.0010
Epoch 3/20
293/293 [=====] - 81s 276ms/step - loss: 0.1870 - accuracy: 0.9302 - val_loss: 0.1855 - val_accuracy: 0.9232 - lr: 0.0010
Epoch 17: ReduceLROnPlateau reducing learning rate to 2.70000040931627e-05.
293/293 [=====] - 80s 272ms/step - loss: 0.0091 - accuracy: 0.9987 - val_loss: 0.1713 - val_accuracy: 0.9573 - lr: 9.0000e-05
Epoch 18/20
293/293 [=====] - 80s 272ms/step - loss: 0.0078 - accuracy: 0.9989 - val_loss: 0.1750 - val_accuracy: 0.9559 - lr: 2.7000e-05
Epoch 19/20
293/293 [=====] - ETA: 0s - loss: 0.0075 - accuracy: 0.9989
Epoch 19: ReduceLROnPlateau reducing learning rate to 8.10000013655517e-06.
293/293 [=====] - 79s 271ms/step - loss: 0.0075 - accuracy: 0.9989 - val_loss: 0.1789 - val_accuracy: 0.9573 - lr: 2.7000e-05
Epoch 20/20
293/293 [=====] - 80s 272ms/step - loss: 0.0070 - accuracy: 0.9991 - val_loss: 0.1797 - val_accuracy: 0.9573 - lr: 8.1000e-06
```

Figure 27 : Résultat de l'entraînement du premier modèle

```
Epoch 17: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
293/293 [=====] - 91s 309ms/step - loss: 0.1324 - accuracy: 0.9983 - val_loss: 0.1726 - val_accuracy: 0.9772 - lr: 8.1000e-06
Epoch 18/20
293/293 [=====] - 93s 317ms/step - loss: 0.1323 - accuracy: 0.9981 - val_loss: 0.1688 - val_accuracy: 0.9787 - lr: 2.4300e-06
Epoch 19/20
293/293 [=====] - ETA: 0s - loss: 0.1296 - accuracy: 0.9996
Epoch 19: ReduceLROnPlateau reducing learning rate to 1e-06.
293/293 [=====] - 92s 313ms/step - loss: 0.1296 - accuracy: 0.9996 - val_loss: 0.1708 - val_accuracy: 0.9772 - lr: 2.4300e-06
Epoch 20/20
293/293 [=====] - 92s 313ms/step - loss: 0.1334 - accuracy: 0.9974 - val_loss: 0.1722 - val_accuracy: 0.9772 - lr: 1.0000e-06
```

Figure 28 : Résultat de l'entraînement du deuxième modèle

4. Evaluation des modèles

Training History for Model 1



Training History for Model 2

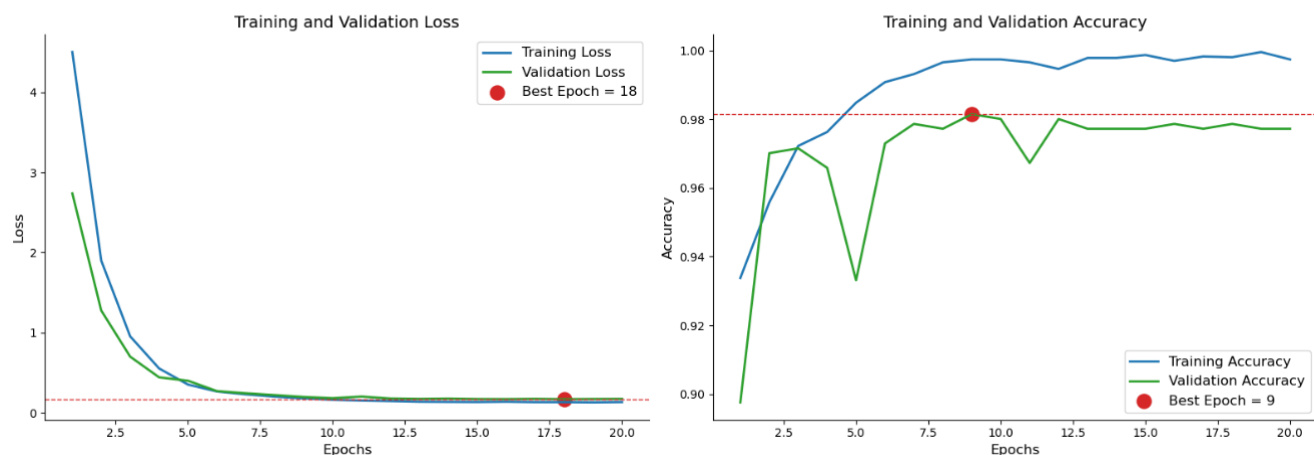


Figure 29 : courbes de perte et de précision de l'entraînement et de la validation pour les deux modèles

En comparant les résultats des deux modèles, nous pouvons remarquer que le modèle 2 a une précision légèrement supérieure au modèle 1 sur le jeu de données de validation (95,73 % atteinte dans la treizième époque contre 98,16 % atteinte dans la neuvième époque). Cependant, les deux modèles ont montré une capacité excellente à généraliser sur l'ensemble de données de test, avec une précision qui s'approche beaucoup de 100%.

En examinant les graphiques de la perte d'apprentissage et de validation pour les deux modèles, nous pouvons voir que le modèle 1 atteint une perte et une précision de validation beaucoup plus rapidement que le modèle 2. De plus, le modèle 2 continue à améliorer sa perte de validation pendant les dernières époques d'apprentissage. En revanche, le modèle 1 atteint plus facilement un meilleur résultat pour la perte seulement en treize époque.

Il faut signaler aussi que le modèle 2 a nécessité moins de temps d'entraînement que le modèle 1 pour atteindre sa précision maximale (seulement neuf époques).

En conclusion, le modèle 2 est plus performant que le modèle 1 en termes de généralisation et de vitesse de convergence. Il a réussi à obtenir une précision de 98,24 % sur le test Dataset, ce qui est très satisfaisant.

```
train_score1 = model1.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score1 = model1.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score1 = model1.evaluate(test_gen, steps= test_steps, verbose= 1)

7/7 [=====] - 2s 236ms/step - loss: 0.0055 - accuracy: 1.0000
7/7 [=====] - 2s 272ms/step - loss: 0.1673 - accuracy: 0.9643
7/7 [=====] - 7s 998ms/step - loss: 0.1647 - accuracy: 0.9424

train_score2 = model2.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score2 = model2.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score2 = model2.evaluate(test_gen, steps= test_steps, verbose= 1)

7/7 [=====] - 2s 255ms/step - loss: 0.1214 - accuracy: 1.0000
7/7 [=====] - 2s 212ms/step - loss: 0.1444 - accuracy: 0.9911
7/7 [=====] - 8s 1s/step - loss: 0.1972 - accuracy: 0.9744
```

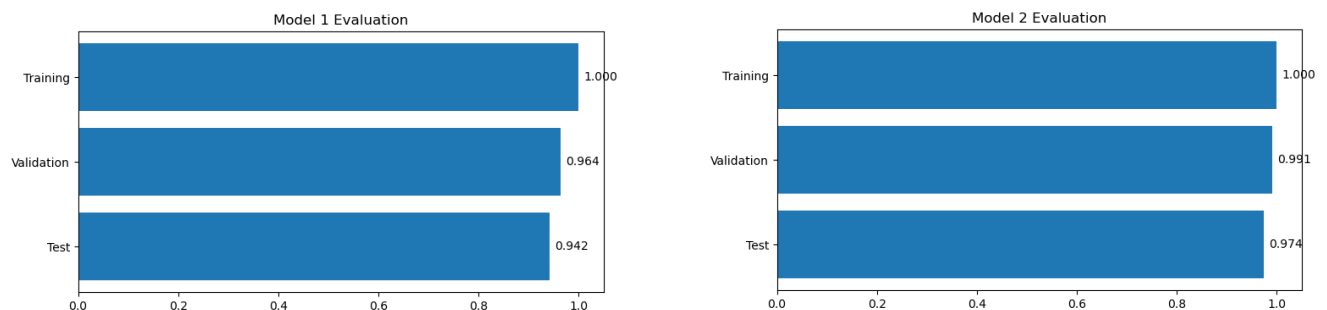


Figure 30 : Evaluation des modèles par précision de chaque ensemble

Ces résultats semblent très bons, avec une précision (accuracy) de plus de 99% sur les données d'entraînement (training Set) et de plus de 95% sur les données de validation (validation Set). Cependant, il est important de noter que ces résultats ne reflètent pas nécessairement les performances du modèle sur des données nouvelles et non vues (unseen data), qui est le véritable test de la capacité du modèle à généraliser.

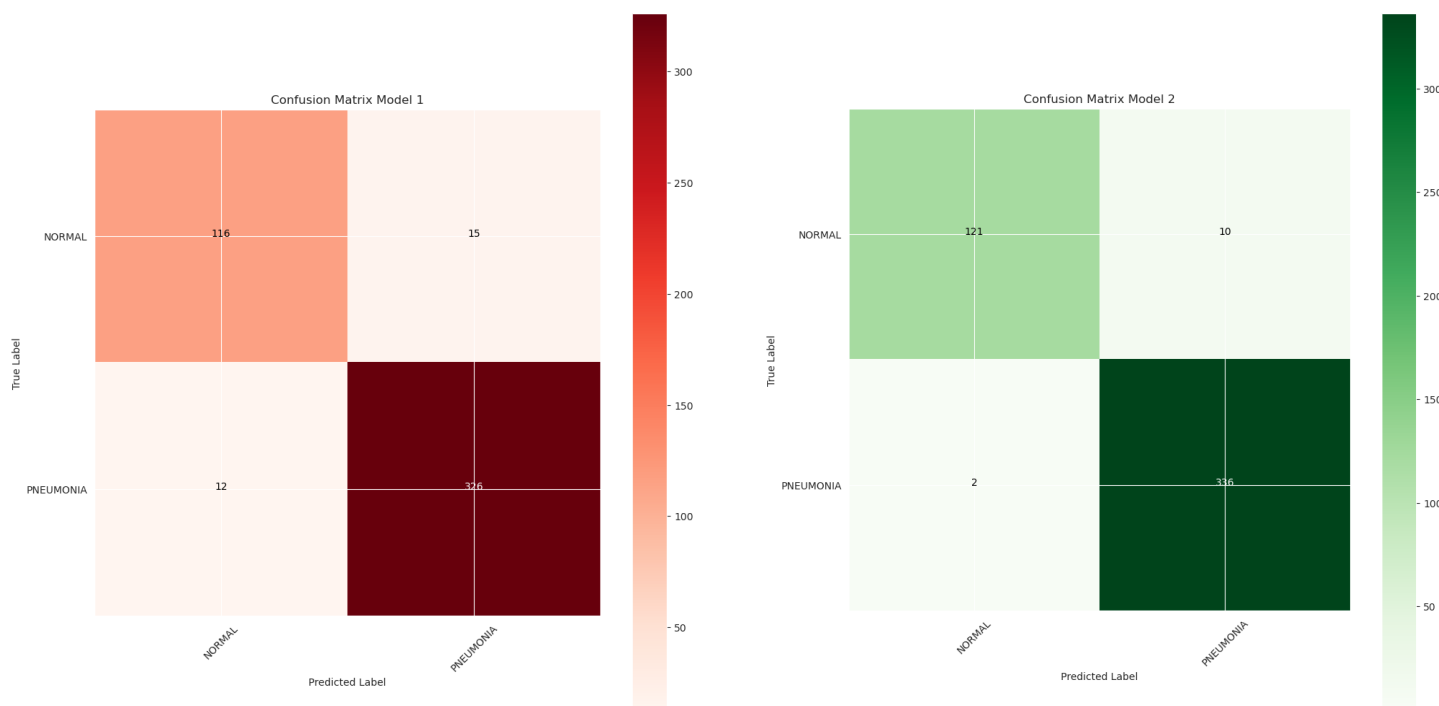


Figure 31 : Matrice de Confusion

	precision	recall	f1-score	support
NORMAL	0.91	0.89	0.90	131
PNEUMONIA	0.96	0.96	0.96	338
accuracy			0.94	469
macro avg	0.93	0.92	0.93	469
weighted avg	0.94	0.94	0.94	469

Figure 33 : Rapport de classification du modèle 1

	precision	recall	f1-score	support
NORMAL	0.98	0.92	0.95	131
PNEUMONIA	0.97	0.99	0.98	338
accuracy			0.97	469
macro avg	0.98	0.96	0.97	469
weighted avg	0.97	0.97	0.97	469

Figure 32 : Rapport de classification du modèle 2

Dans les figures ci-dessus :

Pour le modèle 1, nous pouvons constater que la précision pour la classe NORMAL est de 0,91, ce qui signifie que 91% des images classées comme NORMAL le sont effectivement. La précision pour la classe PNEUMONIA est de 0,96, ce qui signifie que 96% des images classées comme PNEUMONIA le sont effectivement. Le rappel pour la classe NORMAL est de 0,89, ce qui signifie que 89% des images qui sont réellement de la classe NORMAL ont été correctement identifiées par le modèle. Le rappel pour la classe PNEUMONIA est de 0,96, ce qui signifie que 96% des images qui sont réellement de la classe PNEUMONIA ont été correctement identifiées par le modèle. Le F1-score pour le modèle 1 est de 0,94 et l'exactitude globale (accuracy) est de 0,94.

Pour le modèle 2, les résultats sont globalement supérieurs à ceux du modèle 1. Nous pouvons constater que la précision pour la classe NORMAL est de 0,98, ce qui signifie que 98% des images classées comme NORMAL le sont effectivement. La précision pour la classe PNEUMONIA est de 0,97, ce qui signifie que 97% des images classées comme PNEUMONIA le sont effectivement. Le rappel pour la classe NORMAL est de 0,92, ce qui signifie que 92% des images qui sont réellement de la classe NORMAL ont été correctement identifiées par le modèle. Le rappel pour la classe PNEUMONIA est de 0,99, ce qui signifie que 99% des images qui sont réellement de la classe PNEUMONIA ont été correctement identifiées par le modèle. Le F1-score pour le modèle 2 est de 0,97 et l'exactitude globale (accuracy) est de 0,97.

En conclusion, le modèle 2 a des performances nettement supérieures au modèle 1. Il a une meilleure précision et un meilleur rappel pour les deux classes, ainsi qu'un F1-score et une exactitude globale (accuracy) plus élevés. Ces résultats suggèrent que le modèle 2 est plus efficace pour classer les images en fonction de la présence ou non de la pneumonie.

```
modele2.save(f"/kaggle/working/new_data/usingefficientnet.h5")
```

```
modele1.save(f"/kaggle/working/new_data/ourowndevcnn.h5")
```

Figure 34 : Sauvegarde des modèles

Les modèles ont été sauvegardés pour une utilisation future dans notre application web, qui sera équipée d'une fonctionnalité de détection de la pneumonie à partir d'images thoraciques. Nous aborderons davantage ce déploiement dans la partie suivante.

5. Déploiement du modèle

La dernière étape de notre projet était le déploiement de la solution développée pour une utilisation pratique. Pour notre solution de détection de pneumonie basée sur l'IA, nous avons choisi de déployer notre modèle en créant une application web à l'aide de « Flask API » et « PythonAnywhere » pour l'hébergement.

5.1. Création de l'API Flask :

Dans un premier temps, nous avons créé une API Flask pour intégrer notre modèle de détection de pneumonie. L'API a été conçue pour recevoir les images de radiographies

thoraciques en entrée et renvoyer les résultats de la détection de pneumonie. Tout d'abord, nous avons importé le modèle dans l'API. Ensuite, en cliquant sur "Upload image", l'image a été transformée en une matrice conforme à l'entrée du modèle, avec une taille cible de (224, 224). Enfin, en cliquant sur "Predict", le résultat a été affiché.

5.2. Interface Utilisateur :

Pour faciliter l'utilisation de notre solution, nous avons créé une interface utilisateur permettant aux utilisateurs d'envoyer des images de radiographies thoraciques et de visualiser les résultats de la détection de pneumonie en choisissant l'un des modèles que nous avons créés. Nous avons utilisé des styles CSS pour améliorer l'apparence de l'interface utilisateur.

5.3. Hébergement :

Nous avons ensuite hébergé l'API sur PythonAnywhere, le service cloud permettant de rendre notre API accessible au public. [42]

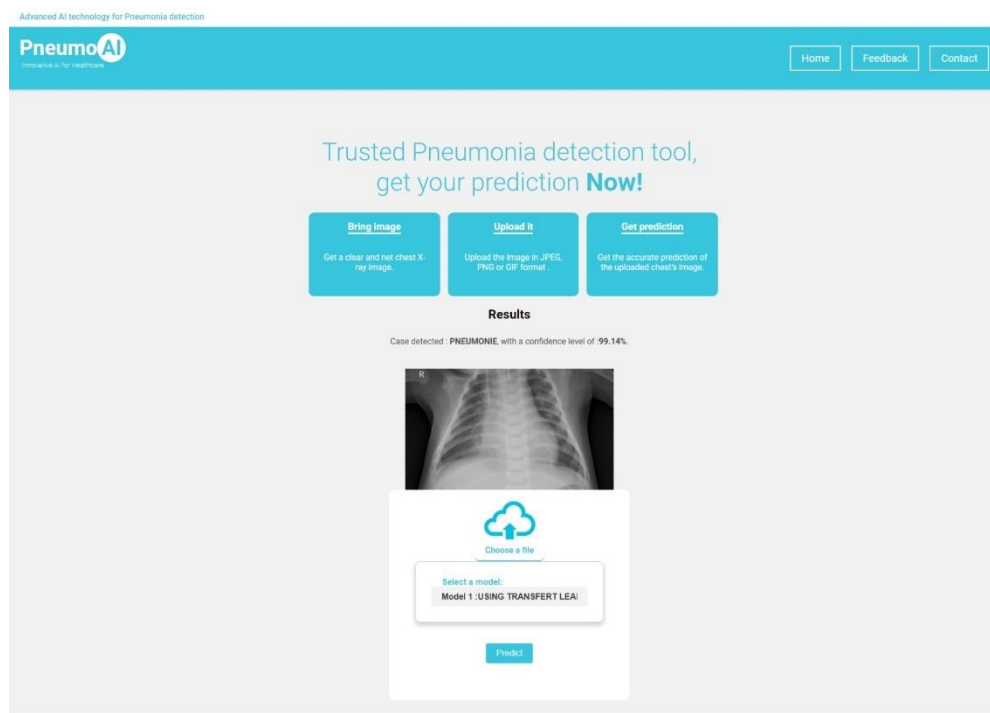


Figure 35 : Pneumonia Online Detector (aziz0220.pythonanywhere.com)

Enfin, nous avons effectué des tests finaux pour nous assurer que l'application fonctionne correctement et que les résultats de la détection de pneumonie sont précis. Nous avons également effectué des tests de charge pour nous assurer que notre solution peut gérer plusieurs requêtes simultanément sans diminuer les performances.

Conclusion

Finally, we have successfully deployed our AI solution for pneumonia detection.

The application is now available online and accessible to any user with an Internet connection.

Conclusion générale et perspectives

En conclusion, ce rapport a présenté une étude approfondie sur les différentes étapes et checkpoints qu'on a suivie tout au long de ce projet d'implémentation d'une solution Deep Learning pour la détection d'une pneumonie, nous avons examiné en profondeur le contexte du problème, et mis en évidence les différentes méthodes et approches possibles pour le résoudre. De plus, ce projet a fait preuve que les techniques de Deep Learning peuvent avoir une grande importance dans le domaine médical en aidant pour le diagnostic précoce non pas seulement pour la Pneumonie mais aussi pour beaucoup d'autre maladies graves. La solution que nous avons présentée pourrait potentiellement, après des améliorations, être appliqué dans les hôpitaux et présenter un outil efficace et fiable à utiliser par les médecins pour diagnostiquer plus rapidement les patients. Il est important de souligner que les résultats que nous avons atteint ne sont pas optimales et qu'il reste de nombreuses perspectives et possibilités d'amélioration et de recherche pour approfondir notre compréhension de ce sujet pour pouvoir donner un résultat plus efficace. Il est possible d'augmenter le nombre de données d'entraînement, notre modèle aura une meilleure compréhension des différents cas de pneumonie et sera en mesure de généraliser plus efficacement. On peut également utiliser des techniques d'optimisation plus avancées pour améliorer les performances de notre modèle. Ces techniques permettent d'optimiser les hyperparamètres du modèle de manière plus efficace et peuvent potentiellement améliorer considérablement les performances. Une autre perspective est d'ajouter de nouvelles fonctionnalités à notre modèle, telles que l'âge, le sexe et les antécédents médicaux du patient. En utilisant ces informations supplémentaires, notre modèle pourrait avoir une compréhension plus complète de la situation du patient et pourrait donc prendre des décisions plus précises en matière de diagnostic.

Bibliographie

- [1] O. M. d. I. Santé, «Principaux repères de l'OMS sur la pneumonie,» 11 Novembre 2022. [En ligne]. Available: <https://www.who.int/fr/news-room/fact-sheets/detail/pneumonia>. [Accès le 03 05 2023].
- [2] D. H. H. e. T. N. Wiesel, Cerveau et perception visuelle : histoire d'une collaboration de 25 ans, New York: Oxford University Press US, 2005, p. 106.
- [3] B. Goldstein, Sensation and Perception, London, Wadsworth , 2001.
- [4] I. J. B. R. Z. K. Y. B. M. H. e. a. Rajpurkar P, «A Review on Detection of Pneumonia in Chest X-ray Images Using Neural Networks,» 20 November 2018 . [En ligne]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6245676/>. [Accès le 29 04 2023].
- [5] R. M. F. S. Mahmud T, «CovXNet: A multidilation convolutional neural network for automatic COVID19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization,» 10 2020. [En ligne]. Available: <https://doi.org/10.1016/j.combiomed.2020.103869>. [Accès le 29 Avril 2023].
- [6] «ImageNet: VGGNet, ResNet, Inception, and Xception with Keras,» [En ligne]. Available: <https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>.
- [7] J. e. al., 2018. [En ligne]. [Accès le 04 05 2023].
- [8] P. K. K. O. Alhudhaif A, «Determination of COVID-19 pneumonia based on generalized convolutional neural network model from chest X-ray images,» 10 2021. [En ligne]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417421005820?via%3Dihub>. [Accès le 05 04 2023].
- [9] K. M. M. T. Y. Y. K. R. I. B. Sirazitdinov I, «Deep neural network ensemble for pneumonia localization from a large-scale chest x-ray database,» 04 08 2019. [En ligne]. Available: <https://doi.org/10.1016/j.compeleceng.2019.08.004>. [Accès le 05 04 2023].

- [10] T. P. K. S. G. D. K. A. R. J. Jaiswal AK, «Identifying pneumonia in chest X-rays: A deep learning approach. Measurement,» 05 2019. [En ligne]. Available: <https://doi.org/10.1016/j.measurement.2019.05.076>. [Accès le 04 05 2023].
- [11] N. P. K. G. K. V. H. D. Jain R, «Pneumonia detection in chest X-ray images using convolutional neural networks and transfer learning,» 10 2020. [En ligne]. Available: <https://doi.org/10.1016/j.measurement.2020.108046>. [Accès le 04 05 2023].
- [12] M. R. H. M. Karthik R, «Learning distinctive filters for COVID-19 detection from chest X-ray using shuffled residual CNN,» 2021. [En ligne]. Available: <https://doi.org/10.1016/j.asoc.2020.106744> 106744. [Accès le 04 05 2023].
- [13] B. Y. W. Y. L. H. L. H. X. Y. L. X. L. C. Q. D. Wang J, «Prior-Attention Residual Learning for More Discriminative COVID-19 Screening in CT Images,» IEEE Trans Med Imag, 2020. [En ligne]. Available: <https://doi.org/10.1109/TMI.2020.2994908>. [Accès le 04 05 2023].
- [14] K. M. G. V. F. S. Z. Z. Y. Wang SH, «Deep rank-based average pooling network for Covid-19 recognition,» 02 2022. [En ligne]. Available: <https://doi.org/10.32604/cmc.2022.020140>.
- [15] R. K. Sarada N, «A neural network architecture using separable neural networks for the identification of “pneumonia” in digital chest radiographs,» 2021. [En ligne]. Available: <https://doi.org/10.4018/IJeC.2021010106>. [Accès le 04 05 2023].
- [16] X. X. Z. T. L. Z. Z. M. C. X. Quan H, «DenseCapsNet: Detection of COVID-19 from X-ray images using a capsule neural network,» 15 04 2021. [En ligne]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8049190/>. [Accès le 04 05 2023].
- [17] R. K. Sarada N, «A neural network architecture using separable neural networks for the identification of “pneumonia” in digital chest radiographs,» 2021*. [En ligne]. Available: <https://doi.org/10.4018/IJeC.2021010106>.
- [18] P. G. R. V. e. a. Bhandary A, «Deep-learning framework to detect lung abnormality—a study with chest X-Ray and lung CT scan images,» 2019. [En ligne]. Available: <https://doi.org/10.1016/j.patrec.2019.11.013>.
- [19] A. O. M. M. Ouchicha C, «CVDNet: A novel deep learning architecture for detection of coronavirus (Covid-19) from chest x-ray images,» 2020. [En ligne]. Available: <https://doi.org/10.1016/j.chaos.2020.110245> .

- [20] K. T. L. C. Yi P, «Generalizability of deep learning tuberculosis classifier to COVID-19 chest radiographs: new tricks for an old algorithm? J Thoraic Imaging,» 2020. [En ligne]. Available: <https://doi.org/10.1097/RTI.0000000000000532> .
- [21] Z. Y. R. V. P. R. S. M. R. N. Dey N, «Customized VGG19 Architecture for Pneumonia Detection in Chest X-Rays,» 12 2020. [En ligne]. Available: <https://doi.org/10.1016/j.patrec.2020.12.010>.
- [22] G. P. S. M. M.-M. R. B. P. S. V. Panwar H, « A deep learning and grad-CAM based color visualization approach for fast detection of COVID-19 cases using chest X-ray and CT-Scan images,» 2020. [En ligne]. Available: <https://doi.org/10.1016/j.chaos.2020.110190>.
- [23] H. M. L. L. e. a. Vyas J, «Integrating blockchain technology into healthcare,» ACM, 2020.
- [24] S. A. Q. H. e. a. Huang L, «Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records,» 2019. [En ligne]. Available: <https://doi.org/10.1016/j.jbi . 2019.103291>.
- [25] C. S. T. G. A. S. Rajaraman S, «Visualizing and explaining deep learning predictions for pneumonia detection in pediatric chest radiographs,» 2019. [En ligne].
- [26] L. Z. Gaobo Liang, «A transfer learning method with deep residual network for pediatric pneumonia diagnosis,» 2019. [En ligne]. Available: https://www.researchgate.net/publication/334039488_A_transfer_learning_method_with_deep_residual_network_for_pediatric_pneumonia_diagnosis.
- [27] W. H. R. J. a. M. O'Quinn, «Pneumonia Radiograph Diagnosis Utilizing Deep Learning Network,» Proceedings of 2019 IEEE 2nd International, 2019.
- [28] O. S. M. M. U. J. a. J. D. U. Stephen, « An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare,» Journal of Healthcare Engineering 2019, 2019.
- [29] S. R. M. S. P. R. A. K. a. M. M. Islam, « Automatic Detection of Pneumonia on Compressed Sensing Images using Deep Learning,» 2019 IEEE Canadian Conference of Electrical and Computer Engineering, 2019.
- [30] A. K. T. P. K. S. G. D. K. A. a. R. J. J. Jaiswal, «Identifying pneumonia in chest X-rays: A deep learning approach, Measurement,» 2019. [En ligne]. Available: <https://doi.org/10.1016/j.measurement.2019.05.076>.

- [31] S. S. J. W. Z. Z. G. T. S. V. B. a. C. M. Jaipurkar, «Automated Classification Using End-to-End Deep Learning,» Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society EMBS 2018-July, 2018.
- [32] V. VP, «Study and development of a computer-aided diagnosis system for classification of chest X-ray images using convolutional neural networks pre-trained for imagenet and data augmentation,» 2008. [En ligne]. Available: <https://arxiv.org/abs/1806.00839>.
- [33] C. W. W. X. Z. Y. F. D. Li Q, « Medical image classification with convolutional neural network,» 12 2014. [En ligne]. Available: <https://ieeexplore.ieee.org/document/7064414>. [Accès le 04 05 2023].
- [34] S. M. Parveen N, «Detection of pneumonia in chest X-ray images,» J X-ray Sci Technol, 2011. [En ligne].
- [35] C. A. G. F. Caicedo JC, «Histopathology image classification using bag of features and kernel functions. In: Conference on artificial intelligence in medicine in Europe,» 2009. [En ligne].
- [36] K. D. L. T. W. B. N. H. V. E. Paredes R, «Classification of medical images using local representations,» 02 2002. [En ligne]. Available: https://www.researchgate.net/publication/2414111_Classification_of_Medical_Images_Using_Local_Representations.
- [37] P. MOONEY, «Chest X-Ray Images (Pneumonia),» Kaggle, 24 Mars 2018. [En ligne]. Available: <https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia>. [Accès le 15 02 2023].
- [38] «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» 2015. [En ligne]. Available: <https://arxiv.org/abs/1502.03167>.
- [39] J. Brownlee, «How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification,» machinelearningmastery, 03 01 2020. [En ligne]. Available: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/?fbclid=IwAR0DaqesyVl1ifZb-COTKVzR7jKH842M-UZJYKMIwxj4_u5xlHxocTSwE4I#:~:text=Recall%20is%20a%20metric%20that,indication%20of%20missed%20positive%20predictions. [Accès le 04 05 2023].

- [40] «Le fonctionnement d'un projet de logiciel libre : Scikit-learn,» 15 Avril 2022. [En ligne]. Available: <https://www.lemonde.fr/blog/binaire/2022/04/15/le-fonctionnement-dun-projet-de-logiciel-libre-scikit-learn/>. [Accès le 04 Mai 2023].
- [41] «Les outils d'analyse de données open source,» [En ligne]. Available: <https://analytics.fr/analytics/outils-analyse-open-source/>.
- [42] Q. V. L. Mingxing Tan, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» 2020.
- [43] N. M. N. Ben Amor Aziz, «Notre site web crée,» [En ligne]. Available: <http://aziz0220.pythonanywhere.com/>.
- [44] [En ligne]. Available: http://mrl.cs.vsb.cz/data/ano2/ano2_ang/AlexNet_ZFNet_VGGNet_GoogLeNet_ResNet_DenseNet_RCNN.pdf.

Résumé

La détection de la pneumonie à l'aide de l'apprentissage profond est un domaine de recherche prometteur et stimulant dans le domaine médical. Cette étude présente une solution pour la détection de la pneumonie en utilisant des algorithmes d'apprentissage profond, qui combinent différentes techniques pour améliorer la précision du diagnostic. Le modèle proposé a été testé sur un grand ensemble de données, et sa performance a été améliorée en termes de précision et de sensibilité.

Des recherches futures pourraient étendre l'approche proposée en modifiant des paramètres supplémentaires, tels que l'âge et le sexe des patients. Cela pourrait contribuer à améliorer la performance globale du modèle.

Mots clés : pneumonie, apprentissage profond, modèle, jeu de données, réseau de neurones convolutifs.

Abstract

Detection of pneumonia using deep learning is a promising and exciting area of research in the medical field. This study presents a solution for pneumonia detection using deep learning algorithms, which combine different techniques to improve diagnostic accuracy. The proposed model has been tested on a large data set, and its performance has been improved in terms of accuracy and sensitivity.

Future research could extend the proposed approach by modifying additional parameters, such as patient age and gender. This could help improve the overall performance of the model.

Keywords: pneumonia, deep learning, model, dataset, Convolutional Neural Network.

الملخص

يعد اكتشاف الالتهاب الرئوي باستخدام التعلم العميق مجالًا بحثيًا واعدًا ومثيرًا في المجال الطبي. تقدم هذه الدراسة حلاً للكشف عن الالتهاب الرئوي باستخدام خوارزميات التعلم العميق، والتي تجمع بين تقنيات مختلفة لتحسين دقة التشخيص. تم اختبار النموذج المقترح على مجموعة كبيرة من البيانات، وتم تحسين أدائه من حيث الدقة والحساسية. يمكن أن يوسع البحث المستقبلي النهج المقترح من خلال تعديل المعلومات الإضافية، مثل عمر المريض والجنس. هذا يمكن أن يساعد في تحسين الأداء العام للنموذج.

الكلمات المفتاحية: الالتهاب الرئوي، التعلم العميق، النموذج، مجموعة البيانات، الشبكة العصبية التلافيفية.