

RAPPORT COMPUTER VISION

*Application des algorithmes
Machine Learning sur l'imagerie
des infections pulmonaire due au
Covid-19*

January 26, 2022

LACHHEB HAJAR - REGUIG JAMILA

Encadré par: MIKRAM Mounia

Résumé

Ce présent rapport évalue l'utilisation potentielle de l'imagerie médicale et du traitement numérique des images pour la classification des imageries de poumons.

Les travaux effectués et les modèles développés sur chacune des datasets à part ont déjà atteint de bonne qualité de prédiction. Notre objectif étant de construire un modèle qui effectue une classification sur les images pulmonaires des deux Datasets combinés.

Le travail bibliographique et l'analyse des publications formelles et informelles concernant le Dataset nous révèlent différents modèles qui ont été réalisés et dont certains d'eux atteignent une précision de 97% notamment celle du modèle CNN.

Le travail qu'on a effectué sur les datasets consiste principalement sur l'importation des données, unifier la représentation des images et l'ajustement de tailles et des bandes de spectres utilisés (RGB) pour avoir des données homogènes.

Vu que notre dataset est une base de données avec trois classes, il nous s'est avéré plus correct d'adapter un modèle qui permet déjà de classifier respectueusement les 3 classes de la dataset. Pour aboutir finalement à un modèle capable de classifier les 3 classes des datasets indifféremment.

Les résultats obtenus du modèle CNN développé sont respectivement : une précision de 97%. La problématique donc est d'essayer d'augmenter le taux d'accuracy en utilisant des modèles qui pourraient prédire si le patient est atteint de Covid19 ou d'une maladie pulmonaire tout en implémentant un réseau de neurones convolutifs qui opère sur un Dataset d'images élargis.

On a réussi alors à atteindre une précision de 98 % sur les données de test. Et un pourcentage qui dépasse les 90% au niveau des autres modèles pretraites. Les recommandations proposées visent à améliorer ce score en effectuant des traitements supplémentaires dans le but d'éviter la dégradation de qualité des images mais aussi de proposer une solution qui va être implémentée par la suite afin d'être un support pour le staff médical.

Table des matières

Résumé	1
Introduction - Contexte Métier	3
I. Étude théorique: Description des modèles	4
A. CNN (From Scratch)	4
B. MobileNetV2	7
C. VGG16	9
D. VGG19	10
II. Etude Pratique: Présentation de la solution	12
A. Description des Datasets	12
B. Méthodes utilisées de classification	16
C. Méthodes utilisées de l'attaque	20
D. Méthodes utilisées de défense	24
Conclusions	27
Recommandations	28

Introduction:

COVID-19 est une crise sanitaire mondiale, avec plus de 332 millions de personnes infectées et plus de 5 millions de décès signalés (jusqu'en janvier 2020) dans le monde. L'impact qui en résulte sur les systèmes de santé est que de nombreux pays ont augmenté leurs ressources pour atténuer la propagation de la pandémie. De plus, un degré élevé de variance dans les symptômes du COVID-19 a été signalé, avec des symptômes allant d'une grippe légère au syndrome de détresse respiratoire aiguë (SDRA) ou à une pneumonie fulminante. Il existe un besoin urgent de médicaments et de vaccins efficaces pour le traitement et la prévention de la COVID-19. En raison du manque de thérapeutiques validées, la plupart des mesures de confinement visant à freiner la propagation de la maladie reposent sur la distanciation sociale, les mesures de quarantaine et les politiques de confinement. La transmission du COVID-19 a été ralentie grâce à ces mesures, mais pas éliminée. De plus, avec la facilité des restrictions, la peur de la deuxième vague d'infection est répandue. Pour prévenir de nouvelles épidémies de COVID-19, il est nécessaire de mettre en place des mesures de confinement avancées telles que la recherche des contacts et l'identification des points chauds.

L'apprentissage automatique (ML) est un sous-domaine de l'IA qui se concentre sur les algorithmes qui permettent aux ordinateurs de définir un modèle pour des relations ou des modèles complexes à partir de données empiriques sans être explicitement programmés. L'apprentissage profond (DL), une sous-catégorie de ML, atteint une grande puissance et flexibilité par rapport aux modèles ML conventionnels en s'inspirant des réseaux de neurones biologiques pour résoudre une grande variété de tâches complexes, y compris la classification de l'imagerie médicale et le traitement du langage naturel (NLP).

Contexte Métier

Pour notre projet, il est important de préciser le contexte de travail, vu qu'il traite une problématique importante. En effet, il faut d'abord se positionner dans le contexte pandémique.

La situation sanitaire dans le monde et au Maroc, relative à la pandémie du Corona (COVID-19), est à l'origine d'une large mobilisation des pouvoirs publics, afin de limiter la propagation du virus et pour la protection nécessaire et indispensable de la population. Il est certain que cette pandémie a eu des effets positifs sur les différents aspects de l'environnement puisque plus de 180 pays ont opté pour le confinement des populations et l'arrêt presque total des activités humaines et industrielles. Sans oublier la mobilisation du staff médicale afin d'assurer une bonne gestion interne au sein des hôpitaux.

Ceci dit, l'un des plus grands problèmes auxquels les hôpitaux ont été confrontés est la gestion du nombre croissant de membres du personnel qui sont tombés malades ou ont été exposés au virus et qui doivent s'isoler pendant 14 jours. Sans oublier, qu'au

sein de nos hôpitaux, il n'y avait que peu de personnels médicaux qualifiés pour lutter contre la pandémie.

Se focaliser sur la pandémie a bien sûr eu un impact négatif sur d'autres maladies ainsi que le plan à suivre afin de réaliser un bon diagnostic. En prenant en considération la fatigue, le nombre d'heures de travail et la situation pandémique, d'autres maladies telles que les maladies du poumons ont été de moins en moins prises en compte surtout en termes de diagnostic. Sans oublier qu'une imagerie de poumons est aussi effectuée pour mesurer l'effet négatif que le Covid a eu sur le corps de la personne atteinte de la maladie.

Toujours dans le contexte de Covid19, un autre point à prendre en compte est le fait que les images x-ray sont toujours moins chères que les tests PCR.

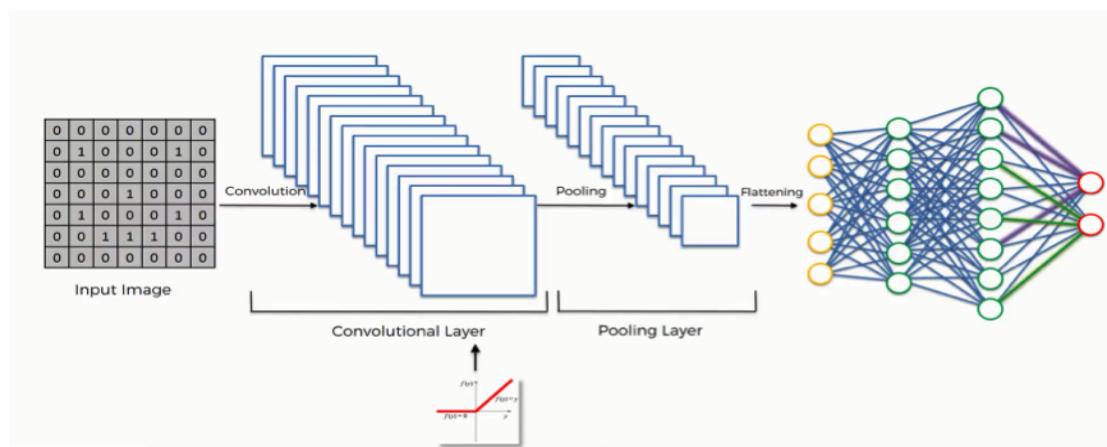
En effet, le temps de résultat est aussi moindre c'est à dire qu'il prend moins de temps. Le résultat des analyses est aussi instantané.

I. Étude théorique: Description des modèles.

❖ CNN (From Scratch) :

1. CNN: Une introduction générale.

En apprentissage automatique, un réseau de neurones convolutifs ou réseau de neurones à convolution est un type de réseau de neurones artificiels acycliques, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux



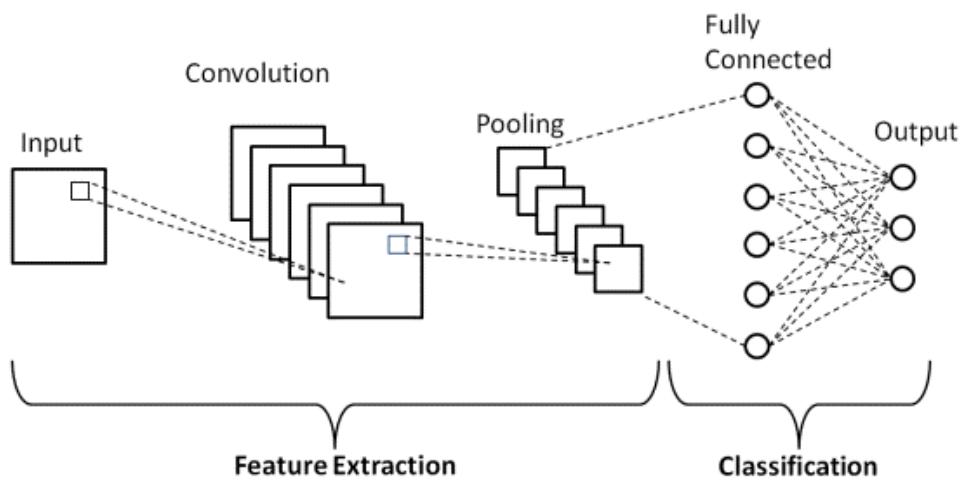
En effet, les neurones de cette région du cerveau sont arrangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges

applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

2. L'architecture du CNN:

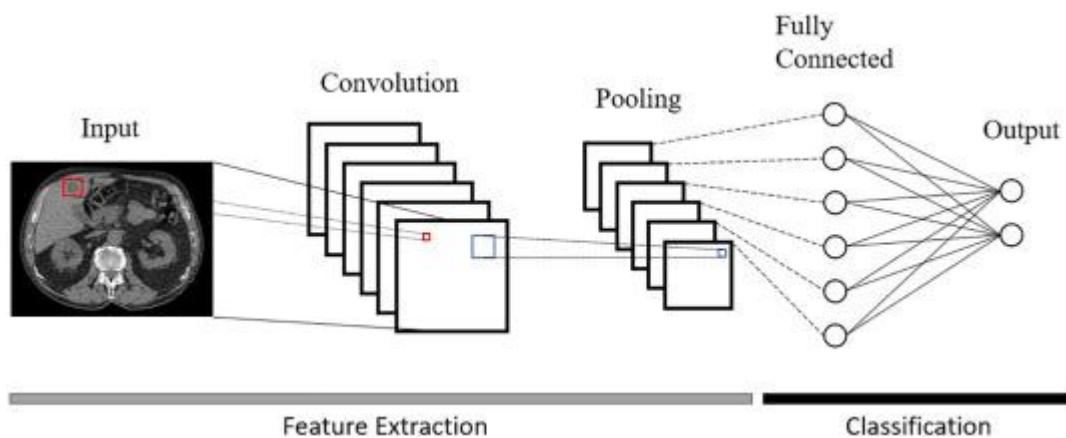
Il y a deux parties principales dans une architecture CNN

- Un outil de convolution qui sépare et identifie les différentes caractéristiques de l'image pour l'analyse dans un processus appelé Extraction de caractéristiques
- Une couche entièrement connectée qui utilise la sortie du processus de convolution et prédit la classe de l'image en fonction des caractéristiques extraites lors des étapes précédentes.



→ Couches de convolution:

Il existe trois types de couches qui composent le CNN, à savoir les couches convolutives, les couches de mise en commun et les couches entièrement connectées (FC). Lorsque ces couches sont empilées, une architecture CNN sera formée. En plus de ces trois couches, il existe deux paramètres plus importants qui sont la couche de décrochage et la fonction d'activation qui sont définis ci-dessous.



1) Couche convulsive:

Cette couche est la première couche utilisée pour extraire les différentes caractéristiques des images d'entrée. Dans cette couche, l'opération mathématique de convolution est effectuée entre l'image d'entrée et un filtre d'une taille particulière MxM. En faisant glisser le filtre sur l'image d'entrée, le produit scalaire est pris entre le filtre et les parties de l'image d'entrée par rapport à la taille du filtre (MxM).

La sortie est appelée carte des caractéristiques qui nous donne des informations sur l'image telles que les coins et les bords. Plus tard, cette carte de caractéristiques est transmise à d'autres couches pour apprendre plusieurs autres caractéristiques de l'image d'entrée.

2) Pooling Layer:

Dans la plupart des cas, une couche convulsive est suivie d'une couche de mise en commun. L'objectif principal de cette couche est de réduire la taille de la carte des caractéristiques convoluées afin de réduire les coûts de calcul. Ceci est effectué en diminuant les connexions entre les couches et opère indépendamment sur chaque carte de caractéristiques. Selon la méthode utilisée, il existe plusieurs types d'opérations de pooling.

Dans Max Pooling, le plus grand élément est extrait de la carte des caractéristiques. Average Pooling calcule la moyenne des éléments dans une section Image de taille prédéfinie. La somme totale des éléments de la section prédéfinie est calculée dans Sum Pooling. La couche de mise en commun sert généralement de pont entre la couche convulsive et la couche FC.

3) Fully Connected Layer:

La couche entièrement connectée (FC) comprend les poids et les biais ainsi que les neurones et est utilisée pour connecter les neurones entre deux couches différentes. Ces couches sont généralement placées avant la couche de sortie et forment les dernières couches d'une architecture CNN. Dans ce cas, l'image d'entrée des couches précédentes est aplatie et transmise à la couche FC. Le vecteur aplati subit ensuite quelques couches FC supplémentaires où les opérations des fonctions mathématiques ont généralement lieu. À ce stade, le processus de classification commence à avoir lieu.

4) Dropout:

Généralement, lorsque toutes les entités sont connectées à la couche FC, cela peut entraîner un surapprentissage dans l'ensemble de données d'apprentissage. Le surapprentissage se produit lorsqu'un modèle particulier fonctionne si bien sur les données d'entraînement, ce qui a un impact négatif sur les performances du modèle lorsqu'il est utilisé sur de nouvelles données.

Pour surmonter ce problème, une couche de suppression est utilisée dans laquelle quelques neurones sont supprimés du réseau neuronal pendant le processus d'apprentissage, ce qui réduit la taille du modèle. En passant un abandon de 0,3, 30% des nœuds sont abandonnés au hasard du réseau de neurones.

5) Activation Functions:

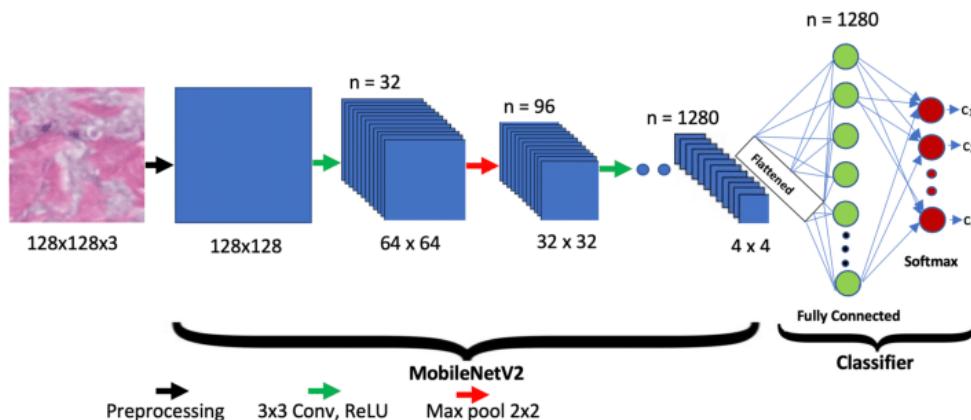
Enfin, l'un des paramètres les plus importants du modèle CNN est la fonction d'activation. Ils sont utilisés pour apprendre et approximer tout type de relation continue et complexe entre les variables du réseau. En termes simples, il décide quelles informations du modèle doivent être déclenchées dans le sens direct et lesquelles ne doivent pas être transmises à la fin du réseau.

Il ajoute de la non-linéarité au réseau. Il existe plusieurs fonctions d'activation couramment utilisées telles que les fonctions ReLU, Softmax, tanH et Sigmoïde. Chacune de ces fonctions a un usage spécifique. Pour un modèle CNN de classification binaire, les fonctions sigmoïde et softmax sont préférées et pour une classification multi-classe, généralement softmax est utilisée.

❖ MobileNetV2:

1. Introduction de la MobileNetV2:

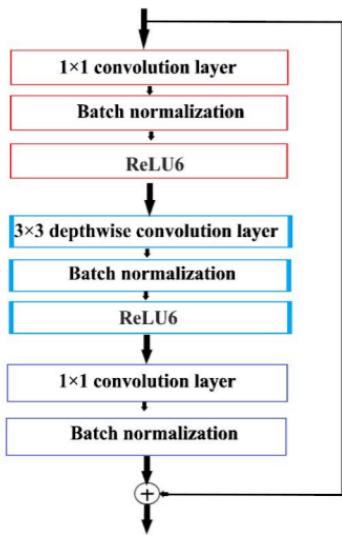
MobileNet est une architecture CNN efficace et portable qui est utilisée dans les applications du monde réel. Les MobileNets utilisent principalement des convolutions séparables en profondeur à la place des convolutions standard utilisées dans les architectures antérieures pour créer des modèles plus légers. sur leurs exigences.



Les principaux changements apportés à l'architecture ont été l'introduction de blocs résiduels inversés et de goulots d'étranglement linéaires et l'utilisation de la fonction d'activation ReLU6 à la place de ReLU.

2. Architecture de la MobileNetV2:

L'architecture du MobileNetV2 est donnée ci-dessous :



Input	Operator	<i>t</i>	<i>c</i>	<i>n</i>	<i>s</i>
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Le MobileNet v2 est présenté comme un raffinement du MobileNet v1 qui le rend encore plus efficace et puissant. Dans l'architecture MobileNet v2, le bloc de convolution séparable en profondeur a été repensé.

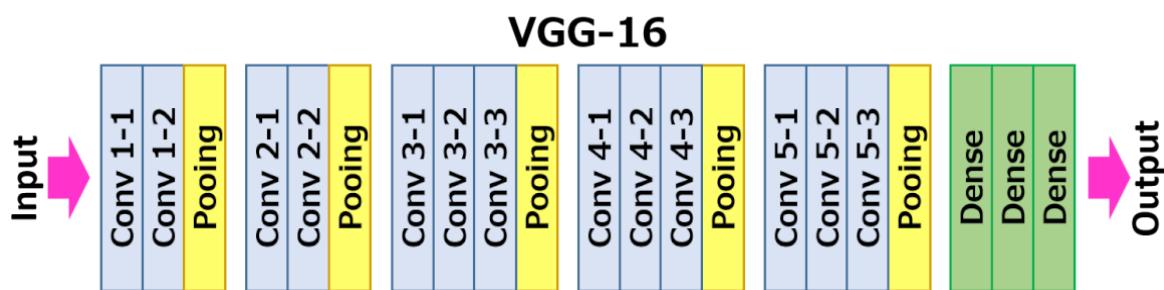
Il y a trois couches convolutives dans le nouveau bloc de convolution séparable en profondeur. La première couche est une couche de convolution 1×1 qui est utilisée pour étendre le nombre de canaux dans la carte des caractéristiques en entrée avant d'entrer dans la couche de convolution en profondeur. La couche intermédiaire est une couche de convolution en profondeur 3×3 qui filtre la carte des caractéristiques en entrée comme dans MobileNet V1. La couche finale est une couche de convolution 1×1 . Cependant, cette couche de convolution finale est utilisée pour projeter des données avec un nombre élevé de canaux dans un tenseur avec un nombre beaucoup plus faible de canaux, réduisant ainsi le nombre de canaux de la carte de caractéristiques d'entrée. Cette dernière couche est également appelée couche de goulot d'étranglement car elle réduit la quantité de données qui circulent sur le réseau.

De plus, la connexion résiduelle comme dans ResNet est adoptée dans l'architecture MobileNet v2 pour faciliter le flux de gradients à travers le réseau. Chaque couche de la nouvelle convolution séparable en profondeur est suivie d'une normalisation par lots et de ReLU6 comme fonction d'activation (sauf la dernière couche de goulot d'étranglement car l'utilisation d'une non-linéarité après que cette couche a détruit des informations utiles). L'architecture complète de MobileNet v2 est illustrée dans le tableau 2. Comme illustré dans le tableau 2, MobileNet v2 se compose de 17 des nouveaux blocs de convolution séparables en profondeur, suivis d'une couche de convolution régulière 1×1 . La première couche est une couche de convolution 3×3 régulière avec 32 canaux.

❖ **VGG16:**

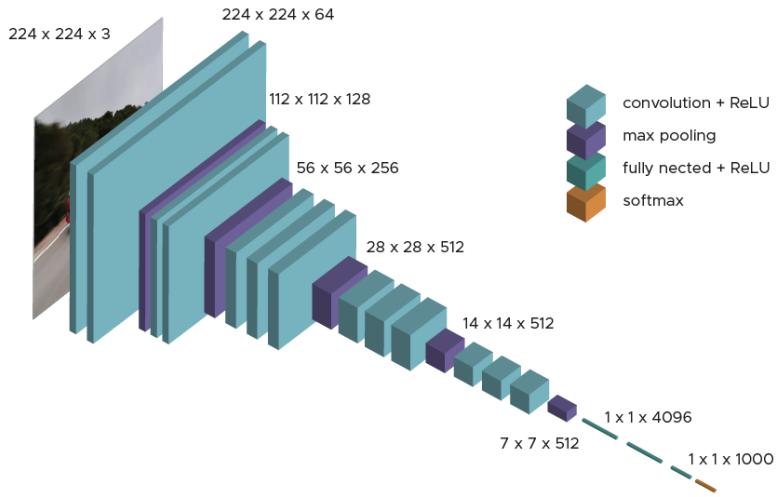
1. **Introduction à la VGG16:**

VGG16 est une architecture de réseau neuronal convolutif (CNN) simple et largement utilisée pour ImageNet, un grand projet de base de données visuelle utilisé dans la recherche de logiciels de reconnaissance d'objets visuels. L'architecture VGG16 a été développée et introduite par Karen Simonyan et Andrew Zisserman de l'Université d'Oxford, en 2014, à travers leur article « Very Deep Convolutional Networks for Large-Scale Image Recognition ». « VGG » est l'abréviation de Visual Geometry Group, qui est un groupe de chercheurs de l'Université d'Oxford qui a développé cette architecture, et « 16 » implique que cette architecture a 16 couches (expliquée plus tard).



Le modèle VGG16 a atteint une précision de test de 92,7% dans le top 5 dans ImageNet, qui est un ensemble de données de plus de 14 millions d'images appartenant à 1000 classes. C'était l'un des célèbres modèles soumis à ImageNet Large Scale Visual Recognition Challenge (ILSVRC) en 2014. Il a apporté des améliorations par rapport à l'architecture AlexNet en remplaçant les grands filtres de la taille du noyau (11 et 5 dans la première et la deuxième couche convulsive, respectivement), avec plusieurs filtres trois × trois de la taille du noyau les uns après les autres. VGG16 a été formé pendant des semaines à l'aide de GPU NVIDIA Titan Black.

2. **Architecture de la VGG16:**



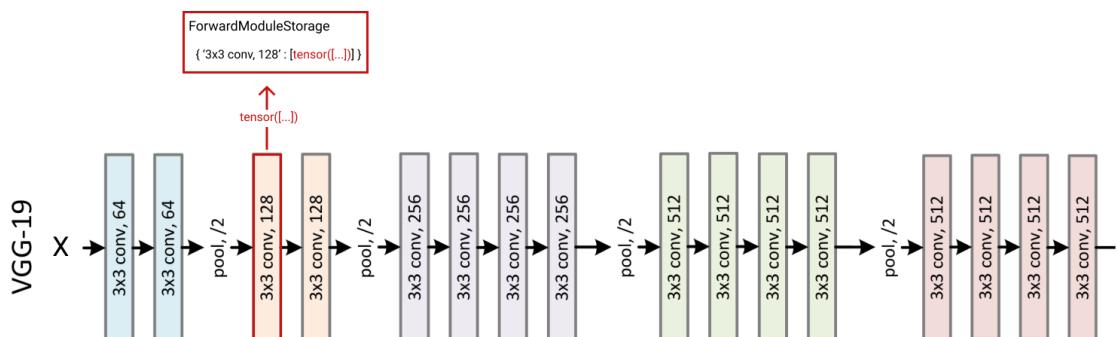
Durant l'apprentissage du modèle, l'input de la première couche de convolution est une image RGB de taille 224 x 224. Pour toutes les couches de convolution, le noyau de convolution est de taille 3×3: la plus petite dimension pour capturer les notions de haut, bas, gauche/droite et centre. C'était une spécificité du modèle au moment de sa publication. Jusqu'à VGG16 beaucoup de modèles s'orientaient vers des noyaux de convolution de plus grande dimension (de taille 11 ou bien de taille 5 par exemple). Rappelons que ces couches ont pour but de filtrer l'image en ne gardant que des informations discriminantes comme des formes géométriques atypiques.

Ces couches de convolution s'accompagnent de couche de Max-Pooling, chacune de taille 2×2, pour réduire la taille des filtres au cours de l'apprentissage.

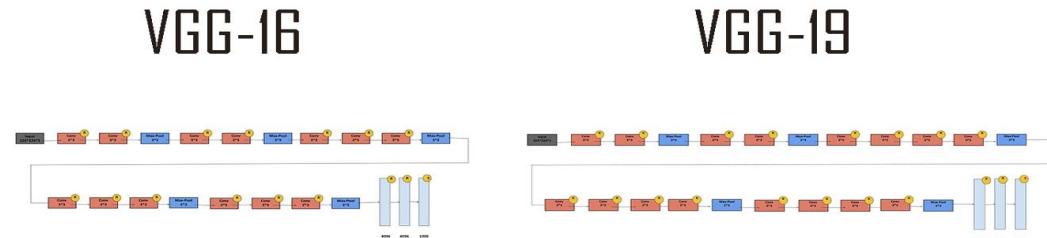
En sortie des couches de convolution et pooling, nous avons 3 couches de neurones Fully-Connected. Les deux premières sont composées de 4096 neurones et la dernière de 1000 neurones avec une fonction d'activation softmax pour déterminer la classe de l'image.

❖ VGG19:

1. Introduction à la VGG16:



Le concept du modèle VGG19 (également VGGNet-19) est le même que celui du VGG16 sauf qu'il prend en charge 19 couches. Les « 16 » et « 19 » représentent le nombre de couches de poids dans le modèle (couches convolutives). Cela signifie que VGG19 a trois couches convolutives de plus que VGG16.



2. Architecture de la VGG19:

- Une image RGB de taille fixe ($224 * 224$) a été fournie en entrée à ce réseau, ce qui signifie que la matrice était de forme ($224, 224, 3$).
- Le seul prétraitement qui a été effectué est qu'ils ont soustrait la valeur RVB moyenne de chaque pixel, calculée sur l'ensemble de l'ensemble d'apprentissage.
- Des noyaux utilisés de taille ($3 * 3$) avec une taille de foulée de 1 pixel, cela leur a permis de couvrir toute la notion d'image.
- Un remplissage spatial a été utilisé pour préserver la résolution spatiale de l'image.
- Un Max pooling a été effectué sur une fenêtre de $2 * 2$ pixels avec stride 2.
- Cela a été suivi par l'unité linéaire rectifiée (ReLU) pour introduire la non-linéarité afin de mieux classer le modèle et d'améliorer le temps de calcul, car les modèles précédents utilisaient des fonctions tanh ou sigmoïde, cela s'est avéré bien meilleur que ceux-ci.
- Mise en œuvre trois couches entièrement connectées dont les deux premières étaient de taille 4096 et après cela, une couche avec 1000 canaux pour la classification ILSVRC à 1000 voies et la couche finale est une fonction softmax.

II. Etude Pratique: Présentation de la solution.

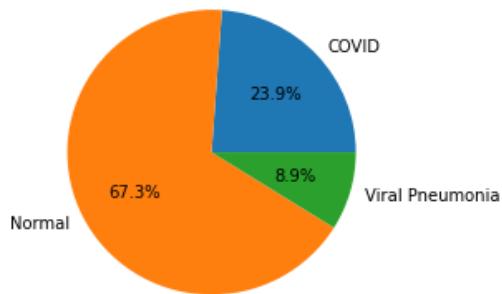
A. Description des Datasets:

- ❖ Nom du Dataset N°1: COVID-19 Radiography Database (COVID-19 Chest X-ray Database)

- ❖ Description du Dataset:

Une équipe de chercheurs de différentes universités du monde entier, en collaboration avec des médecins, a créé une base de données d'images de radiographie pulmonaire pour les cas positifs au COVID-19 ainsi que des images de pneumonie normale et virale. L'ensemble de données est toujours mis à jour dans différentes versions. À compter de la deuxième mise à jour, la base de données a été augmentée à 3616 cas positifs au COVID-19 ainsi que 10 192 images normales, 6012 images d'opacité pulmonaire (infection pulmonaire non COVID) et 1345 images de pneumonie virale.

```
fig, ax = plt.subplots()
ax.pie(
    class_len.values(),
    labels = class_len.keys(),
    autopct = "%1.1f%"
)
fig.show()
```



- ❖ Lien de téléchargement:

<https://www.kaggle.com/tawsifurrahman/covid19-radiography-database>

- ❖ Affichage des échantillons d'images :

```

fig, axs = plt.subplots(len(labels), 5, figsize = (15, 15))

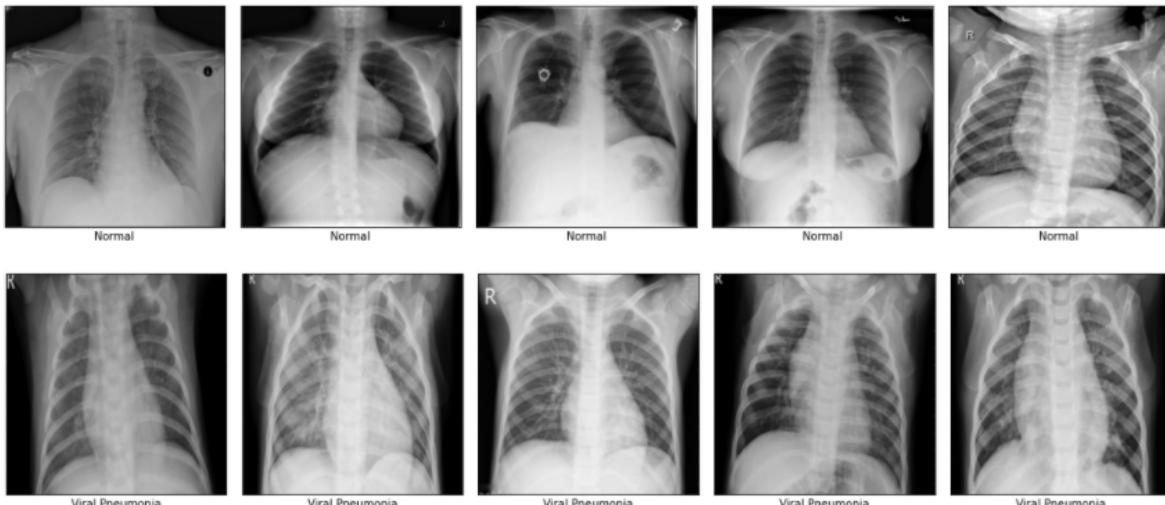
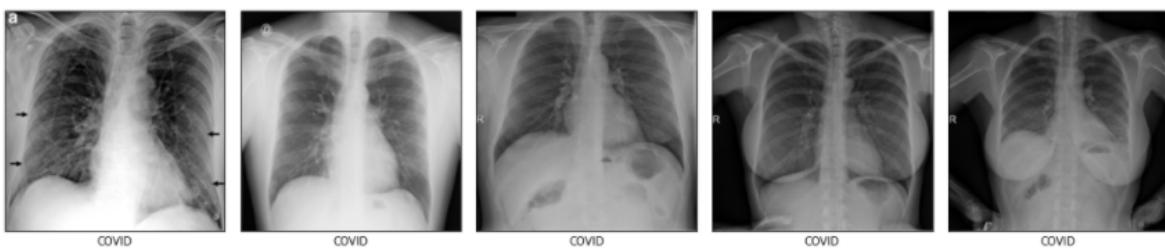
class_len = {}
for i, c in enumerate(labels):
    class_path = os.path.join(path, c)
    all_images = os.listdir(class_path)
    sample_images = random.sample(all_images, 5)
    class_len[c] = len(all_images)

    for j, image in enumerate(sample_images):
        img_path = os.path.join(class_path, image)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        axs[i, j].imshow(img)
        axs[i, j].set(xlabel = c, xticks = [], yticks = [])

fig.tight_layout()

```



❖ **Nom du Dataset N°2: Tuberculosis (TB) Chest X-ray Database (The largest TB Chest X-ray Database)**

❖ **Description du Dataset:**

Il s'agit d'une base de données d'images radiographiques thoraciques pour les cas positifs de tuberculose (TB) ainsi que d'images normales. Dans notre version actuelle, il y a 700 To d'images accessibles au public et 3500 images normales.

Pour des raisons de commodité du projet, nous utiliserons cette base de données uniquement à des fins d'*augmentation de données*.

❖ Lien de Téléchargement:

<https://www.kaggle.com/tawsifurrahman/tuberculosis-tb-chest-xray-dataset>

❖ Affichage d'un échantillon des images:

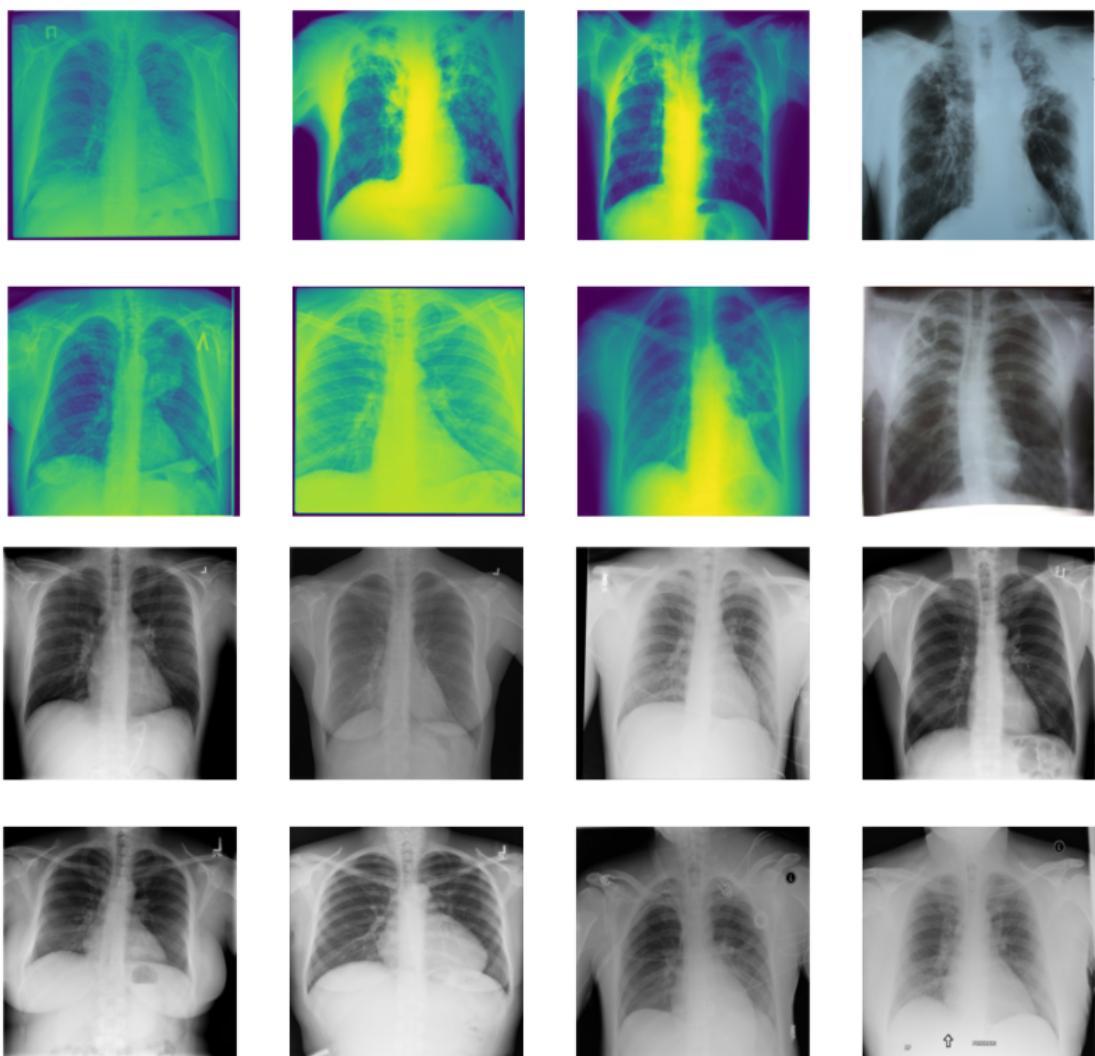
```
fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)
pic_index+=8

tuberculosis_img = [os.path.join(tuberculosis_data, image) for image in os.listdir(tuberculosis_data)]
normal_img = [os.path.join(normal_data, image) for image in os.listdir(normal_data)[pic_index:]]

for i, image_path in enumerate(tuberculosis_img+normal_img):
    sp = plt.subplot(nrows, ncols, i + 1)
    sp.axis('Off')

    img = mpimg.imread(image_path)
    plt.imshow(img)

plt.show()
```



❖ **La version finale de notre Dataset :**

Dans notre cas d'étude, nous avons décidé de fusionner les deux datasets. Nous avons ajouté la deuxième dataset à la première dataset, en réalisant ainsi une Data Augmentation.

• **Au niveau de la division d'entraînement- Train Data:**

- 12 576 Chest Images
- 3 Classes (Covid, Normal, Pneumonia)
- 4192 Covid Images, 4192 Normal Images et 4192 Pneumonia Images

```
COVID      4192
PNEUMONIA  4192
NORMAL     4192
Name: target, dtype: int64
```

• **Au niveau de la division du test - Test Data:**

- 2 516 Chest Images
- 3 Classes (Covid, Normal, Pneumonia)
- 857 Covid Images, 841 Normal Images et 818 Pneumonia Images

```
COVID      857
NORMAL    841
PNEUMONIA 818
Name: target, dtype: int64
```

❖ **Démarche Pratique a suivre :**

Il faut commencer par déterminer le besoin à combler:

1. Effectuer une classification des images en utilisant des modèles pré-entraînés par transfert learning en mode feature extraction et mode fine tuning et un modèle de votre design à entraîner from scratch.
2. Entraîner les modèles et veiller à ce que le taux d'accuracy est élevé et qu'on arrive en final à bien prédire la catégorie ou la "disease", maladie observée au niveau de l'imagerie.
3. Adversarial Machine Learning

1) La Classification:

La classification est le processus de prédiction de la classe de points de données donnés. Les classes sont parfois appelées cibles/étiquettes ou catégories. La modélisation prédictive de classification est la tâche d'approximation d'une fonction de mappage (f) des variables d'entrée (X) aux variables de sortie discrètes (y). La classification appartient à la catégorie de l'apprentissage supervisé où les cibles ont également fourni les données d'entrée. Il existe de nombreuses applications en classification dans de nombreux domaines tels que l'approbation de crédit, le diagnostic médical, le marketing ciblé, etc. Il existe actuellement de nombreux algorithmes de classification, mais il n'est pas possible de conclure lequel est supérieur aux autres. Cela dépend de l'application et de la nature de l'ensemble de données disponibles.

2) Exemple des CNNs:

Les réseaux de neurones convolutifs (CNN ou ConvNet) sont des réseaux de neurones complexes. Les CNNs sont utilisés pour la classification et la reconnaissance d'images en raison de leur grande précision. Le CNN suit un modèle hiérarchique qui fonctionne sur la construction d'un réseau, comme un entonnoir, et fournit enfin une couche entièrement connectée où tous les neurones sont connectés les uns aux autres et la sortie est traitée.

3) Adversarial Machine Learning:

Adversarial Machine Learning est une technique qui tente de tromper les modèles avec des données fausses. C'est une menace croissante dans la communauté de recherche sur l'IA et le machine learning. La raison la plus courante est de provoquer un dysfonctionnement dans un modèle ML. L'Adversarial attack peut impliquer la présentation d'un modèle avec des données inexactes ou fausses lors de son entraînement, ou l'introduction de données conçues de manière malveillante pour tromper un modèle déjà formé.

Une attaque peut influencer le classificateur - c'est-à-dire le modèle - en le perturbant lorsqu'il fait des prédictions, tandis qu'une violation de sécurité implique la fourniture de données malveillantes qui sont classées comme légitimes. Une attaque ciblée tente de permettre une intrusion ou une perturbation spécifique, ou au contraire de créer un chaos général.

B. Méthodes utilisées de classification :

→ Les Modèles de Classification:

Pour notre analyse et afin de répondre au besoin relevé précédemment, nous avons décidé d'adopter et utiliser les modèles ci-dessous.

Pour commencer, il fallait préparer notre dataset. Nous avons divisé notre dataset en trois classes et en deux mini-datasets. En donnant 80% pour le training et 20% pour le test.

Nous avons aussi déterminé le Label de chaque image qui détermine exactement la catégorisation de l'image.

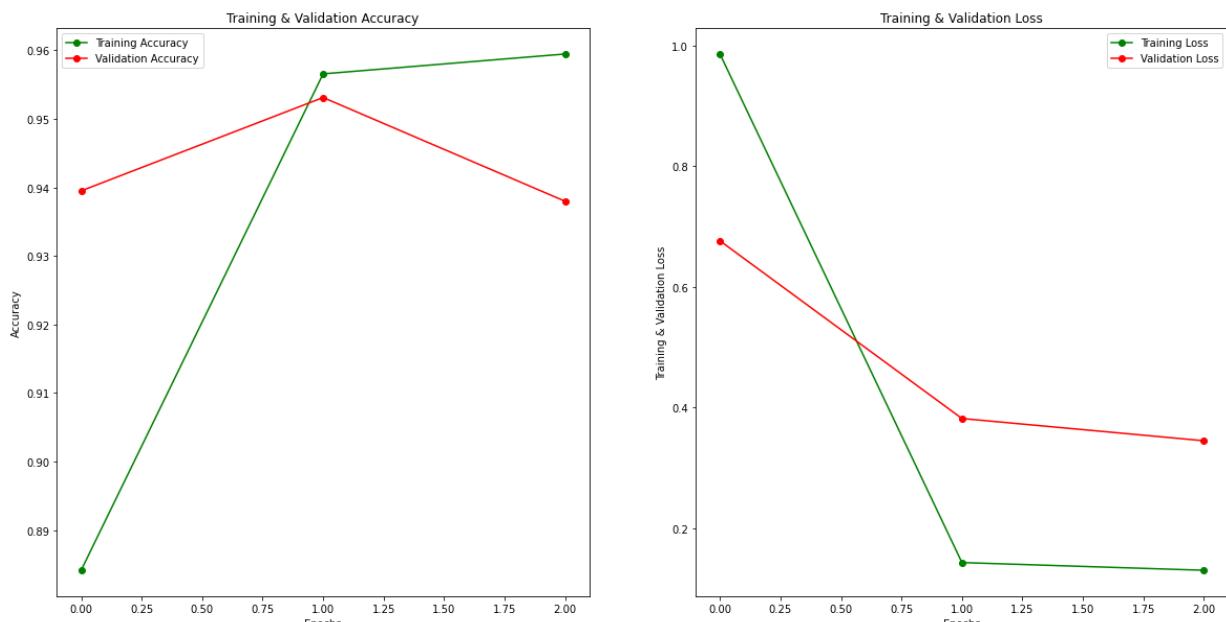
- **Le Modèle CNN (From Scratch):**

```
model = tf.keras.models.Sequential([
    layers.BatchNormalization(),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.3),
    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(256, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.3),
    layers.Conv2D(512, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.15),
    layers.Dense(3, activation= 'softmax')
])
```

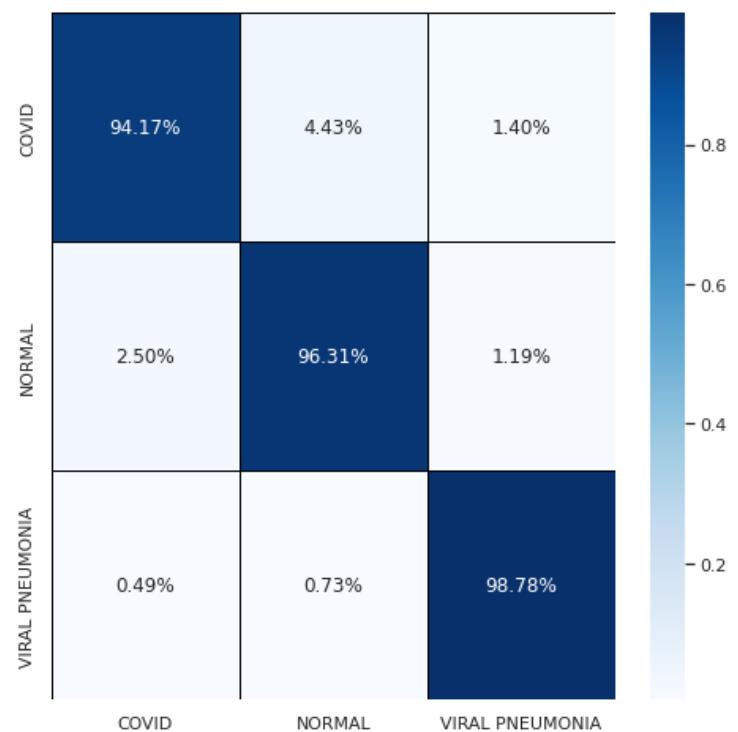
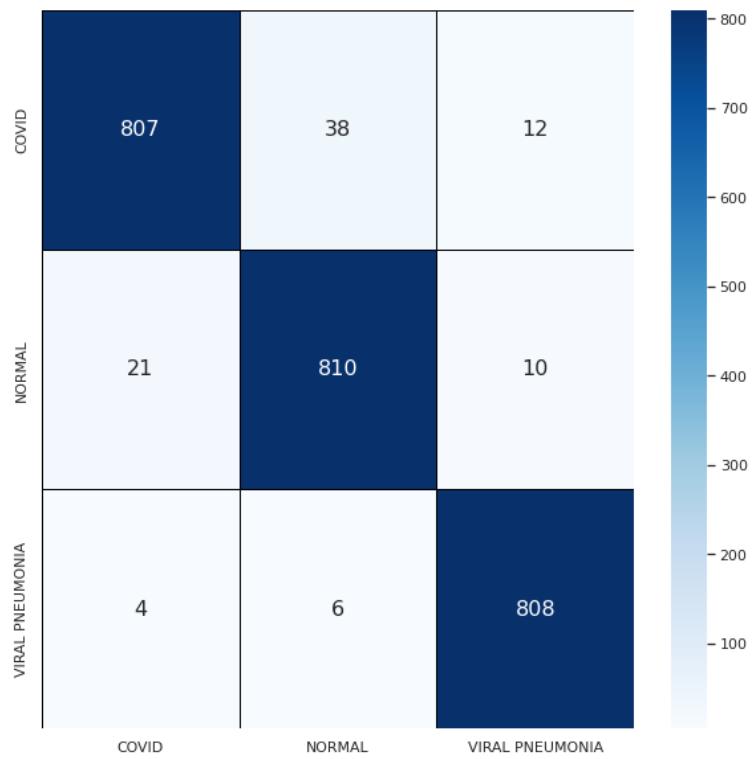
- Le Procès du Training:

```
Epoch 1/3
87/87 [=====] - 1371s 16s/step - loss: 0.9860 - accuracy: 0.8841 - val_loss: 0.6762 - val_accuracy: 0.9395
Epoch 2/3
87/87 [=====] - 1379s 16s/step - loss: 0.1427 - accuracy: 0.9566 - val_loss: 0.3816 - val_accuracy: 0.9531
Epoch 3/3
87/87 [=====] - 1387s 16s/step - loss: 0.1299 - accuracy: 0.9594 - val_loss: 0.3447 - val_accuracy: 0.9380
```

- Évaluer les performances des solutions:



- Confusion Matrices:



- Interprétation des résultats et discussion:

(Les modèles prédéfinis)

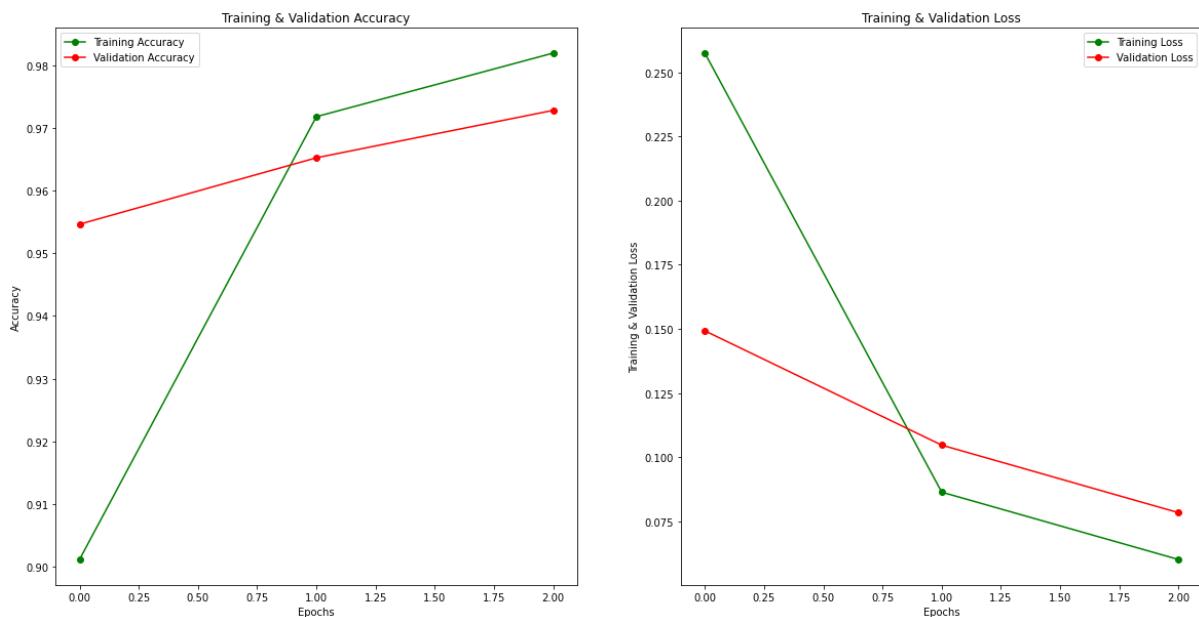
- **MobileNetV2:**

```
base_model = tf.keras.applications.MobileNetV2(input_shape=(224,224,3), include_top = False, weights = "imagenet")
for layer in base_model.layers:
    layer.trainable = False
model = tf.keras.Sequential([base_model,tf.keras.layers.GlobalAveragePooling2D(),
                            tf.keras.layers.Dropout(0.2),
                            tf.keras.layers.Dense(3, activation = "softmax")])
model.summary()
```

- Le Procès du Training:

```
Epoch 1/3
87/87 [=====] - 1371s 16s/step - loss: 0.9860 - accuracy: 0.8841 - val_loss: 0.6762 - val_accuracy: 0.9395
Epoch 2/3
87/87 [=====] - 1379s 16s/step - loss: 0.1427 - accuracy: 0.9566 - val_loss: 0.3816 - val_accuracy: 0.9531
Epoch 3/3
87/87 [=====] - 1387s 16s/step - loss: 0.1299 - accuracy: 0.9594 - val_loss: 0.3447 - val_accuracy: 0.9380
```

- Évaluer les performances des solutions:



- **VGG16:**

```
base_model2=tf.keras.applications.VGG16(
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000,
    classifier_activation='softmax'
)
for layer in base_model2.layers:
    layer.trainable = False
model2 = tf.keras.Sequential([base_model,tf.keras.layers.GlobalAveragePooling2D(),
                            tf.keras.layers.Dropout(0.2),
                            tf.keras.layers.Dense(3, activation = "softmax")])
```

- Le Procès du Training:

```

Epoch 1/3
87/87 [=====] - 1371s 16s/step - loss: 0.9860 - accuracy: 0.8841 - val_loss: 0.6762 - val_accuracy: 0.9395
Epoch 2/3
87/87 [=====] - 1379s 16s/step - loss: 0.1427 - accuracy: 0.9566 - val_loss: 0.3816 - val_accuracy: 0.9531
Epoch 3/3
87/87 [=====] - 1387s 16s/step - loss: 0.1299 - accuracy: 0.9594 - val_loss: 0.3447 - val_accuracy: 0.9380

```

- **VGG19:**

```

base_model2=tf.keras.applications.VGG16(
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000,
    classifier_activation='softmax'
)
for layer in base_model2.layers:
    layer.trainable = False
model2 = tf.keras.Sequential([base_model,tf.keras.layers.GlobalAveragePooling2D(),
                             tf.keras.layers.Dropout(0.2),
                             tf.keras.layers.Dense(3, activation = "softmax")])

```

- Le Procès du Training:

```

Epoch 1/3
87/87 [=====] - 1371s 16s/step - loss: 0.9860 - accuracy: 0.8841 - val_loss: 0.6762 - val_accuracy: 0.9395
Epoch 2/3
87/87 [=====] - 1379s 16s/step - loss: 0.1427 - accuracy: 0.9566 - val_loss: 0.3816 - val_accuracy: 0.9531
Epoch 3/3
87/87 [=====] - 1387s 16s/step - loss: 0.1299 - accuracy: 0.9594 - val_loss: 0.3447 - val_accuracy: 0.9380

```

- **ResNet152V2:**

```

base_model4=tf.keras.applications.ResNet152V2(
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000,
    classifier_activation='softmax'
)
for layer in base_model4.layers:
    layer.trainable = False
model4 = tf.keras.Sequential([base_model,tf.keras.layers.GlobalAveragePooling2D(),
                             tf.keras.layers.Dropout(0.2),
                             tf.keras.layers.Dense(3, activation = "softmax")])

```

- Le Procès du Training:

```

Epoch 1/3
87/87 [=====] - 220s 2s/step - loss: 0.1755 - accuracy: 0.9370 - val_loss: 0.0986 - val_accuracy: 0.9667
Epoch 2/3
87/87 [=====] - 211s 2s/step - loss: 0.0697 - accuracy: 0.9783 - val_loss: 0.0783 - val_accuracy: 0.9773
Epoch 3/3
87/87 [=====] - 216s 2s/step - loss: 0.0549 - accuracy: 0.9804 - val_loss: 0.0688 - val_accuracy: 0.9758

```

C. Méthodes utilisées de l'attaque:

1- L'importance d'utilisation de l'attaque

L'Adversarial Machine Learning est une technique d'apprentissage automatique qui tente d'exploiter des modèles en tirant parti des informations de modèle pouvant être obtenues et en les utilisant pour créer des attaques malveillantes. La raison la plus courante est de provoquer un dysfonctionnement dans un modèle d'apprentissage automatique.

La plupart des techniques d'apprentissage automatique ont été conçues pour fonctionner sur des ensembles de problèmes spécifiques dans lesquels les données d'apprentissage et de test sont générées à partir de la même distribution statistique (IID). Lorsque ces modèles sont appliqués au monde réel, les **adversaires** peuvent fournir des données qui violent cette hypothèse statistique.

Ces données peuvent être organisées pour exploiter des **vulnérabilités** spécifiques et compromettre les résultats. Les quatre stratégies d'apprentissage automatique contradictoires les plus courantes sont l'évasion, l'empoisonnement, le vol de modèle (extraction) et l'inférence.

2- Benchmark des attaques disponibles

Il existe une grande variété d'attaques différentes qui peuvent être utilisées contre les systèmes d'apprentissage automatique.

Beaucoup d'entre eux fonctionnent à la fois sur des systèmes d'apprentissage en profondeur et sur des modèles d'apprentissage automatique traditionnels tels que les SVM et la régression linéaire.

Un échantillon de haut niveau de ces types d'attaques comprend :

- **Adversarial Examples**
- Trojan Attacks / Backdoor Attacks
- Model Inversion
- Membership Inference

Dans notre cas, nous allons utiliser les Adversarial Examples. Afin de choisir l'attaque qui nous importe le plus, nous avons effectué un BenchMark des différentes attaques.

Un Adversarial Exemple fait référence à une entrée spécialement conçue qui est conçue pour sembler "normale" aux humains mais qui provoque une mauvaise classification dans un modèle d'apprentissage automatique. Souvent, une forme de "bruit" spécialement conçue est utilisée pour élucider les erreurs de classification.

Parmi les techniques actuelles utilisées pour générer des exemples contradictoires, on trouve :

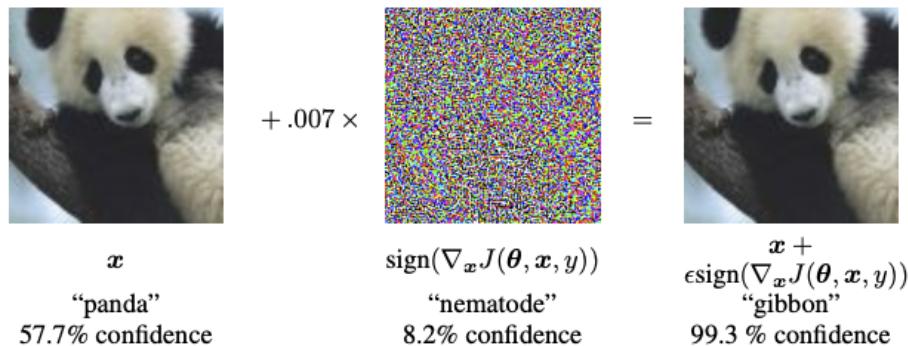
- Gradient-based evasion attack
- **Fast Gradient Sign Method (FGSM)**
- Projected Gradient Descent (PGD)
- Carlini and Wagner (C&W) attack
- Adversarial patch attack

Dans notre cas, nous allons utiliser la méthode du FGSM afin d'attaquer notre machine.

3- Attaque avec FGSM

Premièrement, nous avons importé les packages nécessaires. Définit une fonction Generate_image_adversary qui va nous permettre de perturber notre Mini-Batch d'images.

Le processus de perturbation des images en utilisant la méthode du FGSM est la suivante :



Nous avons choisi cette méthode vu que dans notre cas, nous avons des images / radiographies de personnes! L'erreur au niveau de ces images pourraient fausser tout le diagnostic, donc il fallait essayer de perturber les images, vu que parfois une image pourrait avoir une poussière au dessus ou l'image pourrait être dans un mauvais état, il faut s'assurer que la machine délivre le bon résultat.

L'erreur de la machine risque de fausser le diagnostic, ainsi ne pas bien soigner le patient.

```
# import the necessary packages
from tensorflow.keras.losses import MSE
import tensorflow as tf
def generate_image_adversary(model, image, label, eps=2 / 255.0):
    # cast the image
    image = tf.cast(image, tf.float32)
    # record our gradients
    with tf.GradientTape() as tape:
        # explicitly indicate that our image should be tacked for
        # gradient updates
        tape.watch(image)
        # use our model to make predictions on the input image and
        # then compute the loss
        pred = model(image)
        loss = MSE(label, pred)
    # calculate the gradients of loss with respect to the image, then
    # compute the sign of the gradient
    gradient = tape.gradient(loss, image)
    signedGrad = tf.sign(gradient)
    # construct the image adversary
    adversary = (image + (signedGrad * eps)).numpy()
    # return the image adversary to the calling function
    return adversary
```

Ensuite, nous allons définir une fonction qui va nous permettre de générer un batch en précisant comme entrée le modèle, les images, les labels ainsi que les dimensions.

Cette fonction nous permet de générer et définir un batch que nous allons utiliser afin de réaliser notre attaque. Dans notre cas, nous avons adopté une approche où toutes nos images seront perturbées d'où l'utilisation du len(testX). L'epsilon choisi est un epsilon égale à 0.1.

```
import numpy as np
def generate_adversarial_batch(model, total, images, labels, dims,
                                eps=0.01):
    # unpack the image dimensions into convenience variables
    (h, w, c) = dims
    # we're constructing a data generator here so we need to loop
    # indefinitely
    while True:
        # initialize our perturbed images and labels
        perturbImages = []
        perturbLabels = []
        # randomly sample indexes (without replacement) from the
        # input data
        idxs = np.random.choice(range(0, len(images)), size=total,
                               replace=False)
        # loop over the indexes
        for i in idxs:
            # grab the current image and label
            image = images[i]
            label = labels[i]
            # generate an adversarial image
            adversary = generate_image_adversary(model,
                                                   image.reshape(1, h, w, c), label, eps=eps)
            # update our perturbed images and labels lists
            perturbImages.append(adversary.reshape(h, w, c))
            perturbLabels.append(label)
        # yield the perturbed images and labels
        yield (np.array(perturbImages), np.array(perturbLabels))
```

Nous lançons ainsi le training de notre mini-batch d'images normales et ensuite nous lançons le training de nos images perturbées en utilisant la méthode FGSM.

Nous mesurons ainsi le taux d'accuracy pour la classification des images normales et pour la classification des images perturbées.

```
# make predictions on the testing set for the model trained on
# non-adversarial images
(loss, acc) = model.evaluate(x=testX, y=testY, verbose=0)
print("[INFO] normal testing images:")
print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
# generate a set of adversarial from our test set
print("[INFO] generating adversarial examples with FGSM...\n")
(advX, advY) = next(generate_adversarial_batch(model, len(testX),
                                                testX, testY, (28, 28, 1), eps=0.1))
# re-evaluate the model on the adversarial images
(loss, acc) = model.evaluate(x=advX, y=advY, verbose=0)
print("[INFO] adversarial testing images:")
print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
```

- Le Résultat de l'attaque:

Après le training, nous observons le résultat suivant. En effet, le taux d'accuracy pour **les images normales** en utilisant le modèle du CNN est de : **0.9747**.

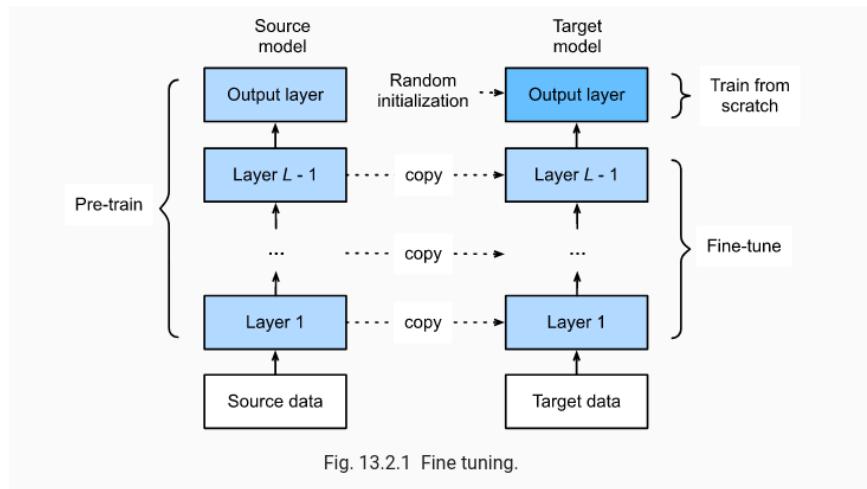
Par contre, le taux d'accuracy pour **les images perturbées** en utilisant le même modèle est de : **0.0672**.

```
[INFO] normal testing images:  
[INFO] loss: 0.0755, acc: 0.9747  
  
[INFO] generating adversarial examples with FGSM...  
  
[INFO] adversarial testing images:  
[INFO] loss: 5.0790, acc: 0.0672
```

Un taux d'accuracy qui atteint les 0.0672 est très réduit par rapport au taux d'accuracy qui avait atteint les 0.9747 précédemment. Donc presque tous les résultats de la classification sont erronés.

D. Méthodes utilisées de défense :

Après avoir effectué une attaque sur notre modèle CNN, nous avons eu comme conclusion que notre modèle n'est pas du tout préparé pour faire face aux attaques. Afin de rendre notre modèle plus performant et l'entraîner à faire face à toute attaque possible, nous avons décidé d'utiliser la technique de défense suivante.



En effet, le fine tuning est une façon d'appliquer ou d'utiliser l'apprentissage par transfert. Plus précisément, le fine tuning est un processus qui prend un modèle qui a déjà été formé pour une tâche donnée, puis règle ou ajuste le modèle pour lui faire effectuer une deuxième tâche similaire.

- L'application du fine-tuning:

Afin d'effectuer un fine-tuning, nous avons décidé de réutiliser nos données en réduisant au maximum le temps que va nécessiter cette action.

Donc on utilise notre modèle et on utilise un Optimizer dont le nom est Adam. Il est à noter que quand on parle d'Adam Optimizer, on parle d'est une méthode de descente de gradient stochastique basée sur l'estimation adaptative des moments de premier et de second ordre.

On définit alors les inputs du modèle dont on aura besoin tel que la série d'images advX et la série d'images advY, ainsi que la taille de notre batch et le nombre d'epochs.

```
# lower the learning rate and re-compile the model (such that we can
# fine-tune it on the adversarial images)
print("[INFO] re-compiling model...")
opt = Adam(lr=1e-4)
model.compile(loss="categorical_crossentropy", optimizer=opt,
    metrics=["accuracy"])
# fine-tune our CNN on the adversarial images
print("[INFO] fine-tuning network on adversarial examples...")
model.fit(advx, advy,
    batch_size=64,
    epochs=10,
    verbose=1)
```

Ci-dessous est alors le training de notre machine en compilant le modèle en utilisant la méthode du fine-tuning. Ça dure alors pendant 10 Epochs et on atteint de très bons résultats d'accuracy.

```
[INFO] re-compiling model...
[INFO] fine-tuning network on adversarial examples...
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
    super(Adam, self).__init__(name, **kwargs)
157/157 [=====] - 6s 31ms/step - loss: 1.4494 - accuracy: 0.6728
Epoch 2/10
157/157 [=====] - 5s 31ms/step - loss: 0.1970 - accuracy: 0.9418
Epoch 3/10
157/157 [=====] - 5s 31ms/step - loss: 0.0685 - accuracy: 0.9811
Epoch 4/10
157/157 [=====] - 5s 31ms/step - loss: 0.0392 - accuracy: 0.9900
Epoch 5/10
157/157 [=====] - 5s 31ms/step - loss: 0.0252 - accuracy: 0.9949
Epoch 6/10
157/157 [=====] - 5s 31ms/step - loss: 0.0179 - accuracy: 0.9964
Epoch 7/10
157/157 [=====] - 5s 31ms/step - loss: 0.0147 - accuracy: 0.9974
Epoch 8/10
157/157 [=====] - 5s 31ms/step - loss: 0.0118 - accuracy: 0.9979
Epoch 9/10
157/157 [=====] - 5s 30ms/step - loss: 0.0096 - accuracy: 0.9983
Epoch 10/10
157/157 [=====] - 5s 31ms/step - loss: 0.0089 - accuracy: 0.9986
<keras.callbacks.History at 0x7f3c09a58d50>
```

Nous allons faire afficher les résultats de notre classification du modèle après l'implémentation de la méthode fine-tuning dans le cas d'une classification des images normales et le cas d'une classification des images perturbées.

```
# now that our model is fine-tuned we should evaluate it on the test
# set (i.e., non-adversarial) again to see if performance has degraded
(loss, acc) = model.evaluate(x=testX, y=testY, verbose=0)
print("")
print("[INFO] normal testing images *after* fine-tuning:")
print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
# do a final evaluation of the model on the adversarial images
(loss, acc) = model.evaluate(x=advX, y=advY, verbose=0)
print("[INFO] adversarial images *after* fine-tuning:")
print("[INFO] loss: {:.4f}, acc: {:.4f}\n".format(loss, acc))
```

- Le Résultat de la défense:

Après avoir fait un training de la machine, et essayer de classifier les images perturbées par notre attaque précédente en utilisant la méthode du fine-tuning.

Nous observons ci dessous le résultat :

```
[INFO] normal testing images *after* fine-tuning:
[INFO] loss: 0.0652, acc: 0.9802

[INFO] adversarial images *after* fine-tuning:
[INFO] loss: 0.0034, acc: 0.9996
```

Ce qu'on remarque c'est que la méthode du fine-tuning ne nous a pas seulement aidé à faire éléver notre accuracy dans le cas d'un mini batch d'images attaquées et perturbées. Nous avons eu comme résultat une acc de 0.99.

Par contre pour les images normales, nous avons eu comme résultat un taux d'accuracy de 0.98.

Conclusion

Dans le domaine du deep learning, la classification d'images et son application ont fait de grands progrès ces dernières années. D'une part, les cercles universitaires ont fait de grands efforts pour concevoir une variété de modèles CNN efficaces, qui ont atteint une grande précision et ont même dépassé la capacité de reconnaissance humaine. D'autre part, l'application du modèle CNN dans l'analyse d'images médicales est devenue l'une des directions les plus attrayantes de l'apprentissage en profondeur. La technologie actuelle d'apprentissage en profondeur a obtenu des résultats de recherche dans le domaine de l'imagerie par ultrasons tels que le cancer du sein, les artères cardiovasculaires et carotides.

Le machine learning applique largement tous les aspects de l'analyse d'images médicales, y compris l'ophtalmologie, la neuroimagerie, l'échographie, etc. Avec le développement de cette technologie, de plus en plus de domaines médicaux s'appliqueront.

La raison pour laquelle le deep learning peut se développer si rapidement dans le domaine médical est indissociable d'un grand nombre de pratiques cliniques. Comment mieux appliquer le machine learning à toutes les étapes du traitement médical devient une tâche plus difficile. Cela dépend de deux aspects: l'un est l'itération constamment mise à jour de la technologie et l'autre est l'accumulation continue d'expérience médicale.

À l'heure actuelle, un certain nombre d'excellents algorithmes ont émergé dans les domaines de driverlessness (l'absence de conducteur), du traitement du langage naturel, de computer vision, etc. Ces algorithmes ont attiré une grande attention dans leurs domaines respectifs, et la façon d'utiliser ces algorithmes avancés d'apprentissage en profondeur est un aspect digne de la réflexion et l'innovation constantes de nos chercheurs.

Recommendations

Dans notre projet en Computer Vision, nous avons pu classifier les imageries des poumons afin de déterminer l'état du patient. Ceci nous permet d'avoir un diagnostic en moins de temps et prédire même l'état de santé du patient en prenant en considération d'autres éléments.



Nous avons pu imaginer la continuité de notre projet ou on créera tout un système qui sera implanté au niveau de chaque hôpital. Ce système va d'abord détecter si le patient a bel et bien une maladie au niveau de ces poumons, ceci détectée, le système pourrait ainsi avoir :

- Un capacité de généralisation élevée et validation clinique rigoureuse.
- Un triage précis et rapide qui permet l'identification et l'alerte en temps réel des cas suspects de pneumonie, pour une protection optimale du personnel médical et un traitement rapide des patients.
- Une analyse précise des lésions c'est -à -dire que les informations sur les lésions telles que le volume et l'emplacement sont disponibles en détail avec des visualisations intelligentes.
- Une évaluation automatique de l'état. En effet, les différences entre les analyses précédentes et actuelles peuvent être identifiées avec précision pour une évaluation précise du développement et des études de suivi.
- Réalisation des rapports structurés : génération automatique de rapports conformément aux dernières directives.

Références:

<https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/>

<https://fullscale.io/blog/machine-learning-computer-vision/>

<https://www.kaggle.com/tawsifurrahman/covid19-radiography-database>