

# Table des matières

[**Table des matières**](#_heading=h.j302fkub3avk) **2**

[Introduction](#_heading=h.gjdgxs) **4**

[But](#_heading=h.30j0zll) 4

[Méthode expérimentale](#_heading=h.3znysh7) **4**

[Choix des algorithmes de détection](#_heading=h.2et92p0) 4

[Description du projet](#_heading=h.tyjcwt) 4

[**Théorie et Algorithmes**](#_heading=h.dxnyo3i72y6b) **5**

[Principes généraux](#_heading=h.824vprnxqh63) 5

Préparation des données

Algo 1 : Convolutional Neural Network (CNN) 5

[Fonctionnement](#_heading=h.ww0bvgiki2tb) 5

Architecture du modèle

Choix techniques

[Jupyter book](#_heading=h.37ufa6o8a5qk) 5

[Algo 2](#_heading=h.4f4rpjmdyya8) : Boosted Tree 6

[Fonctionnement](#_heading=h.1yevc0syssya) 6

Architecture du modèle

Choix techniques

[Jupyter book](#_heading=h.y3nfogvvhdwo) 6

[**Resultats**](#_heading=h.hlc2vqj1zudd) **7**

[Algo 1](#_heading=h.ch27dyn91eu3) 7

[Tableaux de resultats](#_heading=h.bmrib7wu22e) 7

[Graphiques](#_heading=h.1y810tw) 7

[Incertitudes / Commentaires](#_heading=h.bj07wcjwkje6) 7

[Algo 2](#_heading=h.a8fm5fe7hcq6) 7

[Tableaux de resultats](#_heading=h.tcpnn0vcrea4) 7

[Graphiques](#_heading=h.t5nwdnunx20b) 7

[Incertitudes / Commentaires](#_heading=h.85xj7hsx41ip) 7

[Discussions](#_heading=h.4i7ojhp) comparer les 2 resultats? **8**

[Conclusion](#_heading=h.2xcytpi) **8**

difficultés rencontrees

organisation du groupe

ressenti de cette intro à l’IA

# Introduction

## But

Le but de ce projet est de pouvoir déterminer à l’aide d’intelligence artificielle, si un poumon est sain, ou s’il est atteint d’une pneumonie, et s’il est malade, déterminer la nature de cette maladie, qu’elle soit virale ou bactérienne.

# Méthode expérimentale

## Choix des algorithmes de détection

Au départ nous avions envisagé d’utiliser un réseau de neurones artificiel (ANN) mais on a conclu que nous nous retrouverions avec une quantité énorme de paramètres et donc un temps d’apprentissage trop long. Par ailleurs il aurait fallu aplanir un vecteur unidimensionnel l’image avant de les faire passer dans le réseau et on perdrait toutes les informations bi-dimensionnels.

* K-Nearest Neighbor algorithm (KNN)

La “méthode des k plus proches voisins” est un algorithme de classification ou de régression permettant l’analyse d’images représenter dans l’espace (distance entre deux points d’une image par exemple). L’avantage de cet algorithme est sa capacité à pouvoir traiter de multiples classes (A.K.A: Chats, Chiens, Lapins, etc…), cependant, son plus grand point faible et le fait d’utiliser l'entièreté du jeu de données d’apprentissage pour faire une classification, ce qui peut mener à des erreurs de prédictions notamment dû à un jeu de données très petit.

# Artificial neural network (ANN)

Le réseau ANN est le plus simple réseau de neurone existant, il est utilisé pour la reconnaissance faciale, et plus généralement pour de la computer Vision.

Cependant, ce genre de réseau implique des problèmes lié aux performances hardware de la machine qui exécute l’apprentissage

* Boosted tree
* Réseaux de Neurones à Convolution (CNN)

Les réseaux de neurones à convolutions ont été créés sur la base de principes physiques et biologiques, humain et animal. En effet, les neurones de ce réseau sont créés de telle sorte à ce qu’il correspondent à des régions qui se chevauchent lors du pavage duchamp visuel, une mécanique emprunté aux cerveaux humain et animal.

Ce type de réseaux de neurones est majoritairement utilisé pour la reconnaissance d’images, de vidéos, mais également pour les systèmes de recommandation et le traitement du langage naturel.

## Description du projet

Pour ce premier projet en IA il nous a été fourni un jeu de données constitué de 5856 images. Ces images sont des radiographies thoraciques provenant d’un hôpital. Il nous faudra donc trouver quel type de modèle est propice à la classification d’images.

Le dataset est brut et n’a donc pas été préparé pour l'entraînement de nos algorithmes. Ce sera donc à nous de nettoyer ce dataset avant de pouvoir l’exploiter.

Notre modèle devra s'entraîner sur ce dataset pour apporter ensuite des résultats concluants en matière de prédictions sur des images qu’il n’a jamais vu.

Le projet consiste d’une part à découvrir l’importance de trois étapes clés. La préparation des données, le modèle, l'interprétation des résultats.

# Théorie et Algorithmes

Dans cette partie, nous allons expliquer comment fonctionnent les différents algorithmes que nous avons choisis, la théorie qui les régit, de manière à comprendre et interpréter plus facilement les résultats par la suite.

## Principes généraux

Overfitting

Under fitting

Bias

Variance

Descente de gradient

Fonction d’activation

Un neurone

### Préparation des données

1 - Premier aperçu de notre dataset.

La première grande étape à laquelle nous avons été confrontés consiste à rendre le jeu de données que l’on nous a fourni, exploitable. Une fois le dataset nettoyé, il pourra être utilisé et réutilisé par tous nos modèles.

Le dataset qui nous a été fourni est une collection de 5856 images représentant des radiographies thoraciques. Le dataset a été divisé en 3 sous parties : TRAIN , TEST et VAL. Chaque sous partie contient des dossiers libellés NORMAL et PNEUMONIA.

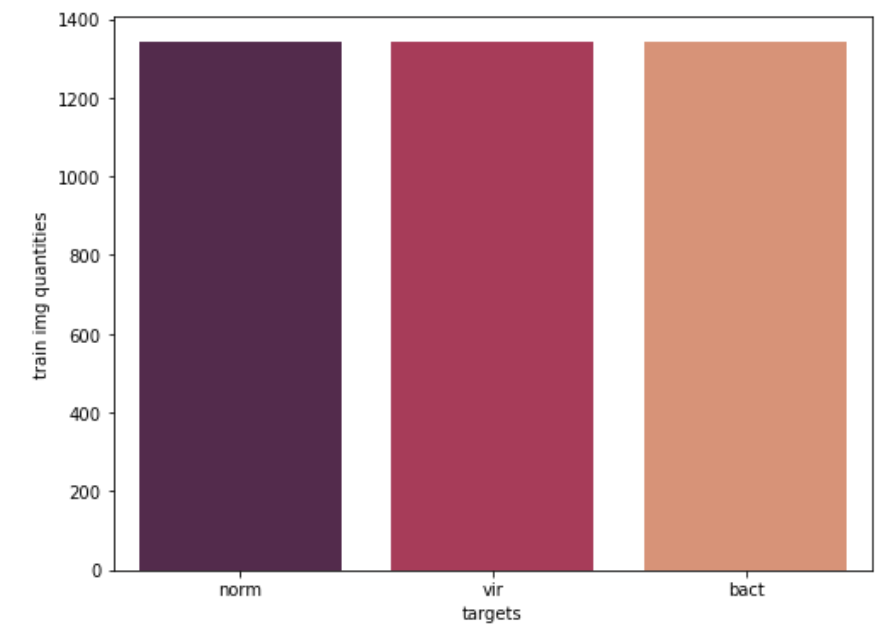
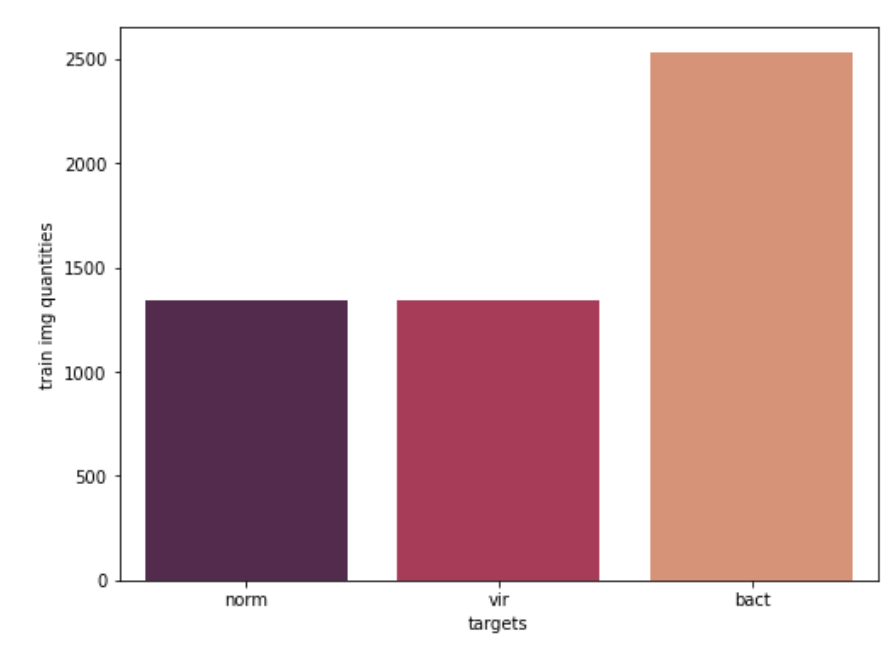
Nous remarquons que les images situées dans les dossiers PNEUMONIA présentent systématiquement le mot « bacteria » ou « virus » dans le nom du fichier. Ceci va nous permettre de distinguer au final 3 targets qui seront Normal, Virus et Bacteria et nous permettront d'entraîner nos modèles sur ces trois critères.

2 - Le nettoyage

Après avoir récolté nos données, nous allons devoir les nettoyer. Cette phase souvent longue mais indispensable, elle permettra de préparer notre dataset pour un entraînement de notre modèle dans de meilleures conditions.

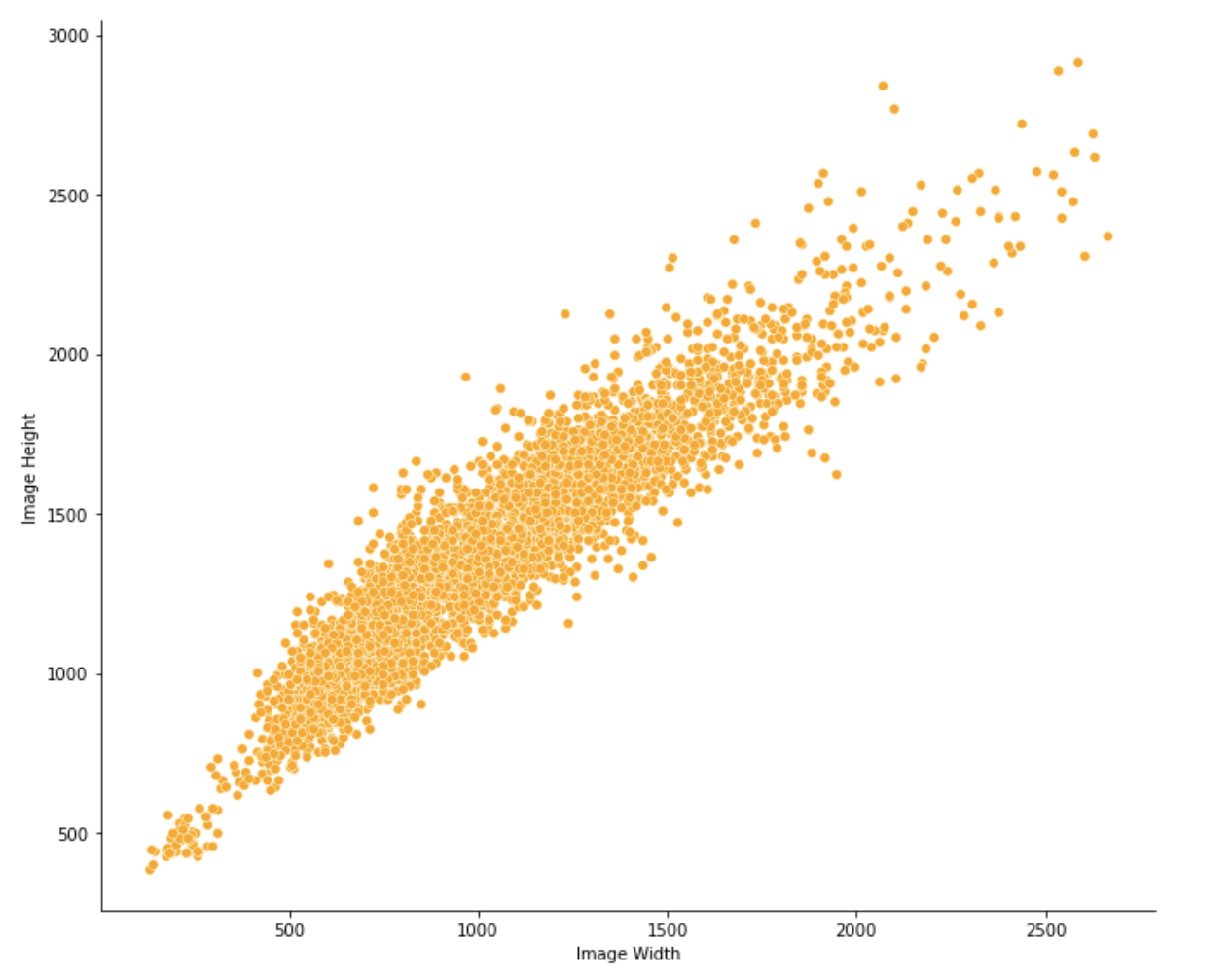
Tout d’abord nous devons nous assurer que la répartition des images au sein de nos 3 targets soit homogène. Un déséquilibre pourrait désavantager une target au profit d’un sur-apprentissage de la target dominante.

Dans notre cas, nous distinguons de fortes disparités dans les dossiers TRAIN et TEST. Le graph de gauche correspondant au dossier TRAIN illustre une dominance d’images Bacteria.



Les proportions sont ensuite ré équilibrées (graph de droite), le dataset est cependant amputé d’environ 1500 images mais nous y remédierons plus tard grâce à la data augmentation.

Pour pouvoir entraîner notre modèle, nous devons nous assurer que toutes les images ont les mêmes dimensions. Là encore, nous avons des surprises comme le témoigne le graph ci dessous



Nous re dimensionnons toutes nos images au format 64x64 pixels. Ce choix fut un bon compromis entre le temps d'entraînement limité par la puissance de calcul de nos machines et les résultats obtenus en matière de précision.

D’après cette publication scientifique (<https://pubs.rsna.org/doi/full/10.1148/ryai.2019190015>), pour de l’imagerie médicale, les meilleurs résultats sont obtenus avec des images entre 256 et 320 pixels de côté.

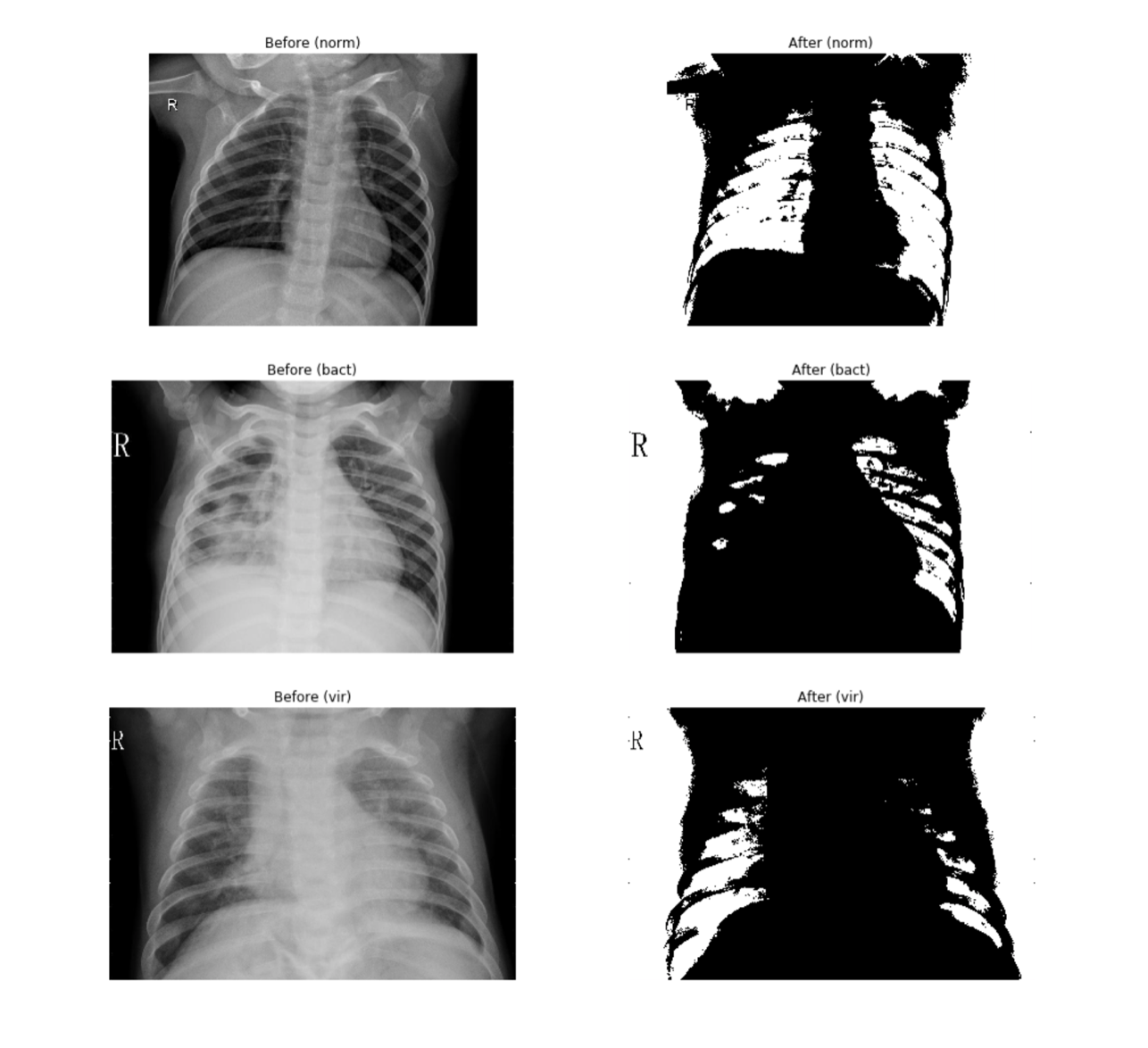
Nos images devront toutes avoir le même nombre de canaux colorimétriques. Nous les convertirons toutes sur 3 canaux, R,G et B.

Chaque pixel de nos images correspond a une valeur comprise entre 0 et 255. Nous normalisons tous nos pixels pour obtenir des valeurs comprises entre 0 et 1 en divisant la valeur de chaque pixel par la valeur maximale, soit 255. Cette opération réduira la complexité du problème que notre modèle essaye de solutionner. Par conséquent, le modèle s'entraîne plus rapidement et pourra potentiellement avoir une meilleure précision.

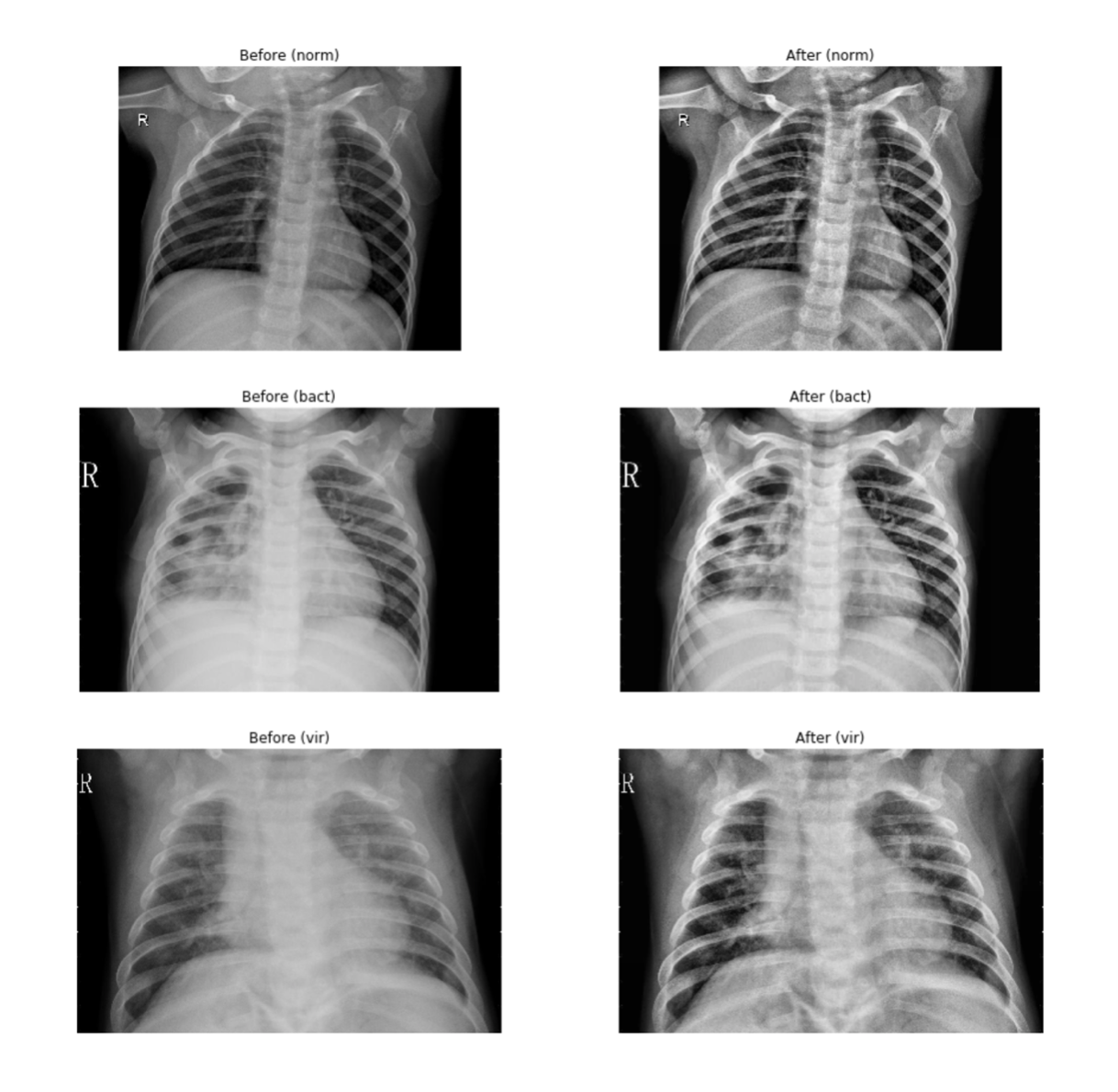
Pour aller un peu plus loin dans la préparation de nos données, nous avons voulu enrichir nos images en leur appliquant des traitements. Le but étant toujours d’optimiser l’apprentissage de notre modèle.

Pour se faire, avons explorés deux approches:

La première consiste à segmenter les images afin de délimiter les pixels de part et d’autre d’une valeur seuil. Le but étant de séparer clairement les poumons sains des poumons malades. Cette technique nous a permis d’obtenir de très bons résultats lorsque notre modèle ne se contentait que de deux targets (normal et pneumonia). En revanche, la segmentation sur un modèle à trois targets n’a pas été convaincante. Le modèle ne parvenait pas à différencier les Bacteria des Virus.



La seconde approche fut celle que nous avons gardé. Celle-ci consistait à optimiser l’équilibre des contrasts afin d’éliminer le bruit et rendre les images plus détaillées. Nous avons utilisé l’algorithme CLAHE (Contrast-Limited Adaptive Histogram Equalization) présent dans la librairie OpenCV. Les résultats ont été concluants.



Ensuite nous voulons rééquilibrer la proportion d’images TEST et d’images TRAIN.

Pour se faire, nous avons redistribué équitablement nos images en trois blocs. Chaque bloc correspond à un tiers de la dataset à l’exception de nos 16 images du dossier VAL. Chaque bloc possède le même nombre d’images NORM, VIR, et BACT. De cette façon nous pourrons faire lors de l'entraînement un k-fold cross-validation. Il suffira de concaténer deux des trois blocs pour former les données de TRAIN, et le bloc restant sera les données de TEST. Soit une proportion de 66% TRAIN, 33% TEST et 1%VAL.

Ce split est cohérent pour un bon entraînement de notre modèle.

Pour qu’un algorithme puisse s'entraîner sur nos images nous devons les libellés. De par le nom de chaque fichier d’image nous allons pouvoir libeller chaque image en fonction des trois critères que nous recherchons : norm, bact, vir.

Le libellé est sous un format de matrice plus communément appelé one-hot encoding. Dans notre cas il se présente comme ceci :

[ 1, 0, 0 ] pour un patient sain

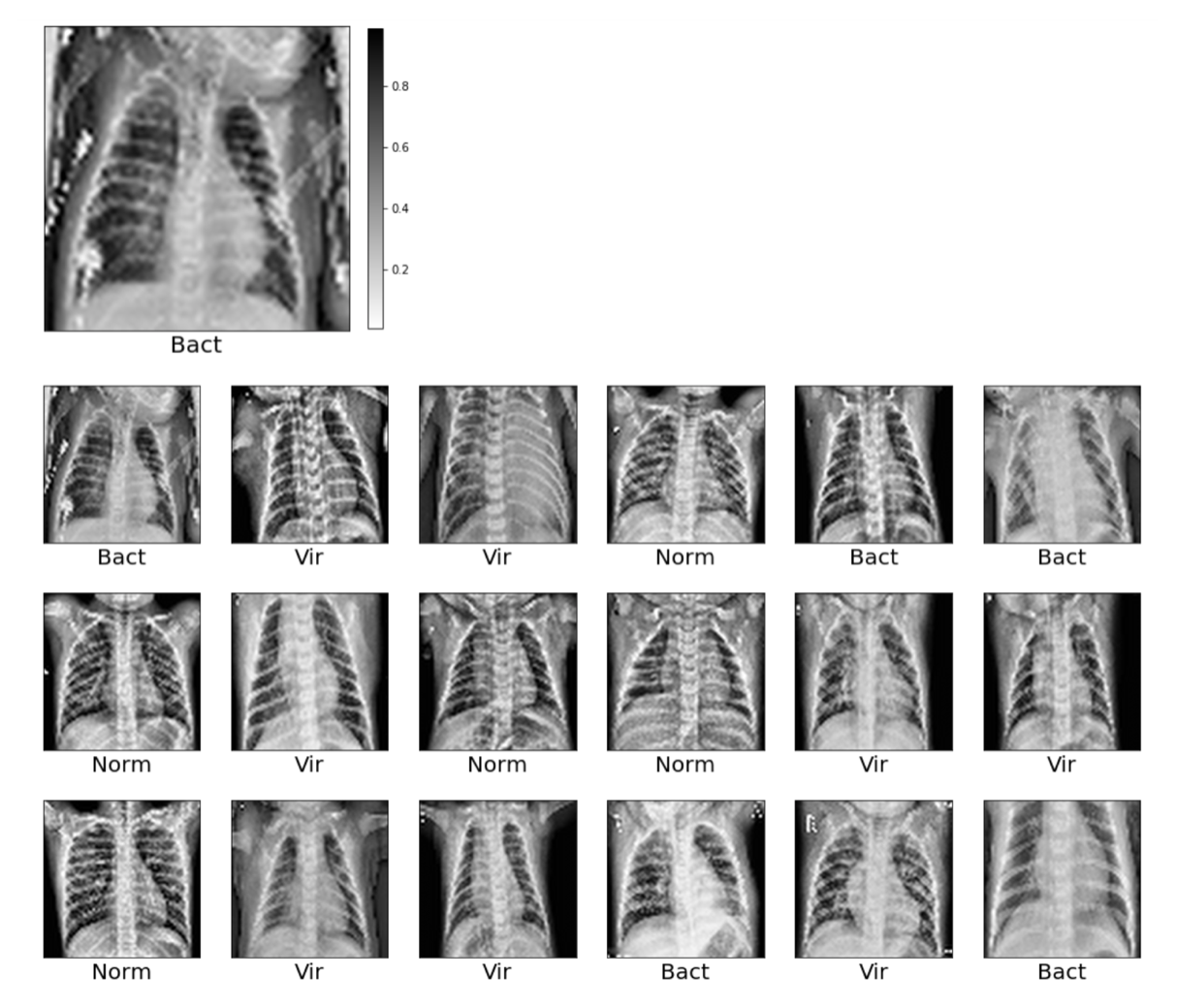
[ 0, 1, 0 ] pour un patient souffrant de pneumonie virale

[ 0, 0, 1 ] pour un patient souffrant de pneumonie bactérienne

Chaque indice correspond donc à une target.

A ce stade nous avons des images de même forme (shape),normalisées, enrichies par CLAHE, et en quantités similaires parmi les trois targets.

Une fois tous ces changements apportés à nos données, nous obtenons des images comme celles de l'échantillon ci-dessous.



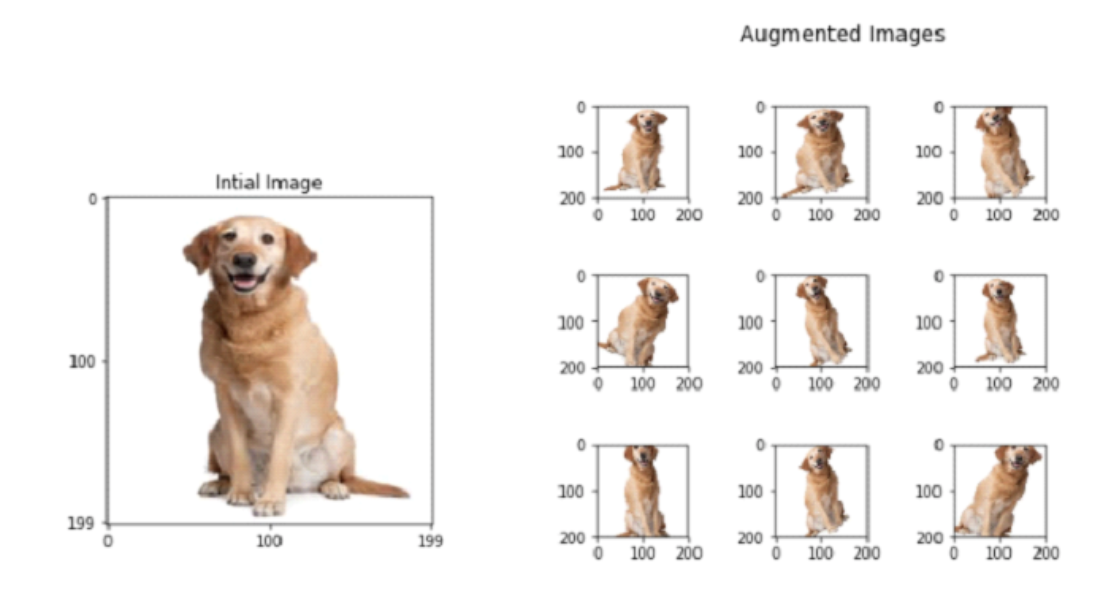
La dernière étape de la préparation de nos données sera de pallier au nombre insuffisant d’images présentes dans le dataset.

Suite au rééquilibrage des proportions correspondant à chaque target, nous avons réduit d’environ 25% le volume notre dataset initial. Nous craignons n’avoir plus assez d’images pour correctement entraîner notre modèle et risque de faire de l’overfitting trop facilement.

Pour éviter ce désagrément nous avons artificiellement généré de nouvelles images à partir de celles déjà présentes dans notre dataset. Les images artificielles présentent une série de modifications légères sans pour autant les dénaturer et conserveront le même label.

C’est via le module ImageDataGenerator de Keras que nous y parviendront.

Les modifications apportées sont diverses comme une rotation, inversion, amincir ou étirer sur un axe, assombrir, zoomer etc… Illustration à titre d’exemple ci-dessous:



## Boosted Tree

### Les différents types d’arbres de décisions

Les **Random Forests** et les **Gradient Boosted Tree** sont des algorithmes de machine learning performants pour tout ce qui est arbre de décision. Ils sont principalement utilisés pour des problèmes de régression ou de classification. Bien que ce soit tout deux des arbres de décisions, ils ne sont pas créés de la même manière, et ont chacun leurs avantages et désavantages.

#### L’arbre de décision

L’arbre de décision est un simple diagramme de prise de décision. L’idée, c’est de traverser un unique arbre de décision avec plusieurs conditions pour atteindre une solution. Cette approche, bien que basique et ne pouvant pas être appliquée à des cas concrets, reste la fondation des Random Forest et des Boosted Trees.

#### Les Random Forests

Les Random Forests utilisent un très grand nombre d’arbres, où tous les résultats de chaque arbre est combiné, souvent par probabilité, pour fournir une probabilité unique.

#### Les Gradient Boosted Trees

## Réseau neuronal à convolution

### Fonctionnement

Pour ce projet, le modèle que nous avons retenu est un réseau neuronal de convolution. Les modèles CNN sont populaires pour faire de la classification ou reconnaissance d’images pour leur excellente précision. Nous comparerons les résultats de notre CNN avec d'autres modèles plus bas dans le rapport afin de confirmer notre choix.

Le concept de la convolution consiste en une petite matrice qui se déplace pour être appliquée à une image entière. Chaque case dans la matrice correspond un poids qu’elle appliquera sur le pixel où elle se trouve. Si l’on additionne les produits de chaque poids avec la valeur de leurs pixels alors on obtient une valeur de sortie. Après avoir parcouru toute l’image, on obtient une nouvelle image de résolution inférieure.

Une couche de convolution est créée lorsque nous appliquons plusieurs matrices aux images d’entrée. Les valeurs des poids seront mis à jour par retro-propagation du gradient et la couche sera entraînée à déterminer les meilleures valeurs de poids de filtre. Illustration ci-dessous :



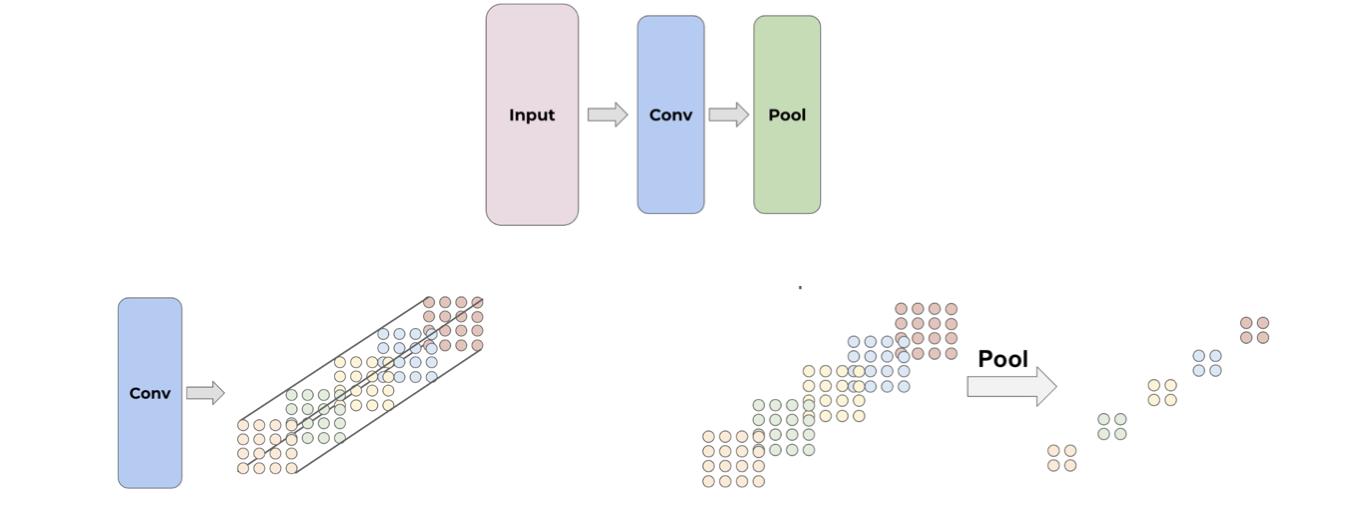
Dans Tensorflow nos couches de convolution auront pour paramètres le nombre de nos matrices sous le nom de « filters », les dimensions de chaque matrice sous « kernel\_size » , puis le pas de déplacement sous « stride ». Nous n’appliquerons pas de bord « padding » à notre image car le bord de nos radiographies ne comportent pas d'éléments utiles à la résolution de notre problème.

Nos couches de convolution seront aussi dotées d’une fonction d’activation ce qui apportera de la non-linéarité dans notre réseau. Sans fonction d’activation, notre CNN ne serait finalement qu’une régression linéaire. Ici nous avons utilisé la fonction d’activation Relu, populaire dans les CNN.

La couche de convolution va nous permettre de réduire le nombre de paramètres grâce à la connectivité locale.

Contrairement à une couche dense, tous les neurones ne sont pas entièrement connectés et seront connectés à un sous-ensemble de neurones locaux dans la couche suivante.

Ensuite nous avons les couches de pooling qui acceptent les couches convolutives comme entrée et vont à nouveau réduire considérablement le nombre de paramètres. Le fonctionnement d’une couche de pooling est assez similaire au filtre de la convolution. Une matrice généralement de 2 par 2 va parcourir le tensor issu de la couche de convolution et à chaque déplacement va recueillir la valeur de pixel la plus grande.

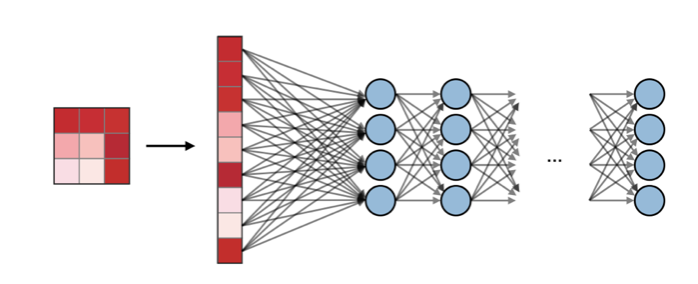


Nous utiliserons une autre technique courante qui s’appelle le dropout. Relativement simple et très efficace pour éviter le sur-apprentissage ou plutôt l’overfitting. Le principe du dropout est de désactiver aléatoirement un pourcentage de neurones. A chaque epoch, la combinaison de neurones sur lequel le modèle s'entraîne sera différente. A savoir, une epoch est une itération d’apprentissage sur le jeu de données complet.

Notre CNN devra ensuite procéder à un flatten. Le but étant de transformer le résultat du dernier pooling en vecteur a une dimension.

Ce nouveau vecteur pourra ensuite être passé aux couches denses de notre réseau de neurones.

Nos couches denses où chaque neurone est connecté avec tous les neurones de la couche suivante. Ceci va accroître considérablement le nombre de paramètres. Là aussi nous appliquons la fonction d’activation Relu.



La dernière couche dense communément appelée output layer aura comme fonction d’activation Softmax car nous avons 3 targets différents. L’output sera dans notre cas un vecteur a une dimensions au format [ x, y, z]. Chaque index correspond à une target. La valeur à chaque index correspond à une probabilité. Softmax va arrondir les probabilités pour obtenir 3 cas possibles :

[ 1, 0, 0 ] pour un patient sain

[ 0, 1, 0 ] pour un patient souffrant de pneumonie virale

[ 0, 0, 1 ] pour un patient souffrant de pneumonie bactérienne

Enfin un optimiseur sera nécessaire pour finaliser notre CNN.

L’optimiseur a pour but d’optimiser le modèle en modifiant ses poids le plus efficacement possible. La fonction de perte avec une descente de gradient va permettre au modèle de progressivement trouver les poids les plus adaptés. Nous avons obtenu de bons résultats avec deux optimiseurs. Adam qui est réputé pour sa rapidité d'entraînement et son efficacité. Puis Nadam, une variante de Adam, qui bénéficie d’un accélérateur de gradient au nom de Nesterov momentum.

**Architecture du modèle**

L’architecture que nous utiliserons :

Nous avons en entrée des images de 64x64x3

Une première couche de convolution à 32 filtres, et un kernel de 4x4

Une première couche de pooling avec un kernel de 2x2

Une seconde couche de convolution à 64 filtres, et un kernel de 2x2

Une seconde couche de pooling avec un kernel de 2x2

Une troisième couche de convolution à 128 filtres, et un kernel de 2x2

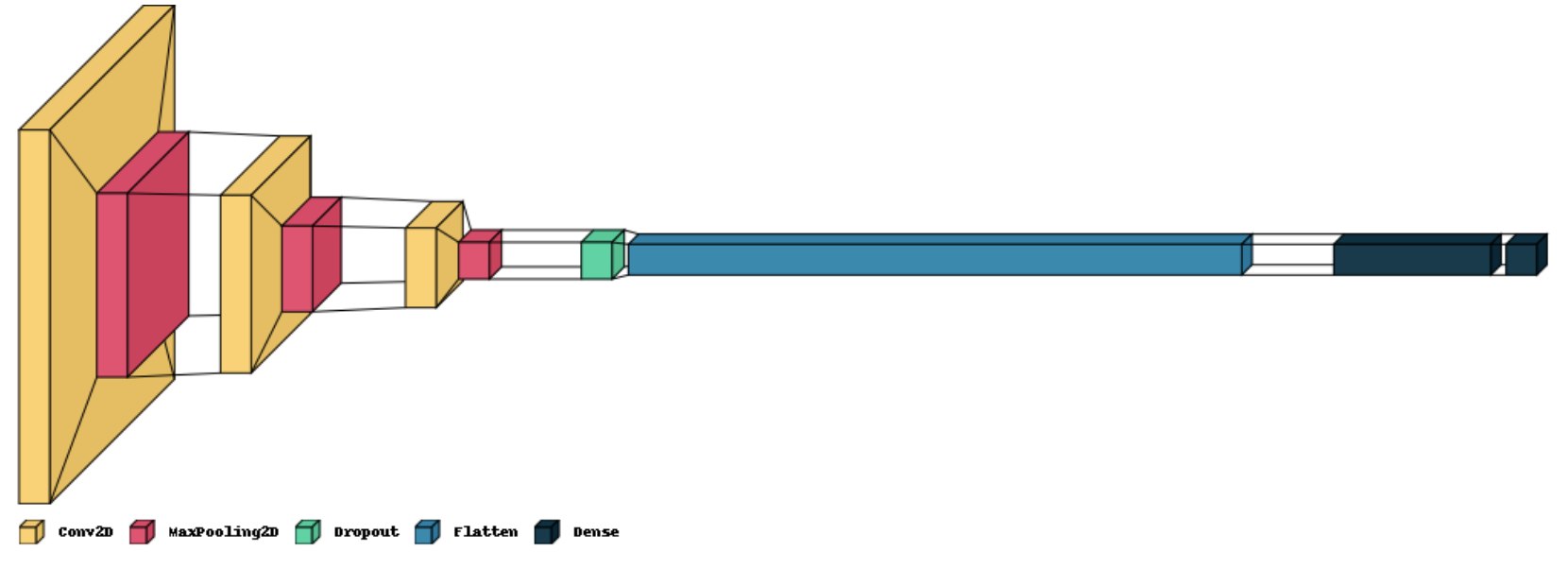
Une troisième couche de pooling avec un kernel de 2x2

Un dropout de 50%

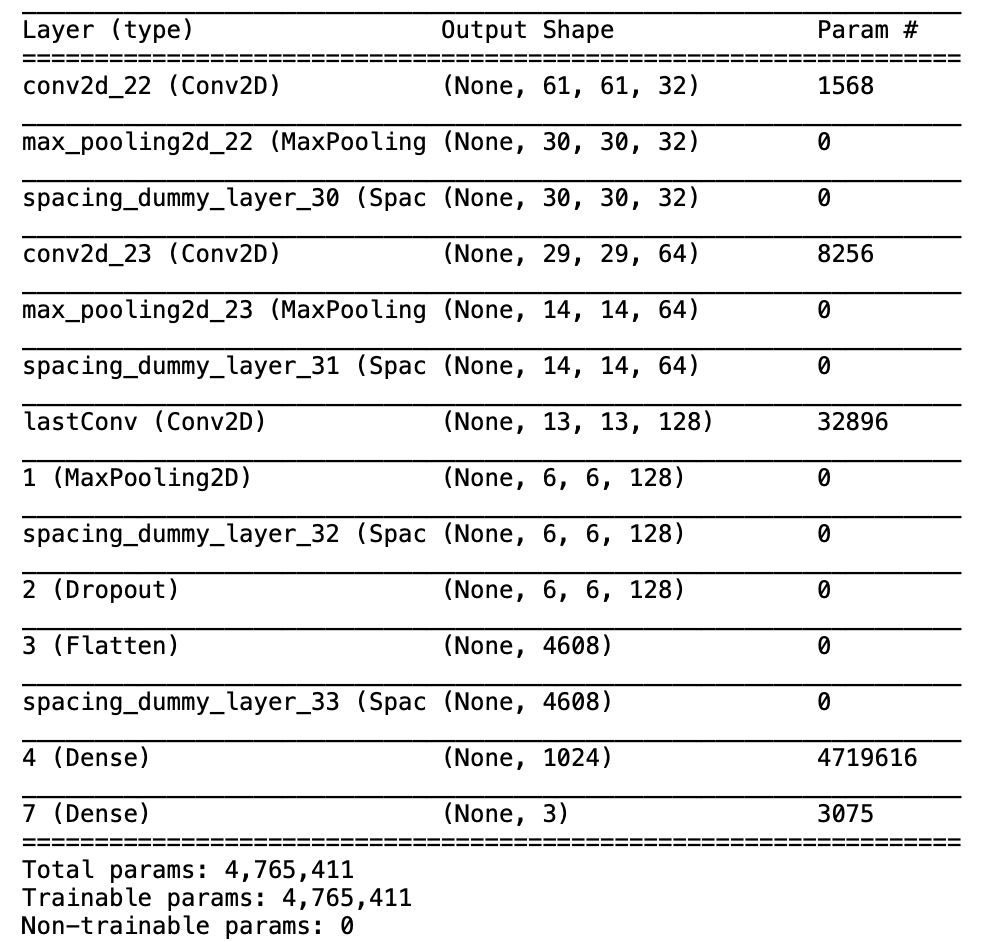
Un flatten

Une couche dense de 1024 neurones

Une couche d’output de 3 neurones



Cette architecture à la fois ni trop simple ni trop complexe nous a permis de maintenir un nombre de paramètres raisonnable.



**Choix techniques**

Environnement :

Nous utiliserons Jupyter Notebook pour écrire notre code, l'exécuter, afficher les résultats et exporter son contenu. Cet environnement de programmation basé sur le web a une prise en main facile et intuitive. Le projet tient sur un seul fichier et est facilement exportable.

Librairie IA :

Côté librairies IA nous utilisons principalement Tensorflow. Soutenu par Google, cette librairie aux ressources débordantes est incontestablement la plus utilisée et se prête parfaitement à notre exercice.

Preprocessing :

La dimension de 64x64x3 pour l’ensemble de nos images. Semble être le seuil pour obtenir des résultats corrects malgré la perte de détail dans les images. Ceci nous a permis d’adapter le temps d’apprentissage à la puissance de nos machines. D’après l’étude <https://pubs.rsna.org/doi/full/10.1148/ryai.2019190015> , la taille d’images pour obtenir un ratio apprentissage/temps d’apprentissage serait de 256x256.

Le train test split, consiste en la proportion d’images destinées au dataset TRAIN et au dataset TEST. Le ratio se situe généralement entre 75/25 et 60/40 pour ce type d’exercice. Nous choisirons ⅔ pour TRAIN et ⅓ pour TEST.

La normalisation de nos pixels sur une échelle entre 0 et 1 afin d’alléger les calculs et de gagner en temps d'entraînement.

Équilibrer les quantités d’images de chaque target et donc avoir autant de NORM que de VIR que de BACT nous a permis d’optimiser l’apprentissage en évitant que le modèle ne s’entraine trop sur une target ou pas assez sur une autre.

Améliorer les contrastes de nos images à l’aide de traitements. Nous utiliserons l’algorithme CLAHE ( contrast limited adaptive histogram equalization) pour amplifier les contrastes dans nos images afin de faire ressortir des poumons les zones malades. Nous constaterons un gain en accuracy à la suite de ce traitement appliqué à notre dataset.

Le one-hot encoding lorsque nous avons libellé les images pour garder un cohérence avec l’output softmax de notre modèle.

Params et HyperParams :

Le Batch size qui correspond au nombre d’images qui seront utilisées pour parcourir le modèle au cours de chaque réajustement des poids. Nous utiliserons un batch size de 8 pour gagner en précision et réduire le bruit lors de l’entraînement au détriment de sa durée.

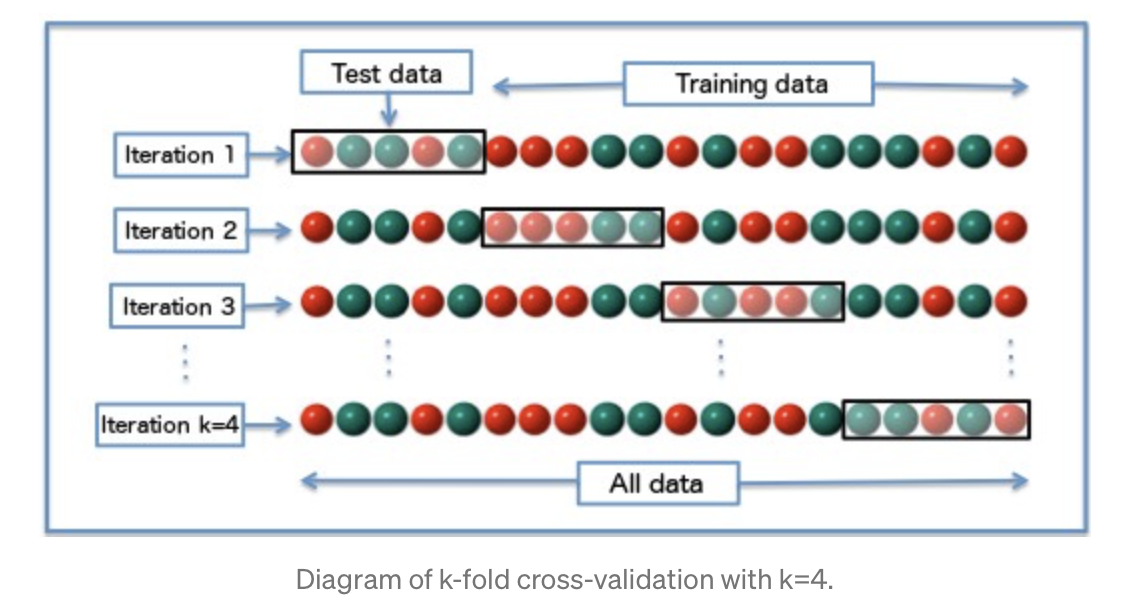
Les Epochs correspondent au passage complet de toutes les batchs dans le modèle. Ici nous avons fixé 100 epochs pour entraîner notre modèle. Soit suffisamment pour que la courbe d’accuracy, d’auc, et de loss s'aplanissent suffisamment pour en déduire que le modèle a appris correctement.

Data augmentation :

Le dataset ne dépassant pas les 6000 images nous semblait trop limité pour un bon apprentissage. Nous avons choisi d’augmenter artificiellement le nombre d’images de notre dataset pour à la fois gagner en accuracy et lutter contre l’overfitting.

K-fold cross validation :

Ce principe de validation croisé va nous permettre de faire trois entraînements à la suite. Chaque entraînement utilisant un dataset TRAIN et TEST diffèrent grâce aux blocs modulables que nous avons créés au préalable. illustration ci-dessous :



cette méthode nous permet de

### Jupyter book

Lien du Jupyter book :

# 

# Résultats

### 

## Algo 1

### Tableaux de résultats

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Graphiques

### Incertitudes / Commentaires

Blablabla

## Algo 2

### Tableaux de résultats

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Graphiques

### Incertitudes / Commentaires

Blablabla

# Discussions

Discuter les résultats, expliquer quel est l’algorithme le plus précis, pourquoi a notre avis, discuter en quoi les incertitudes ont pu fausser certains calculs par rapport à d'autres algo, pourquoi on a décidé de tester ces algo, ignorer les autres, qu’est-ce qu’on peut dire des graphiques, et tout le tralala qui va avec.

# Conclusion

Dire ce qu’on a pu observer, quel est l’algo et les calculs qui nous semble le plus pratique à utiliser, ce qu’on a pu tirer de ce projet, tout ce qu’on a appris, ce qui nous a plus, choqué, impressionné, etc...