RAPPORT IA

1 Intro

1.1 organisation du groupe

Gitlab , tutos, calls etc

1.2 choix des librairies tensorflow / Keras etc

Quelle lib et pourquoi

1.3 difficultés rencontrées

1.4 intro IA

Expliquer neurone etc concepts de base

2 Autre modeles IA explorés

2.1 random forest

2.2 etc…

3 le approche que nous avons retenu , CNN.

3.1 dataset at first glance

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.

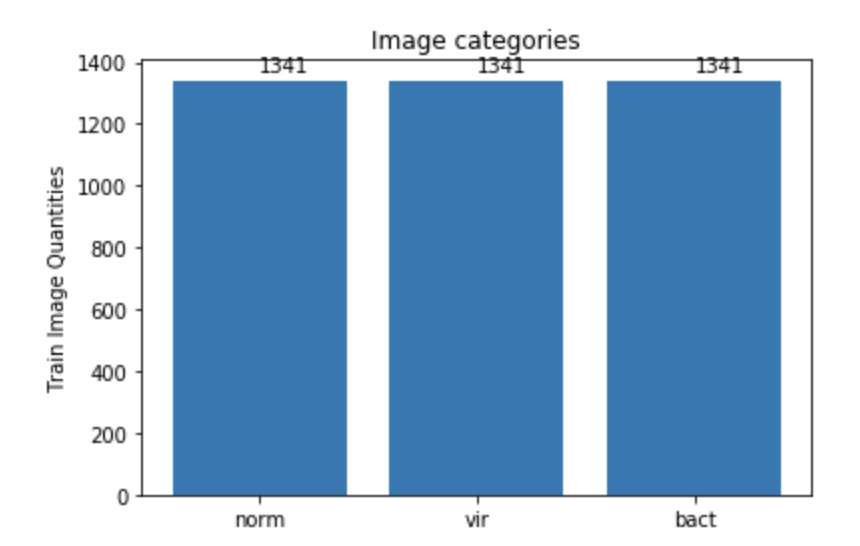
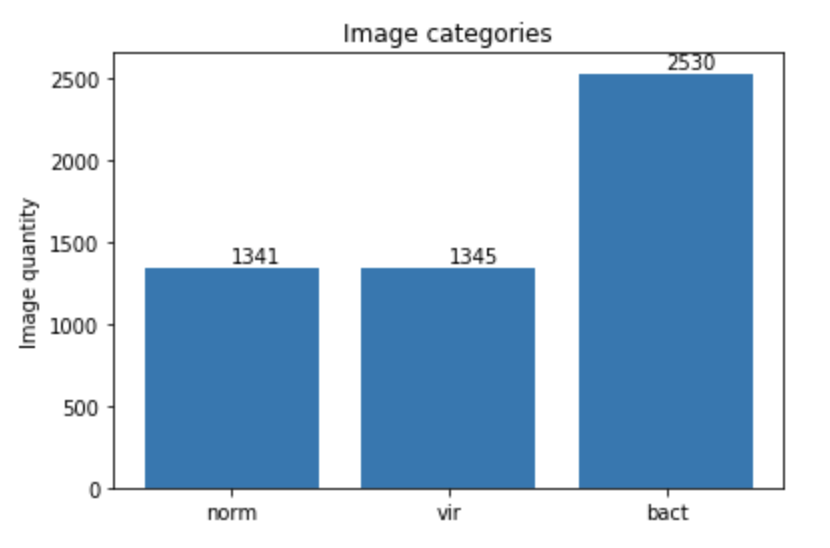
Nous remarquons que les images situées dans les dossiers PNEUMONIA présentent systématiquement le mot « bacteria » ou « virus » dans le nom du fichier. Ceci va nous permettre de distinguer au final 3 targets qui seront Normal, Virus et Bacteria.

3.2 Deeper view of our dataset

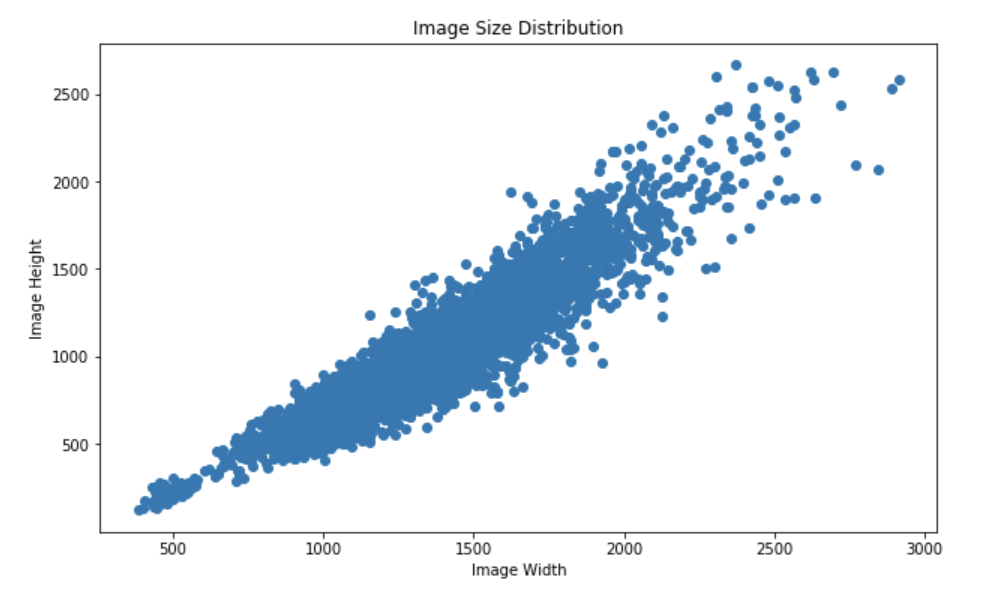
Après avoir récolté nos données, la prochaine étape va consister a nettoyer nos données. Cette phase souvent longue mais indispensable, elle permettra de préparer notre dataset pour un entrainement de notre modèle dans de meilleures conditions.

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 rééquilibrés (graph de droite), le dataset est cependant amputé d’environ 1500 images mais nous y remédieront plus tard grace à la data augmentation.

Pour pourvoir entrainer 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 redimensionnons toutes nos images au format 64x64 pixels. Ce choix fut un bon compromis entre le temps d’entrainement limité par la puissance de calcul de nos machines et le 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 coté.

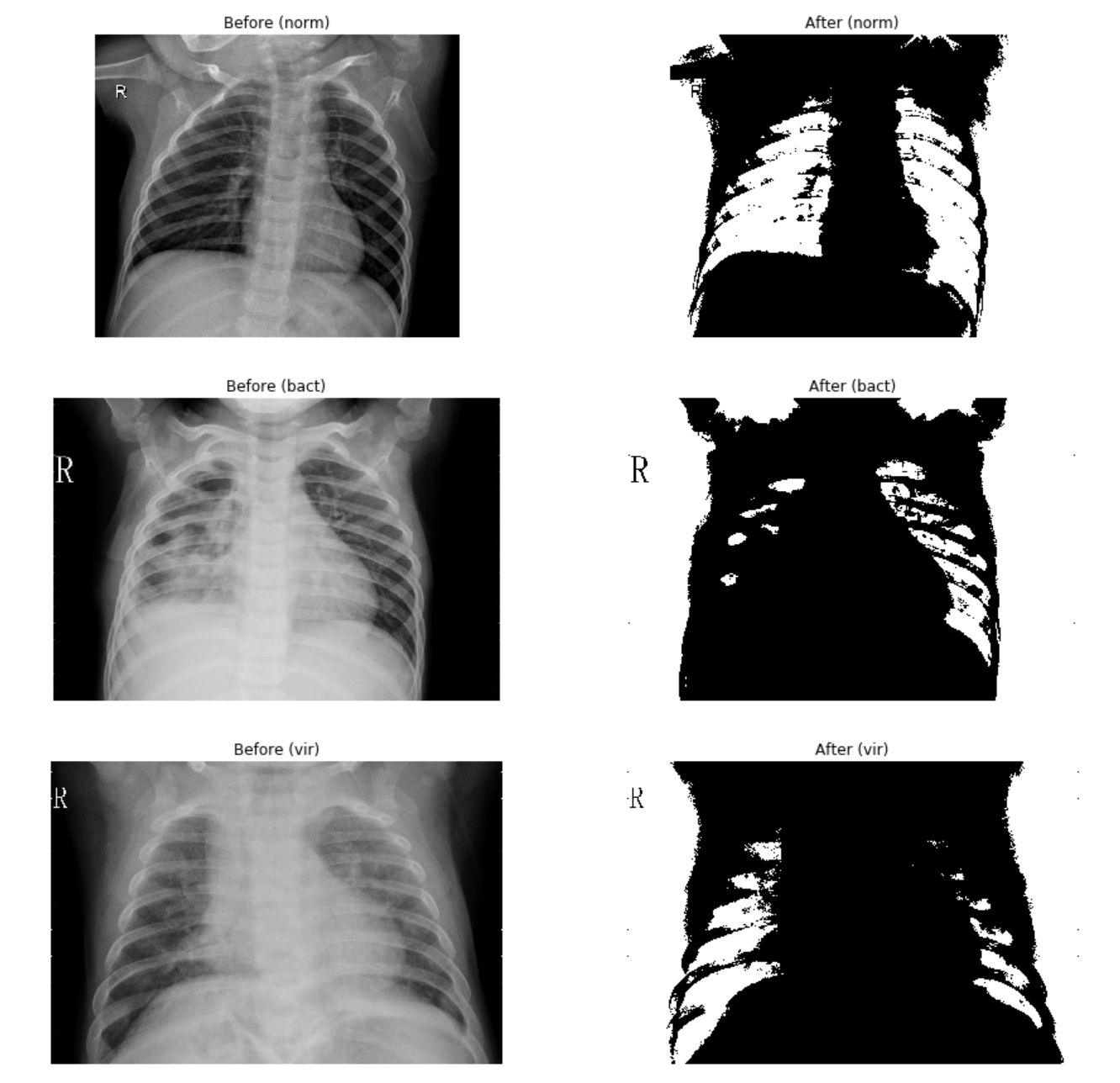
Nos images devront toutes avoir le même nombre de canaux colorimétriques. Nous les convertirons toutes sur 3 canaux, R,G et B. WHY RGB ND NOT GRAYSCALE?

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 conséquence le modèle s’entrainera plus rapidement et pourra potentiellement avoir une meilleure précision.

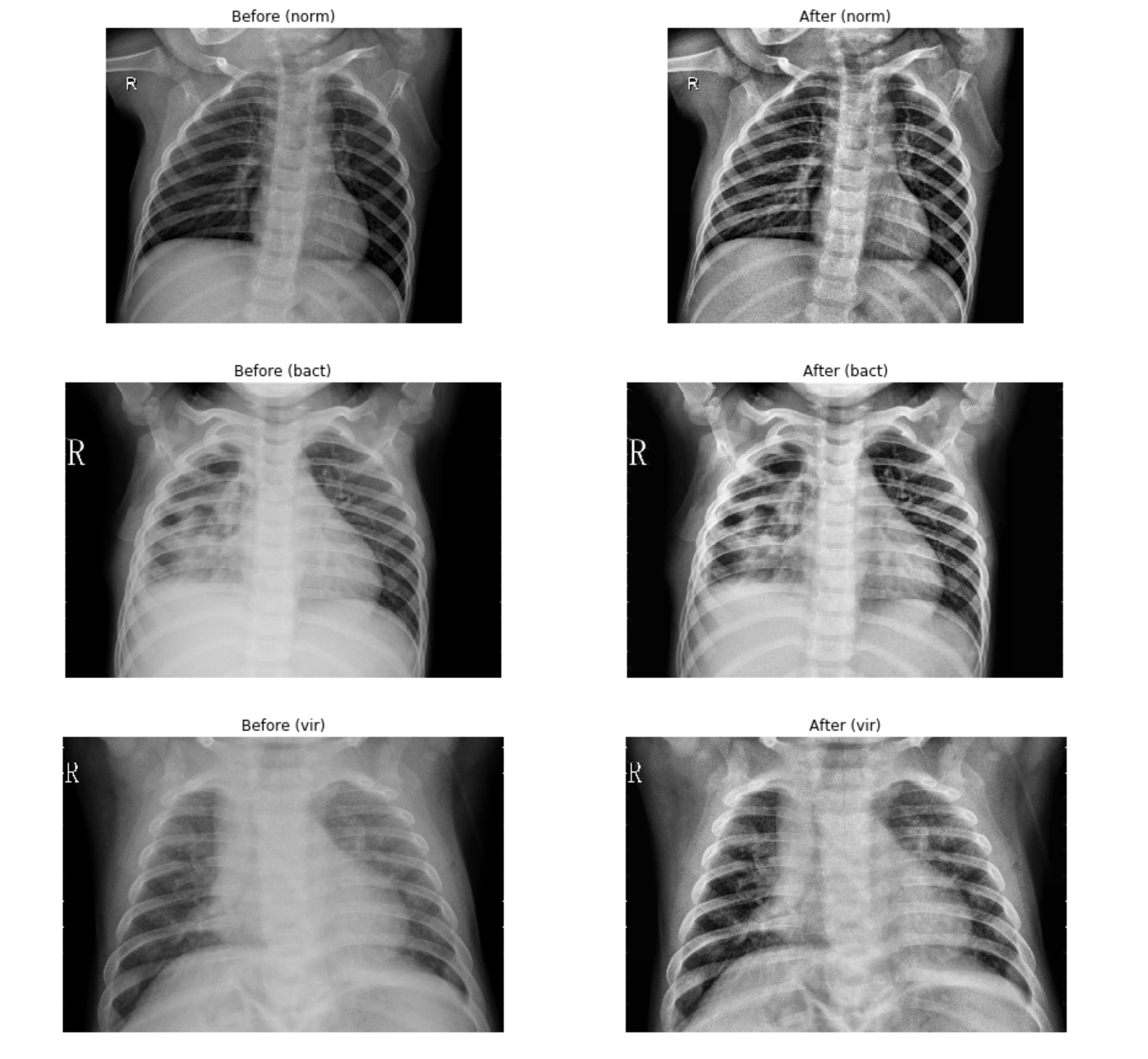
Pour aller un peu plus loin dans la préparation de nos données, nous avons voulu enrichir nos images en leurs appliquant des traitements. Le but étant toujours d’optimiser l’apprentissage de notre modèle.

Pour se faire, avons explorés deux approches:

La première consistait a 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 contentais que de deux targets (normal et pneumonia). En revanche la segmentation sur un modèle à trois targets n’a pas été convaincant. Le modèle ne parvenait pas a différencier les Bacteria des Virus.



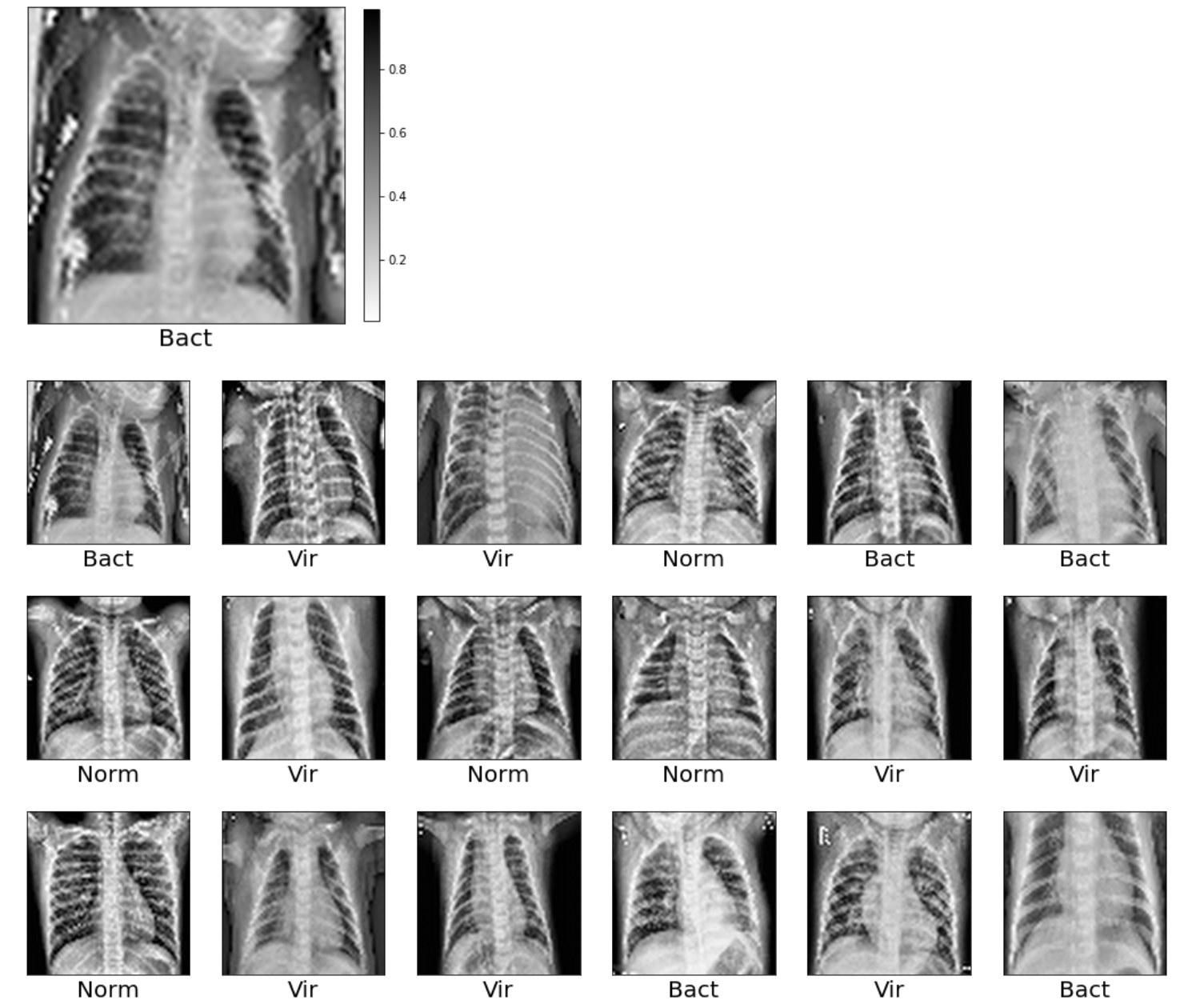
La seconde approche fut celle que nous avons gardé. Celle-ci consistait a optimiser l’équilibre des contrasts afin d’éliminer le bruit et rendre les images plus détaillées. Nous avons utilisé l’algorithme CLAHE (Contrast-Limited Adaptive Histogram Equalization) présente dans la librairie OpenCV. Les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



Une fois tous ces changements apportés a nos données, nous obtenons des images comme celles de l’échantillons ci-dessous.

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

La proportion du dataset entre les dossiers TRAIN TEST et VAL demeurent identique aux proportions de départ, soit : 89% TRAIN, 11% TEST, >1% VAL. Ce split est cohérent pour un bon entrainement de notre modèle.

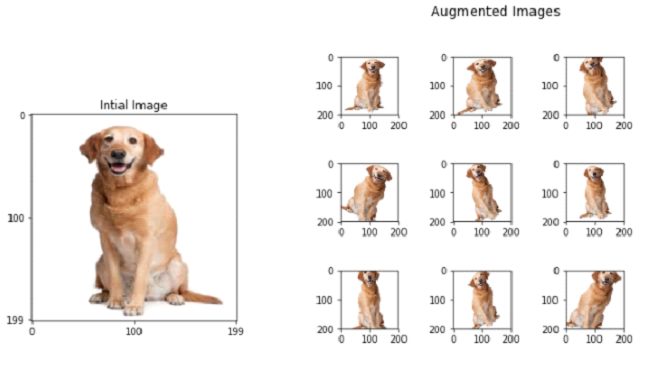


La dernière étape de la préparation de nos données sera de palier au nombre insuffisant d’images présente dans le dataset.

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 entrainer 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 présente dans notre dataset. Les images artificielles présenterons 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:

4 The Model !

4.1 other explored models

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 d’un vecteur unidimensionnel l’image avant de les faire passer dans le réseau et on perdrait toutes les informations bi-dimensionnels.

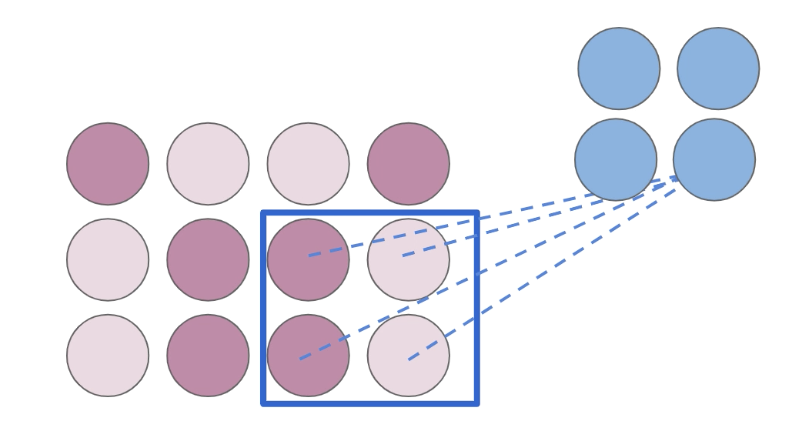
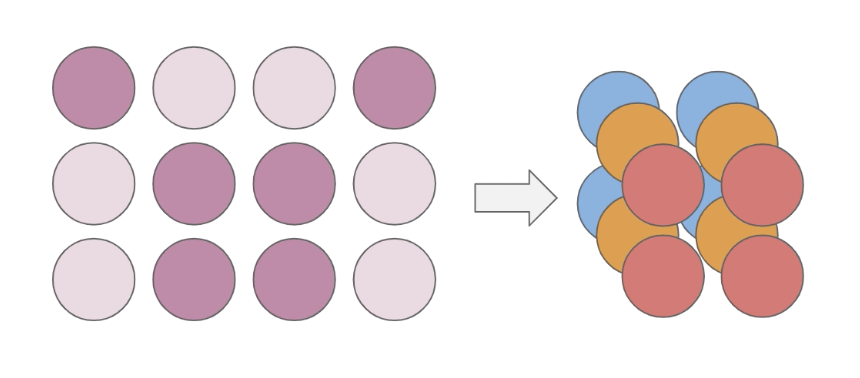
Random forest?

4.2 Type of model we have chosen

Pour ce projet nous avons choisi de faire un réseau neuronal de convolution. Les modèles CNN sont populaires pour faire de la classification ou reconnaissance d’images pour leurs excellente précision. Nous compareront les résultats de notre CNN avec d’autre 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 obtiens une nouvelle image de résolution inférieure.

Une couche de convolution est créée lorsque nous appliquons plusieurs matrices au images d’entrée. Les poids valeurs des poids seront mis a jour par retro-propagation du gradient et la couche sera entrainée 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’elements utils à la résolution de notre problème.

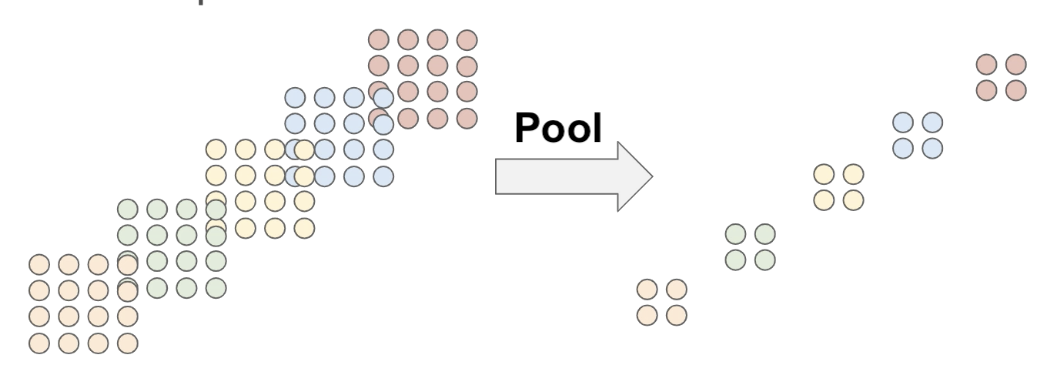
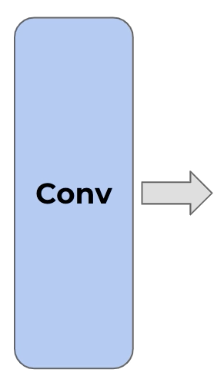
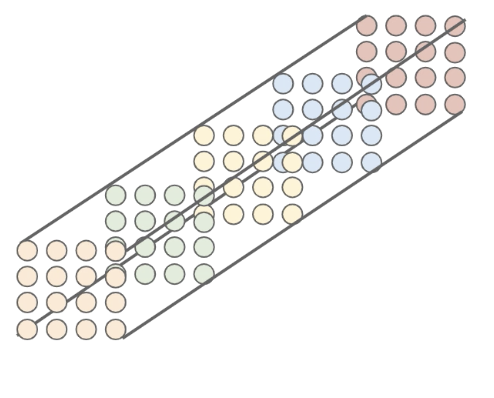
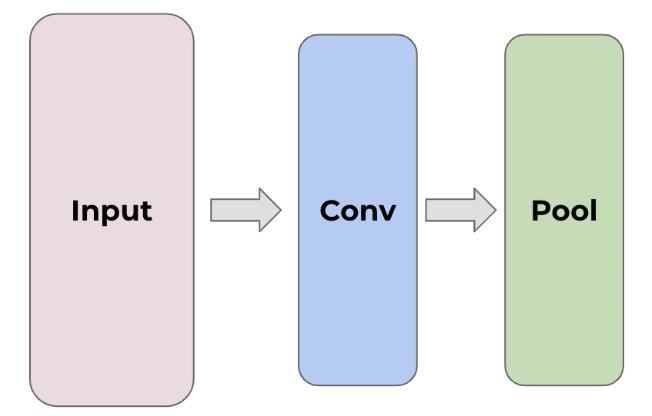
Nos couches de convolution seront aussi dotés d’une fonction d’activation ce qui apportera de la non-linéarité dans notre réseau. Sans fonction d’activation, notre CNN ne serait finalement qu’une regression linéaire. Ici nous avons utilisé la fonction d’activation Relu, populaire dans les CNN.

La couche de convolution va nous permettre de réduire le nombre de paramètres grace 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 réduisent 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.

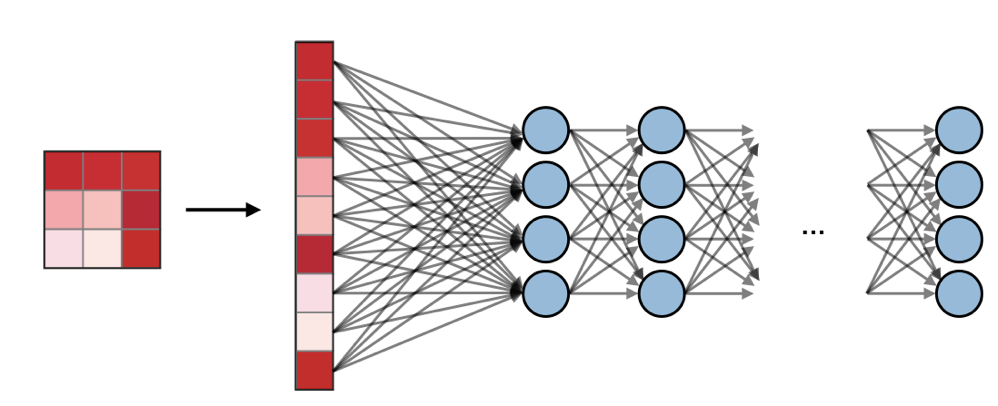
Nous utiliserons une autre technique courante qui s’appelle le dropout. Relativement simple et très efficace pour éviter le sur-apprentissage ou plutôt overfitting. Le principe du dropout est de désactiver aléatoirement un pourcentage de neurones. A chaque epoch, la combinaison de neurones sur lequel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



Notre CNN devra ensuite procéder a un flatten. Le but étant du 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 accroitre 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 différentes. L’output sera dans notre cas un vecteur a une dimensions au format [ x, y, z]. Chaque index correspond a 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 nécéssaire 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 obtenus de bons résultats avec deux optimiseurs. Adam qui est réputé pour sa rapidité d’entrainement et son efficacité. Puis Nadam, une variante de Adam, qui bénéficie d’un accélérateur de gradient au nom de Nesterov momentum.

4.3 Our model’s architecture

L’architecture que nous avons retenu pour notre modèle se schématise comme ci-dessous:

SCHEMA HERE

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 20%

Un flatten

Une couche dense de 16 neurones

Une couche cachée de 32 neurones

Une couche cachée de 64 neurones

Un dropout de 30%

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. Explain more , overfitting etc variance bias etc.

4.4 cross validation

5 Model Training

5.1 Saving the model

5.2 callbacks

5.3 Hyperparams

5.4 Train

Graphs etc bians variance etc

6 Evaluation

6.1 the metrics

Loss accuracy

6.2 classification report

6.3 confusion matrix

6.4 prediction

6.5 diff models perf comparaison

TensorFlow is an end-to-end python machine learning library for performing high-end numerical computations. TensorFlow can handle deep neural networks for image recognition, handwritten digit classification, recurrent neural networks, NLP (Natural Language

Processing), word embedding and PDE (Partial Differential Equation). TensorFlow Python ensures excellent architecture support to allow easy computation deployments across a wide range of platforms, including desktops, servers, and mobile devices.

Abstraction is the major benefit of TensorFlow Python towards machine learning and AI projects. This feature allows the developers to focus on comprehensive logic of the app instead of dealing with the mundane details of implementing algorithms. With this library, python developers can now effortlessly leverage AI and ML to create unique responsive applications, which responds to user inputs like facial or voice expression.

MNIST Handwritten Digits Classification using a Convolutional Neural Network (CNN)

DL : Deep learning is more complex and is typically used for projects that involve classifying images, identifying objects in images, and enhancing images and signals. In these instances, a deep neural network can be applied, as they are designed to automatically extract features from spatially- and temporally-organized data such as images and signals. Common algorithms used in deep learning include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and reinforcement learning (deep Q networks).

ML : Machine learning algorithms may be more desirable if you need quicker results. They are faster to train and require less computational power. The number of features and observations will be the key factors that affect training time. Engineers applying machine learning should expect to spend a majority of their time developing and evaluating features to improve model accuracy.

Deep learning models will take time to train. Pretrained networks and public datasets can shorten training through transfer learning, but sometimes these can be complicated to implement. In general, deep learning algorithms can take anywhere from a minute to a few weeks to train depending on your hardware and computing power. Engineers applying deep learning should expect to spend a majority of their time training models and making modifications to the architecture of their deep neural network.

ML supervisé

Preparer la data

Entrainer sur la data

Tester un modele sur la data

Evaluer modele

Iterer. (Backward forward)

LIVRABLES : 2 docs .

1 doc avec les résultats visuels algos params etc. (Jupiter notebook)

jupiter notebook pr aspet technique

1 rapport pr expliquer la démarche adoptee les choix etc

Explorer PCA , algos de reshape

COMMUNIQUER SUR LE FORUM ! C est une compétence notée!

Where to start :

Suport vecteur machine

Arbre de decisions

Réseau neurones

Sci-kitlearn has Jupiter examples

A RENDRE PR LE 9 MAI

Autos Thibault neveu <https://www.youtube.com/c/ThibaultNeveu/videos>

Deep learning 2 au 12 descente de gradient puis les vids : Comment classifier des images av tensorflow PUIS : réseau a convolution

Clues

<https://converteo.com/blog/kaggle-detection-de-la-pneumonie-dans-les-radiographies/>

@@@ IA

Inputs / features : a node takes a input and gives a feature as response. Feature ex : pixel value? Or feature = input?

**Target :** final output you are trying to predict, also know as y. It can be categorical (sick vs non-sick) or continuous (price of a house).

**Label :** true outcome of the target. In supervised learning the target labels are known for the trainining dataset but not for the test.

Label is more common within classification problems than within regression ones. Nonetheless, they are often used interchangeably without great precision.

Vecteur = suite de nombres

Neurone : en deux parties, préactivation et activation. Activation = sigmoide(réactivation)

Sigmoide : give a probability as neurone output

Weights: corresponds to the importance we give to a certain feature/input

Stride : the step a kernel will move

LOGISTIC REGRESSION :

Binary classification : output is 1 < x < 0. Wich can be seen as a probability

NEURONE :

Constitué de inputs (x1 x2 etc..) , de variables internet (w1 w2 bias), une préactivation, une activation, et une probabilité en sortie. C’est une somme ponderee avec un bias, le tout passé dans une fonction d’activation.

Préactivation : pour chaque input on le multiplie avec son weight , formule math qui prends les features, weights et bias , et output une seule valeur

Activation : formule math qui prend la val du préactivation et donne une nouvelle valeur sous forme de proba

Erreur = abs(val obtenu a la sortie du neurone - val attendue). Ou sinon RMSE(root mean square error) ou MSE(mean square error) funcs

DESCENTE DE GRADIANT : pour minimiser la fonction d’erreur et trouver le point le pkus bas. permet de réduire l’erreur en modifiant les weights dans l expectative de rentre la valeur de sortie du neurons aussi proche que possible de la valeur attendue.

Steps : define inputs/features (clean dataset)

Define target

Find the correct balance between **bias** and **variance BIAS TRADOFF IS THE POINT WHEN THE MODEL STOPS UNDERFITTING AND STARTS OVERFITTING. HIGH VARIANCE MAKES OVERFITTING**

bias is the amount that a model’s prediction differs from the target value, compared to the training data

variance describes how much a random variable differs from its expected value. Variance is based on a single training set. Variance measures the inconsistency of different predictions using different training sets

OVERFITTING : low train loss, high val loss. Le modele s adapte trop au bruit des données

UNDERFITTING :

—> pourquoi pas utiliser un réseau ANN : car on se retrouverai avec un nombre énorme de paramètres et donc un temps d apprentissage bcp trop long.

—> la convo va appliquer une série de filtres (kernels) a l image pour faire ressortir certains elements.

—> le pooling

—> dropout , désactive aléatoirement des neurones pour prévenir de l’overfitting

—> ont-not encoding : donne un résultat de sortie de type [0,0,0,1,0,0]

—> early stopping/callback will halt when model doesn’t improve learning. Patience : if very small batches or large learning rate : then big patience. Else small patience. Usually set patience between 10 and 20

—> validation\_loss

—> to prevent overfitting : data augmentation for big dataset, dropouts, less complexe model archi, earlystopping

—> conv2D : starting with filters in the range *[32, 64, 128]* in the earlier and increasing up to *[256, 512, 1024]* in the deeper layers.

—> Kernells : Typical values for kernel\_size include: (1, 1) , (3, 3) , (5, 5) , (7, 7). If so, consider using a *5×5* or *7×7* kernel to learn larger features and then quickly reduce spatial dimensions — then start working with *3×3* kernels. If your images are smaller than *128×128* you may want to consider sticking with strictly *1×1* and *3×3* filters.