Le Deep Learning appliqué au diagnostic de pneumonies

Théo Gachet

39910

Session 2021-2022

Sommaire I

Sommaire II

Introduction : contexte, motivations et objectifs

Contexte et motivations

- thème "Santé et Prévention"
- contexte sanitaire
- contexte technologique
- études supérieures

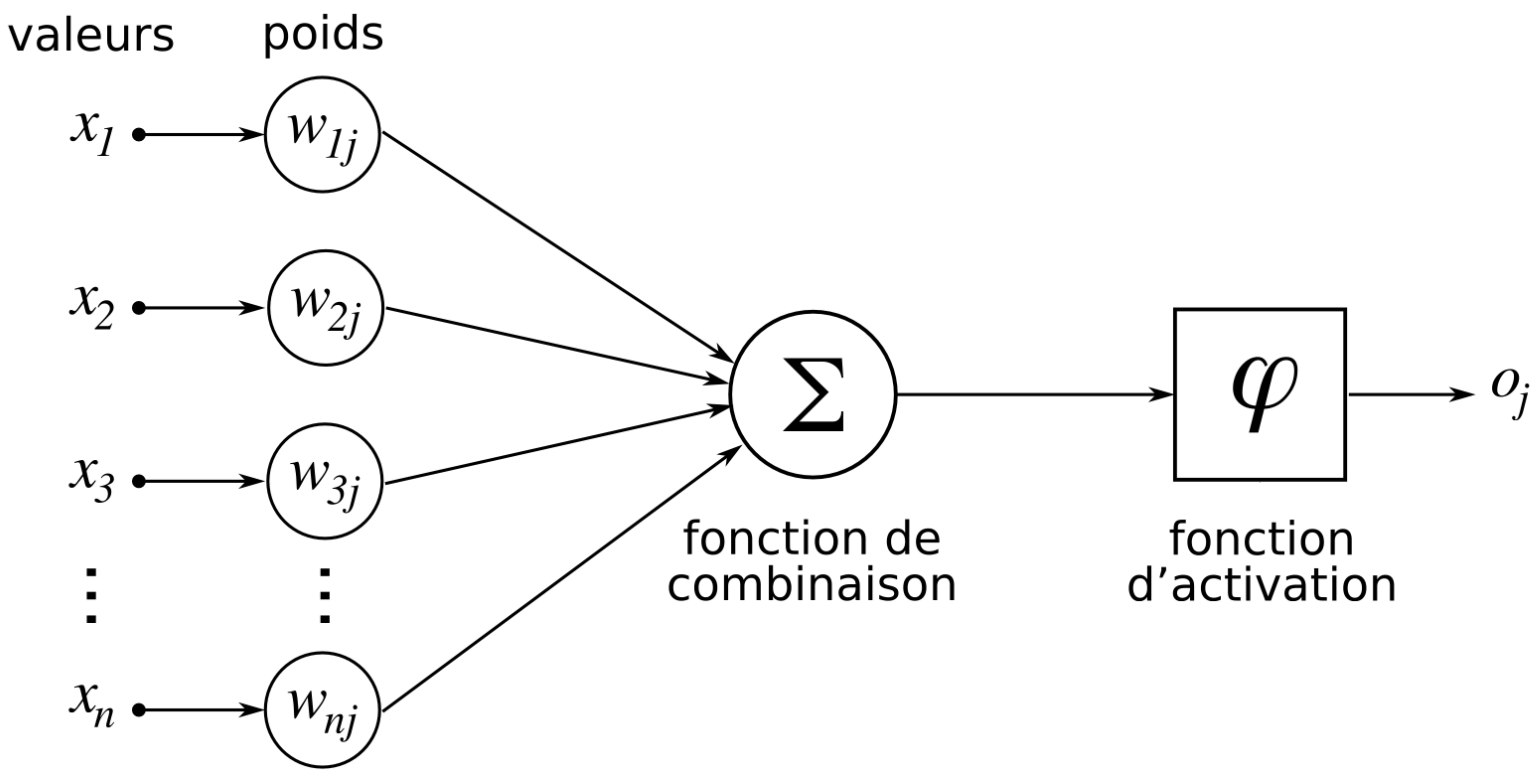
Objectifs du projet

- solution à un problème de minimisation
- classement et indexation de données médicales
- traitement et exploitation des données
- construction d'un réseau de neurones convolutifs
- entraînement du réseau à la reconnaissance de pneumonies

Qu'est-ce qu'un réseau de neurones ?

Combinaison linéaire pondérée par des coefficients synaptiques

$$o_j = \phi\left(\sum_{i=1}^{n}(w_i.x_i)\right)$$

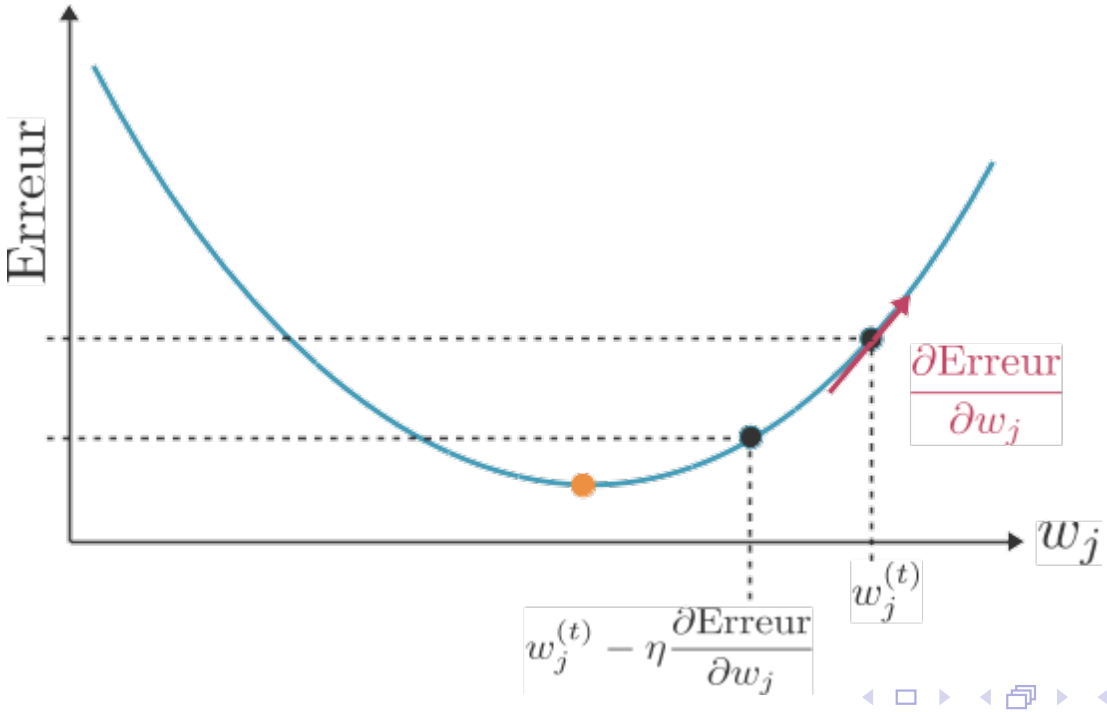


Optimiseur Adam (adaptative moment estimation)

Fonction d'entropie binaire croisée

$$\text{Erreur} = -\frac{1}{n}\sum_{i=1}^{n}(t_i \cdot \log(y_i) + (1 - t_i) \cdot \log(1 - y_i))$$

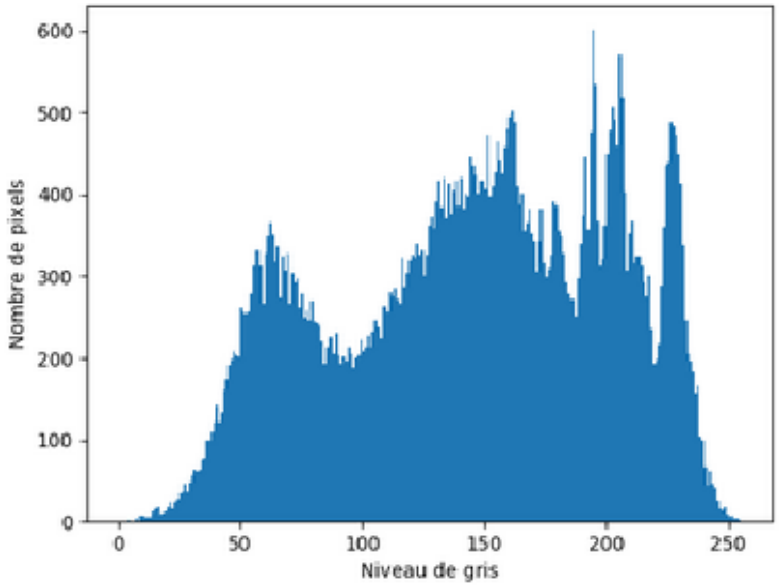
où y_i est la i-ème valeur scalaire de sortie du modèle, t_i est la valeur cible correspondante et n est le nombre de valeurs scalaires à la sortie du modèle



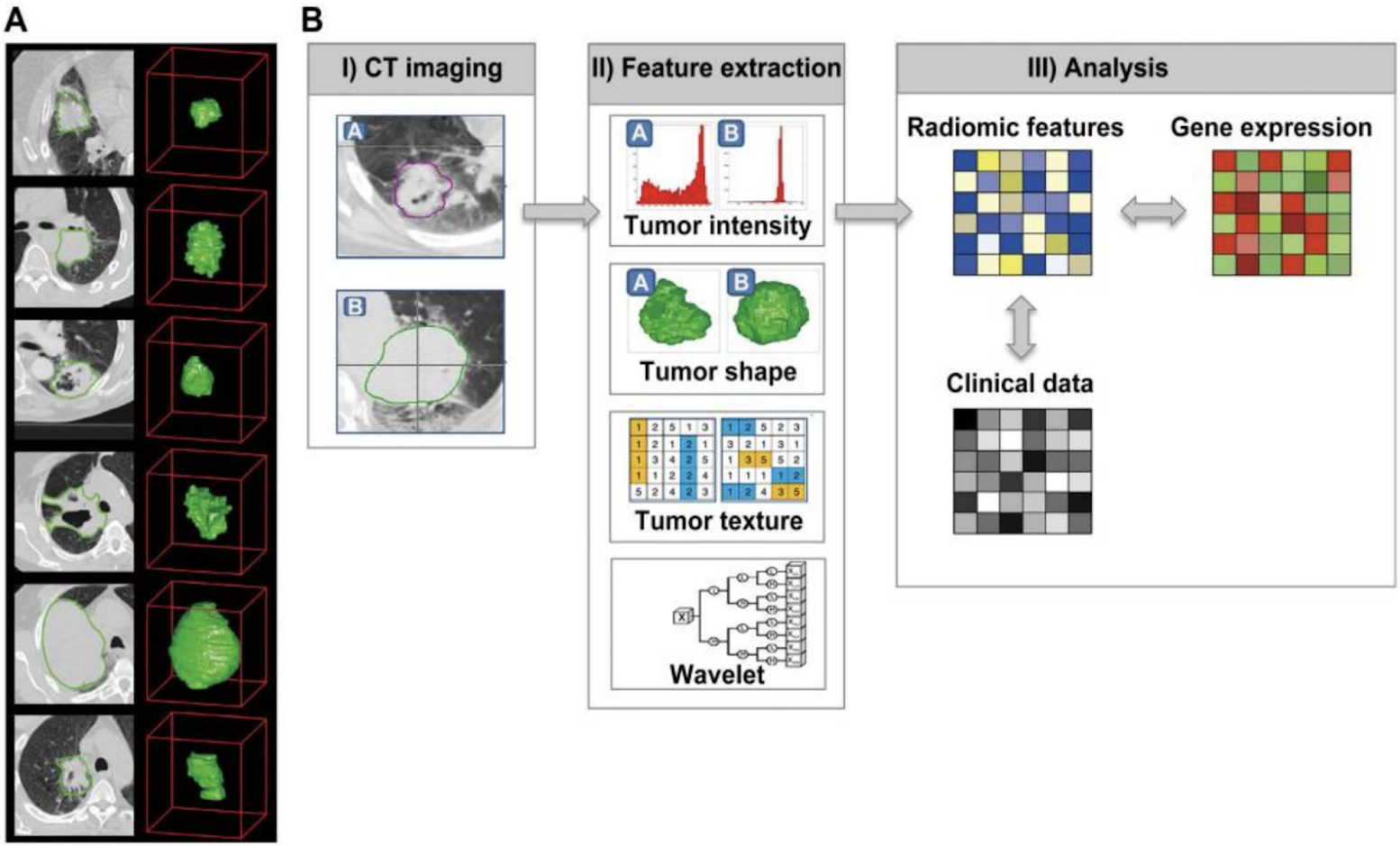
D'une image à une fonction mathématique

A chaque image on peut associer une fonction intensité :

$$I : \left| \begin{array}{ccc} \mathbb{R} & \longrightarrow & \mathbb{R} \\ (x, y) & \longmapsto & I(x, y) \end{array} \right.$$



Extraire l'information d'une image



exemple des features d'une radiographie de tumeur

Données d'entraînement, d'évaluation et de test

On divise les ensembles de données en six sous-ensembles

```
for _set in ['train', 'val', 'test']:
  n_normal = len(os.listdir(input_path + _set + '/NORMAL'))
  n_infect = len(os.listdir(input_path + _set + '/PNEUMONIA'))
```

Ensemble	**Scans sains**	**Pneumonies**
train	1341	3875
val	8	8
test	234	390

Ensembles de données

Constantes de référence : dimension, batch, epochs

Dimension des images, taille du batch et nombre d'epochs

```
img_dims = 150
batch_size = 32
epochs = 10
```

Relation liant les trois constantes de référence

$$\frac{d}{b} = \frac{i}{e}$$

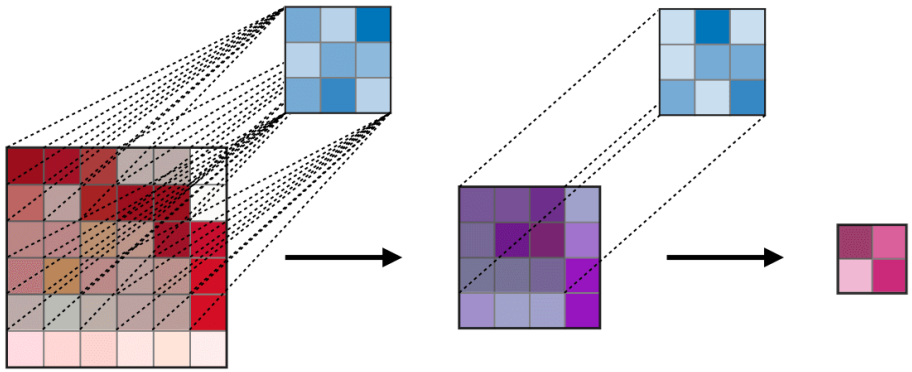
$$\text{avec} \begin{cases} \text{d} = \text{taille du dataset} & \text{e} = \text{nombre d'epochs} \\ \text{i} = \text{nombre d'itérations} & \text{b} = \text{taille du batch} \end{cases}$$

Couche de convolution (CONV), hyperparamètres

Couche de convolution (CONV)

La couche de convolution est le bloc de construction de base d'un réseau de neurones convolutifs. Pour dimensionner le volume de sortie (volume de la couche de convolution), on définit 3 hyperparamètres : la profondeur, le pas et la marge, pour obtenir une feature map, ou activation map.

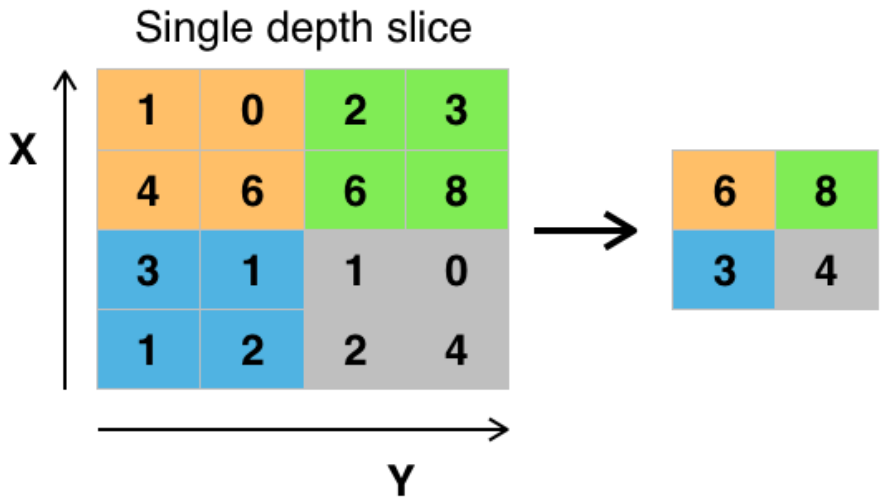
```
x = Conv2D(filters=16,kernel_size=(3,3),activation='relu',padding='same')(inputs)
x = MaxPool2D(pool_size=(2, 2))(x)
```



Couche de pooling (POOL)

Le pooling ("mise en commun") est un sous-échantillonnage de l'image. On divise l'image d'entrée en une série de cellules de n pixels de côté ne se chevauchant pas. Le signal en sortie de cellule est défini en fonction des différents pixels de la cellule. Cela permet de diminuer la quantité de paramètres et le nombre de calculs dans le réseau.

```
x = MaxPool2D(pool_size=(2, 2))(x)
(compression d'un facteur 2*2 = 4 avec le max_pooling, ou l'average_pooling)
```



Couche de correction (ReLU)

Pour améliorer les performances du réseau, on fait appel à une fonction d'activation à la sortie de chaque couche.

Fonctions d'activation usuelles

- tangente hyperbolique :

$$f(x) = tanh(x)$$

- fonction sigmoïde :

$$f(x) = (1 + e^{-x})^{-1}$$

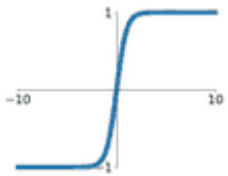
- tangente hyperbolique saturante :

$$f(x) = |tanh(x)|$$

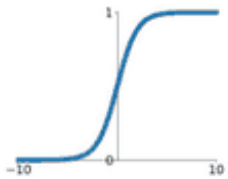
- Unité Linéaire Rectifiée (ReLU) :

$$f(x) = max(0, x)$$

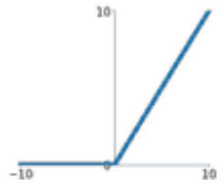
tanh
$\tanh(x)$



Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

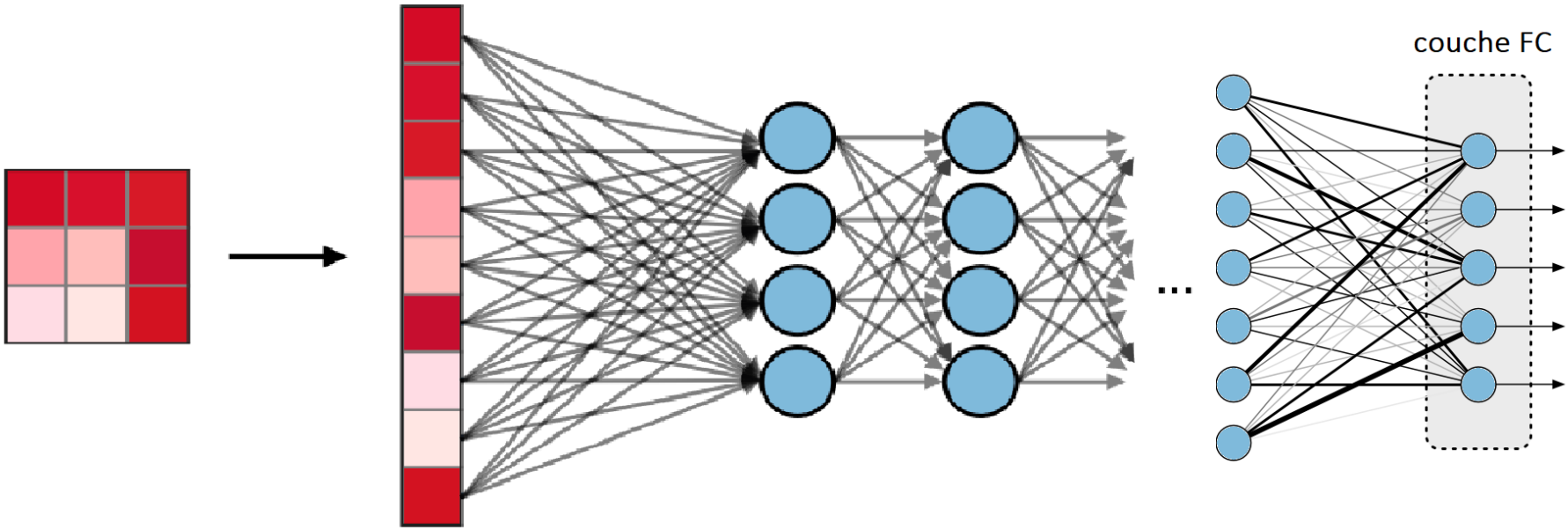


ReLU
$\max(0, x)$



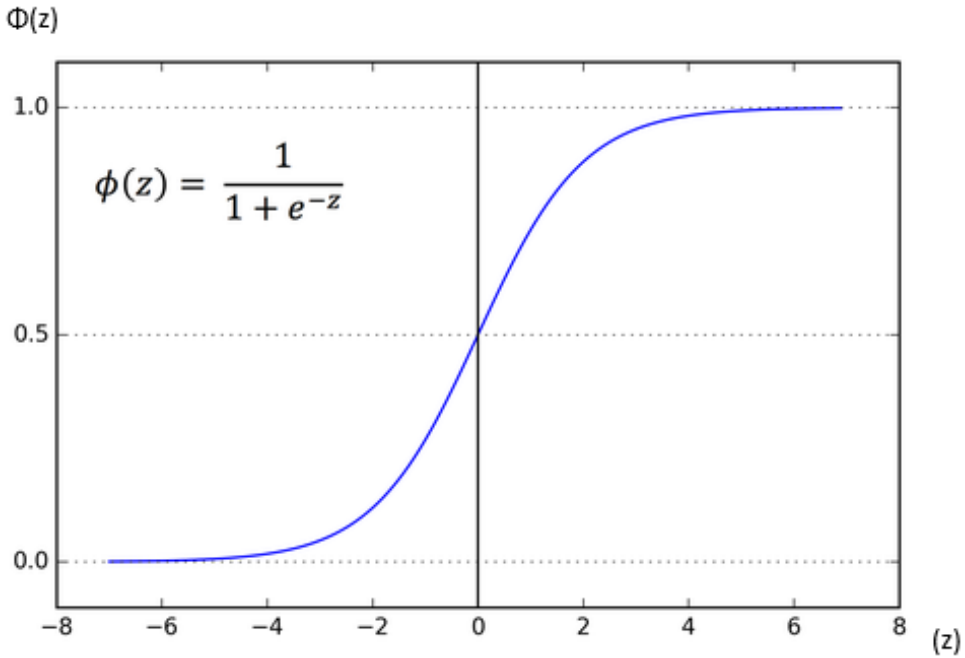
Couche entièrement connectée (FC)

A la fin de chaque couche, la couche "entièrement connectée" connecte chaque neurone d'une couche à ceux de la suivante. Elle reçoit un vecteur et produit un nouveau vecteur en sortie qui est obtenu en appliquant une combinaison linéaire aux valeurs du vecteur d'entrée. Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe.



Fonction sigmoïde

Jusqu'à présent, j'ai utilisé la fonction ReLU. Pour la dernière couche du réseau, j'ai exceptionnellement appliqué la fonction sigmoïde parce que, à la fin, notre problème est fondamentalement binaire. En effet, la fonction sigmoïde donne une valeur entre 0 et 1, une probabilité. Elle est donc très utilisée pour les classification binaire lorsqu'un modèle doit déterminer seulement deux labels, ce qui est notre cas.



Couche de perte (LOSS)

La couche de perte (LOSS) précise la différence entre le résultat attendu et le résultat obtenu. Ainsi, elle est généralement placée à la fin du réseau. Ici, on utilise la fonction d'entropie binaire croisée ("binary crossentropy") avec la fonction sigmoïde en entrée d'activation. L'entropie binaire croisée est une fonction de perte qui permet de traiter dessituations binaires, comme notre cas de diagnostic médical.

Fonction d'entropie binaire croisée

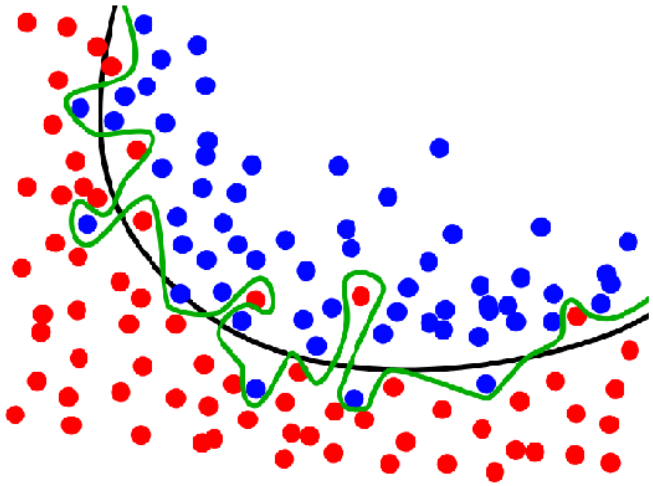
$$\text{Erreur} = -\frac{1}{n}\sum_{i=1}^{n}(t_i \cdot \log(y_i) + (1 - t_i) \cdot \log(1 - y_i))$$

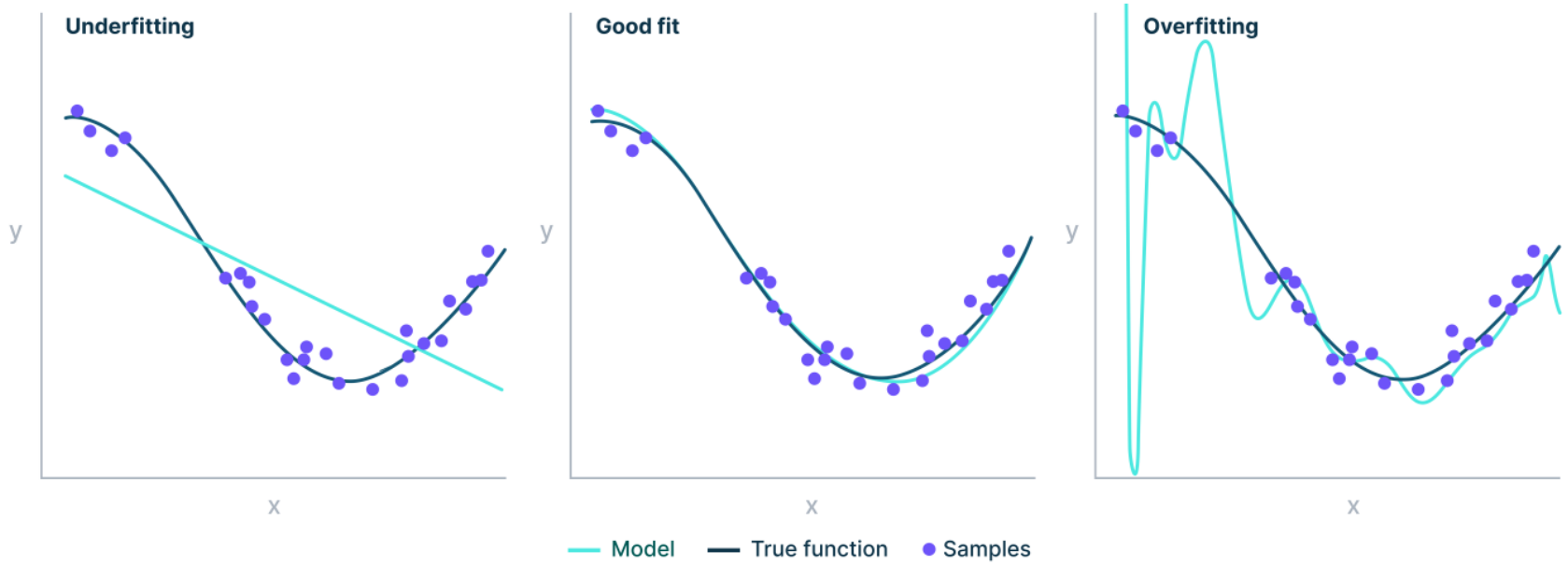
où y_i est la i-ème valeur scalaire de sortie du modèle, t_i est la valeur cible correspondante et n est le nombre de valeurs scalaires à la sortie du modèle

Prétraitement : overfitting et data augmentation

Fonction de preprocessing

```
def process_data():
    rescale=1./255
    zoom_range=0.3
    vertical_flip=True
```

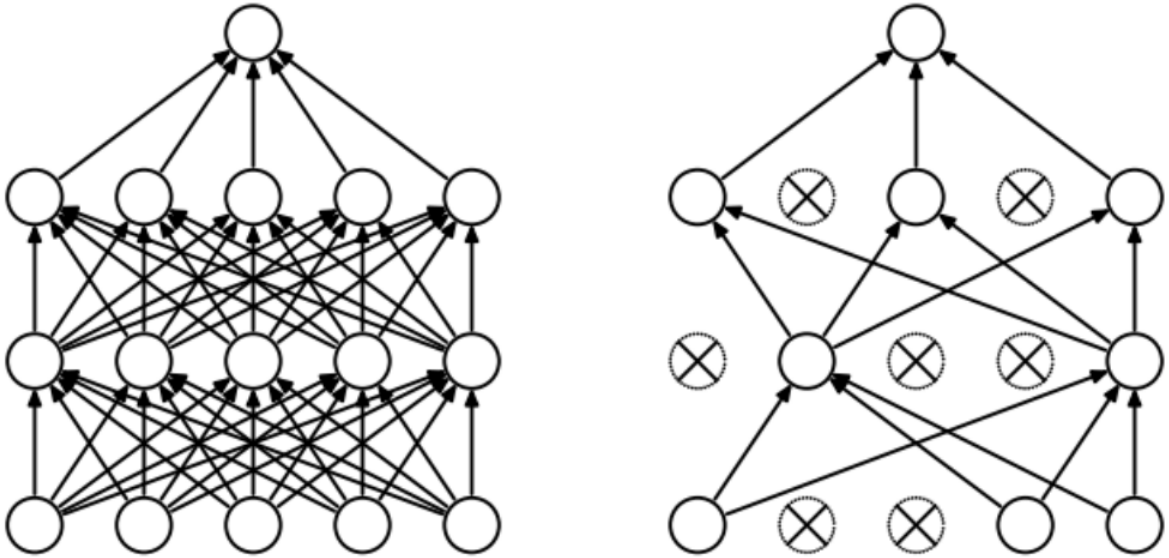




Méthode du décrochage (dropout) contre l'overfitting

Le dropout réduit l'overfitting, c'est un moyen très efficace d'exécuter un moyennage du modèle de calcul avec des réseaux de neurones. Le terme "décrochage" se réfère à une suppression de neurones.

Le réseau neuronal se voit amputé d'une partie de ses neurones pendant la phase d'entrainement (leur valeur est estimée à 0) et ils sont par contre réactivés pour tester le nouveau modèle.



Ajouts pratiques : fonctions de rappel (callback)

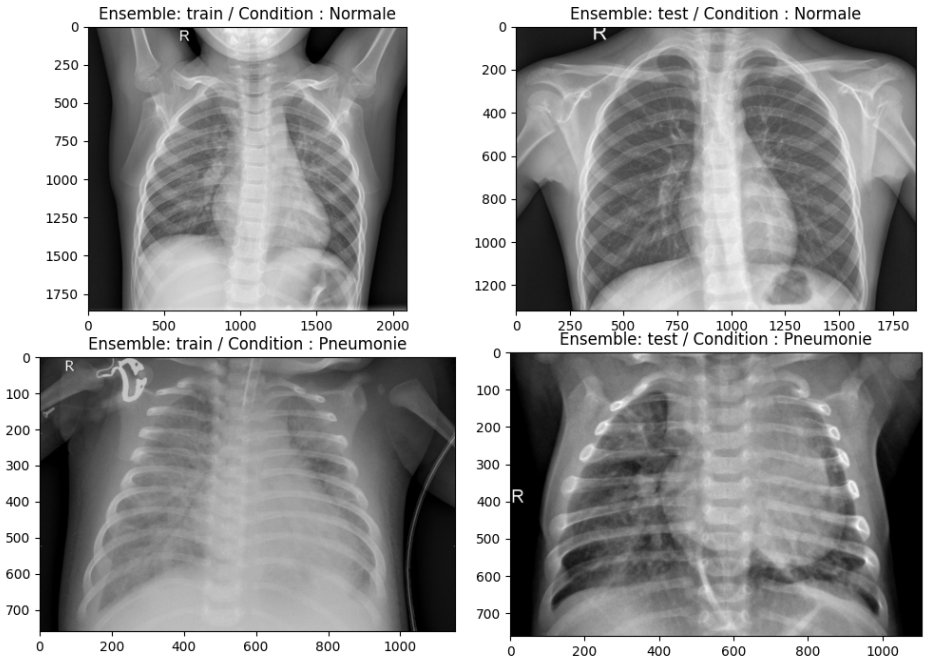
On définit les fonctions de rappel :

```
checkpoint = ModelCheckpoint(filepath='best_weights.hdf5',
    save_best_only=True, save_weights_only=True)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3,
    patience=2, verbose=2, mode='max')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1,
    patience=1, mode='min')
```

- ModelCheckpoint : l'entraînement du réseau nécessite l'exécution avec plusieurs epochs. Pendant l'exécution du programme, ModelCheckpoint sauvegarde, à la fin de chaque itération, la meilleure performance du modèle.
- EarlyStopping : utile pour stopper le processus de généralisation lorsque la différence entre les résultats d'entraînement et les données de validation commence à trop grandir, i.e. en cas d'overfitting.

Entraînement de l'algorithme

epoch(s)	test (%)	train (%)
1	62,5	82,5
3	79.3	90,9
5	86.2	91.8
7	72.0	94.1
10	84,6	95.1



Interprétation des résultats

```
21  seed = 232
22  np.random.seed(seed)
23  tf.random.set_seed(seed)
```

epoch(s)	test (%)	train (%)
1	62,5	82,5
3	79.3	90,9
5	86.2	91.8
7	72.0	94.1
10	84,6	95.1

Le Deep Learning appliqué au diagnostic de pneumonies

Théo Gachet

39910

Session 2021-2022

Merci de votre attention

Annexe : Algorithme de rétropropagation du gradient

Fonctionnement de l'algorithme de gradient

- Soient un échantillon $\vec{x}$ que l'on présente à l'entrée du réseau de neurones et $\vec{t}$ la sortie recherchée pour cet échantillon.
- On propage le signal en avant dans les couches : $x_k^{(n-1)} \mapsto x_j^{(n)}$, avec n le numéro de la couche, et k et j les numéros des neurones sur leur couche respective.
- La propagation vers l'avant se calcule à l'aide de la fonction d'activation g, de la fonction d'agrégation h (produit scalaire entre les poids et les entrées du neurone) et des poids synaptiques $\vec{w_{jk}}$ entre le neurone $x_k^{(n-1)}$ et le neurone $x_j^{(n)}$:

$$x_j^{(n)} = g^{(n)}(h_j^{(n)}) = g^{(n)}\left(\sum_k w_{jk}^{(n)} x_k^{(n-1)}\right)$$

Annexe : Algorithme de rétropropagation du gradient

- Lorsque la propagation vers l'avant est finie, on obtient $\vec{y}$ à la sortie.
- On calcule alors l'erreur entre la sortie donnée par le réseau $\vec{y}$ et le vecteur $\vec{t}$ désiré à la sortie pour cet échantillon. Pour chaque neurone i dans la couche de sortie, on calcule :

$$e_i^{sortie} = g'(h_i^{sortie}(y_i - t_i))$$

- On propage l'erreur vers l'arrière $e_i^{(n)} \mapsto e_j^{(n-1)}$:

$$e_j^{(n-1)} = g'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij}^{(n)} e_i^{(n)}$$

- On met à jour les poids dans toutes les couches :

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \lambda e_i^{(l)} x_j^{(l-1)}$$

où λ représente le taux d'apprentissage compris entre 0 et 1.

Annexe : Démonstration des poids synaptiques

$$E = \frac{1}{2} \sum_i (y_i - t_i)^2$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial h} \frac{\partial h}{\partial w}$$

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_i (y_i - t_i) g'^{(n)}(h_i^{(n)}) \sum_k w_{ik}^{(n)} \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}}$$

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{(n)} \underbrace{g'^{(n)}(h_i^{(n)})(y_i - t_i)}_{e_i^{(n)}}$$

Annexe : Démonstration des poids synaptiques

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{(n)} e_i^{(n)}$$

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(n-2)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{(n-1)} e_i^{(n-1)}$$

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(l)}}{\partial w_{ab}^{(l)}} \sum_i w_{ik}^{(l+1)} e_i^{(l+1)} = x_b^{(l-1)} e_a^{(l)}$$

Annexe : Code

```python
# I ------------------------------------------------------------------

import os               # 1
import numpy as np
import pandas as pd
import random
import cv2
import matplotlib.pyplot as plt

import keras.backend as K                          # 2
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, SeparableConv2D, MaxPool2D, LeakyReLU, Activation
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
import tensorflow as tf

input_path = 'F:/- PNEUMONIA/chest_xray/'

seed = 232                  # 3
np.random.seed(seed)
tf.random.set_seed(seed)

# II -----------------------------------------------------------------

for _set in ['train', 'val', 'test']:                              # 4
    n_normal = len(os.listdir(input_path + _set + '/NORMAL'))
    n_infect = len(os.listdir(input_path + _set + '/PNEUMONIA'))
    print('Ensemble:{}/Scans sains:{}/Scans de pneumonie:{}'.format(_set, n_normal,n_infect))
```

Annexe : Code

```
25  # II ------------------------------------------------------------------
26
27  for _set in ['train', 'val', 'test']:                                  # 4
28      n_normal = len(os.listdir(input_path + _set + '/NORMAL'))
29      n_infect = len(os.listdir(input_path + _set + '/PNEUMONIA'))
30      print('Ensemble:{}/Scans sains:{}/Scans de pneumonie:{}'.format(_set, n_normal,n_infect))
31
32
33  # III -----------------------------------------------------------------
34
35  def process_data(img_dims, batch_size):
36
37      train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, vertical_flip=True)      # 5
38      test_val_datagen = ImageDataGenerator(rescale=1./255)
39
40      train_gen = train_datagen.flow_from_directory(                                              # 6
41          directory=input_path+'train',
42          target_size=(img_dims, img_dims),
43          batch_size=batch_size,
44          class_mode='binary',
45          shuffle=True)
46
47      test_gen = test_val_datagen.flow_from_directory(
48          directory=input_path+'test',
49          target_size=(img_dims, img_dims),
50          batch_size=batch_size,
51          class_mode='binary',
52          shuffle=True)
53
```

Annexe : Code

```
56      test_data = []
57      test_labels = []
58
59      for cond in ['/NORMAL/', '/PNEUMONIA/']:
60          for img in (os.listdir(input_path + 'test' + cond)):
61              img = plt.imread(input_path+'test'+cond+img)
62              img = cv2.resize(img, (img_dims, img_dims))
63              img = np.dstack([img, img, img])
64              img = img.astype('float32') / 255
65              if cond=='/NORMAL/':
66                  label = 0
67              elif cond=='/PNEUMONIA/':
68                  label = 1
69              test_data.append(img)
70              test_labels.append(label)
71
72      test_data = np.array(test_data)
73      test_labels = np.array(test_labels)
74
75      return train_gen, test_gen, test_data, test_labels
76
77  # IV ------------------------------------------------------------------
78
79  img_dims = 150                                                    # 8
80  epochs = 5
81  batch_size = 32
82
83  train_gen, test_gen, test_data, test_labels = process_data(img_dims, batch_size)     # 9
84
```

Annexe : Code

```
85  # V ------------------------------------------------------------------
86
87  inputs = Input(shape=(img_dims, img_dims, 3))
88
89  # 1er bloc convolutif
90  x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
91  x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
92  x = MaxPool2D(pool_size=(2, 2))(x)
93
94  # 2ème bloc convolutif
95  x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
96  x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
97  x = BatchNormalization()(x)
98  x = MaxPool2D(pool_size=(2, 2))(x)
99
100 # 3ème bloc convolutif
101 x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
102 x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
103 x = BatchNormalization()(x)
104 x = MaxPool2D(pool_size=(2, 2))(x)
105
106 # 4ème bloc convolutif
107 x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
108 x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
109 x = BatchNormalization()(x)
110 x = MaxPool2D(pool_size=(2, 2))(x)
111 x = Dropout(rate=0.2)(x) # 12 : dropout
112
113 # 5ème bloc convolutif
114 x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
115 x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
116 x = BatchNormalization()(x)
117 x = MaxPool2D(pool_size=(2, 2))(x)
118 x = Dropout(rate=0.2)(x) # 12 : dropout
```

Annexe : Code

```
120  # Couche "entièrement connectée"                                          # 11
121  x = Flatten()(x)
122  x = Dense(units=512, activation='relu')(x)
123  x = Dropout(rate=0.7)(x) # 12 : dropout
124  x = Dense(units=128, activation='relu')(x)
125  x = Dropout(rate=0.5)(x)
126  x = Dense(units=64, activation='relu')(x)
127  x = Dropout(rate=0.3)(x)
128
129  # Couche de sortie
130  output = Dense(units=1, activation='sigmoid')(x)                          # 13
131
132  # Création du modèle et compilation
133  model = Model(inputs=inputs, outputs=output)
134  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])          # 14 / 15
135
136  # Fonctions de rappel
137  checkpoint = ModelCheckpoint(filepath='best_weights.hdf5', save_best_only=True, save_weights_only=True)   # 16
138  lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=2, verbose=2, mode='max')
139  early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=1, mode='min')
140
141  # VI ------------------------------------------------------------------
142
143  hist = model.fit_generator(
144              train_gen, steps_per_epoch=train_gen.samples // batch_size,
145              epochs=epochs, validation_data=test_gen,
146              validation_steps=test_gen.samples // batch_size, callbacks=[checkpoint, lr_reduce])
147
```