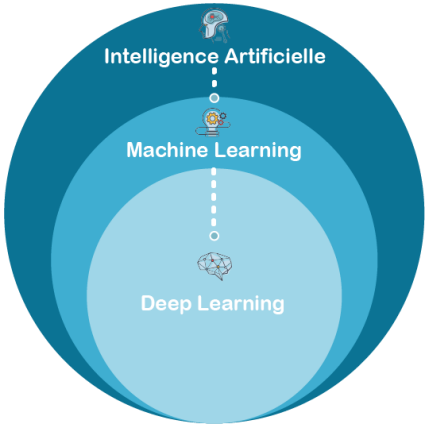
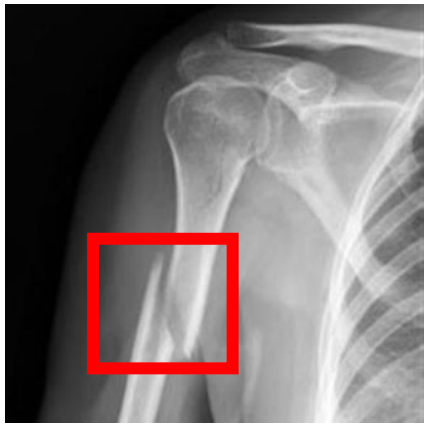
N° candidat : 13910

Automatisation du diagnostic de fractures osseuses par apprentissage profond

Santé, prévention





MELLAH Adib

PLAN

1. Objectif
2. Apprentissage profond
3. Réseau neuronal convolutif
4. Recherche du meilleur algorithme
5. Conclusion
6. Annexes

Objectif

Radiographie par rayons X

Source : https://www.docteurclic.com/maladie/fracture-du-bras.aspx

Algorithme (« modèle ») de **classification binaire** entraîné par **apprentissage profond** sur python

Diagnostic utile à la décision du médecin

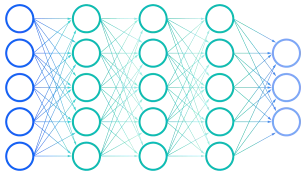
Conversion de l'image en tableau de pixels sous une forme adapté

Détection d'une anomalie (fracture) par l'algorithme

Apprentissage profond

Principe général de fonctionnement

Définition d'un réseau de neurones



Base de données

Apprentissage

Modèle entraîné

Nouvelle donnée

Prédiction

Apprentissage profond

Structure d'un modèle entièrement connecté

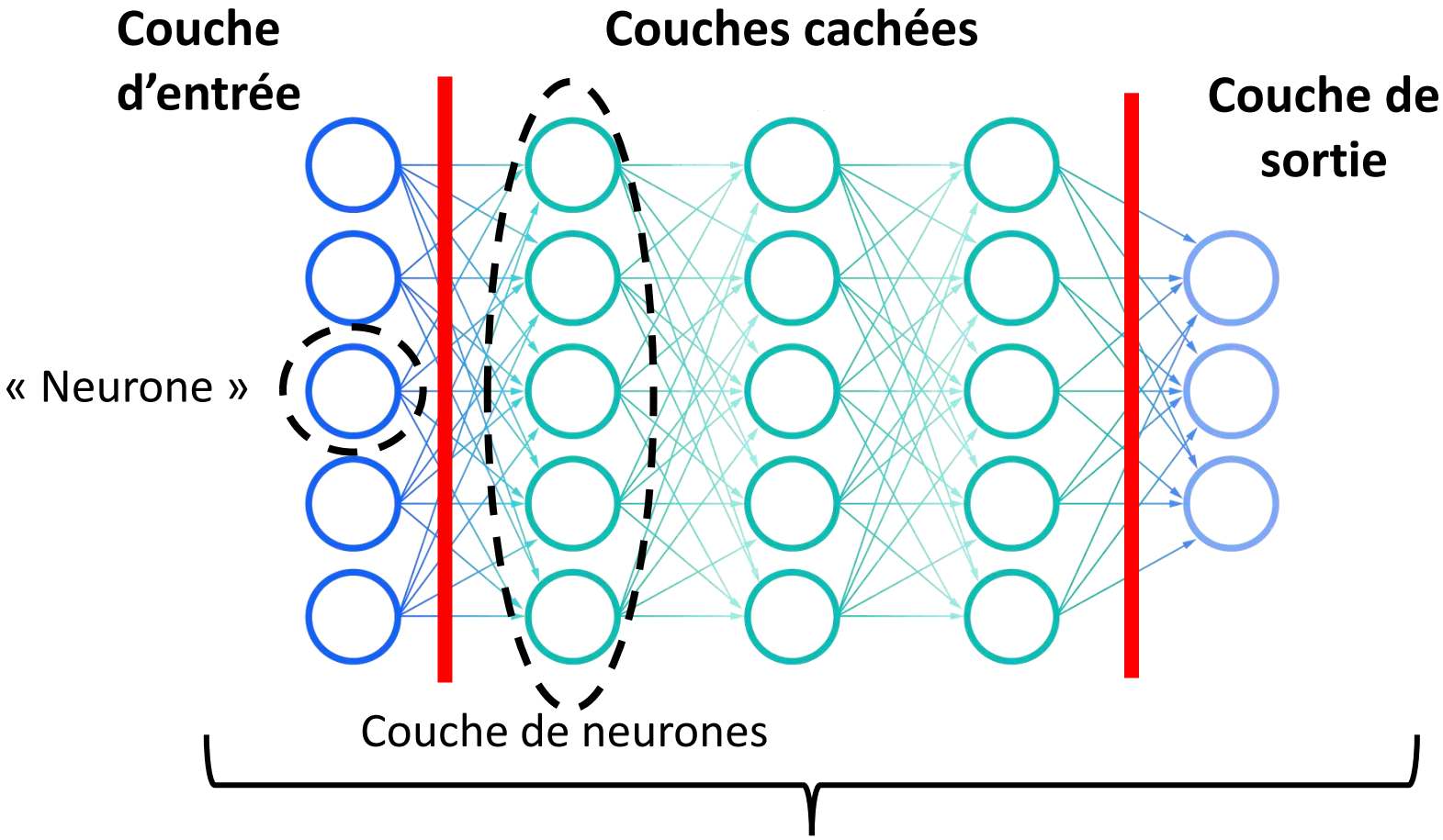
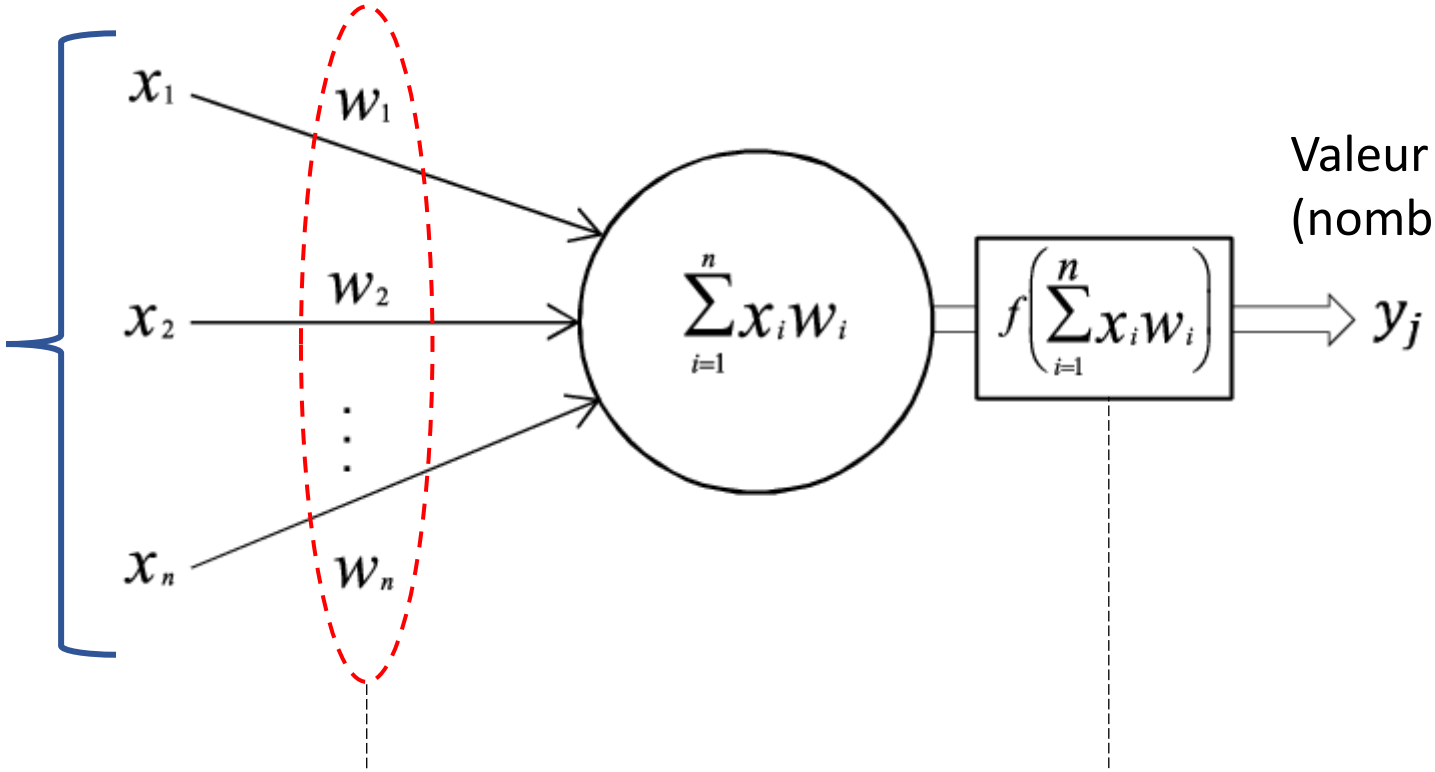


Image : https://1.cms.s81c.com/sites/default/files/2021-01-06/ICLH_Diagram_Batch_01_03-DeepNeuralNetwork-WHITEBG.png

Apprentissage profond

Détail d'un neurone



Valeurs d'entrée (nombres réels)

Valeur de sortie (nombre réel)

Poids

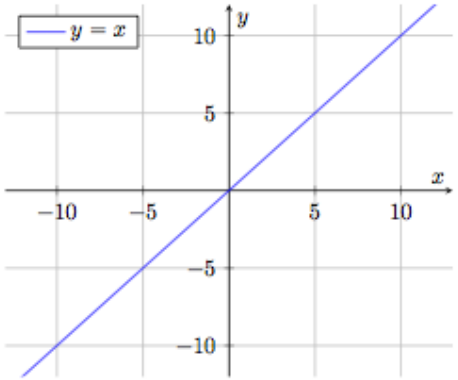
Seuls paramètres variables une fois le modèle défini

Fonction d'activation

Apprentissage profond

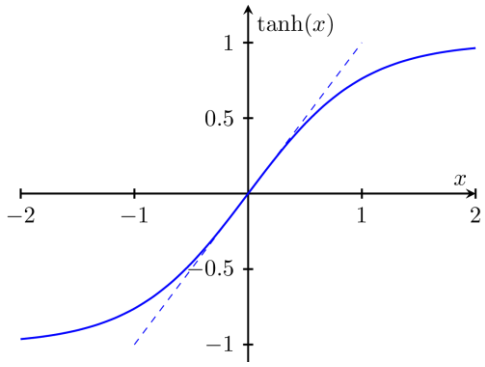
Fonctions d'activation

Identité



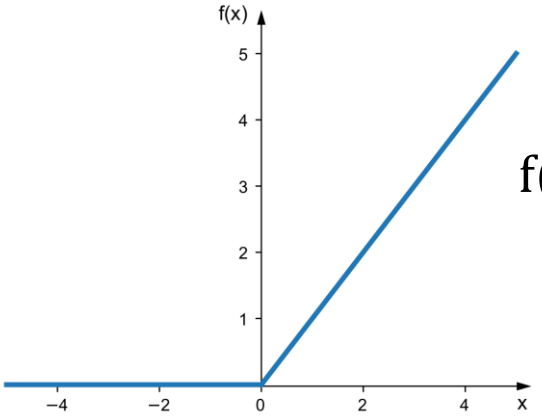
$y = x$

Tangente hyperbolique



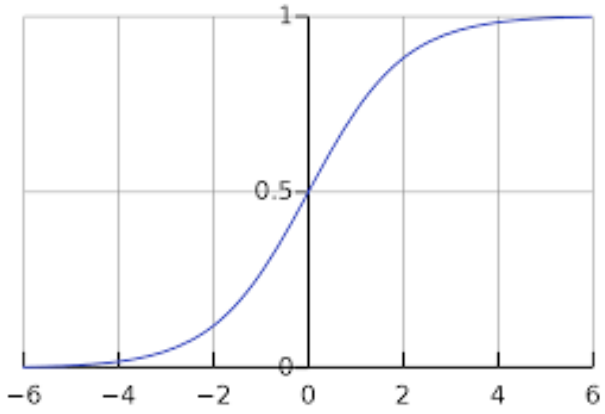
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU (Unité de rectification linéaire)



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

Sigmoïde



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

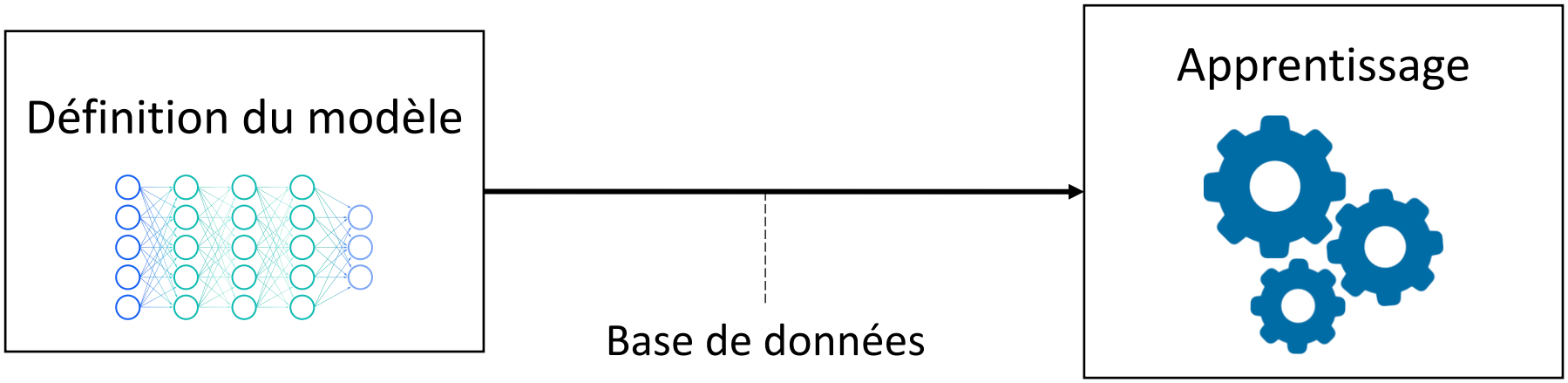
Apprentissage profond
Compilation du modèle

Dernière étape de définition d'un modèle :
configurer sa **manière d'apprendre**

➔ Fonction de perte : évalue l'erreur du modèle par rapport aux données d'entraînements

➔ *Optimizer* : modifie les attributs du modèle en cherchant à minimiser la fonction de perte

Apprentissage profond
Entraînement du modèle

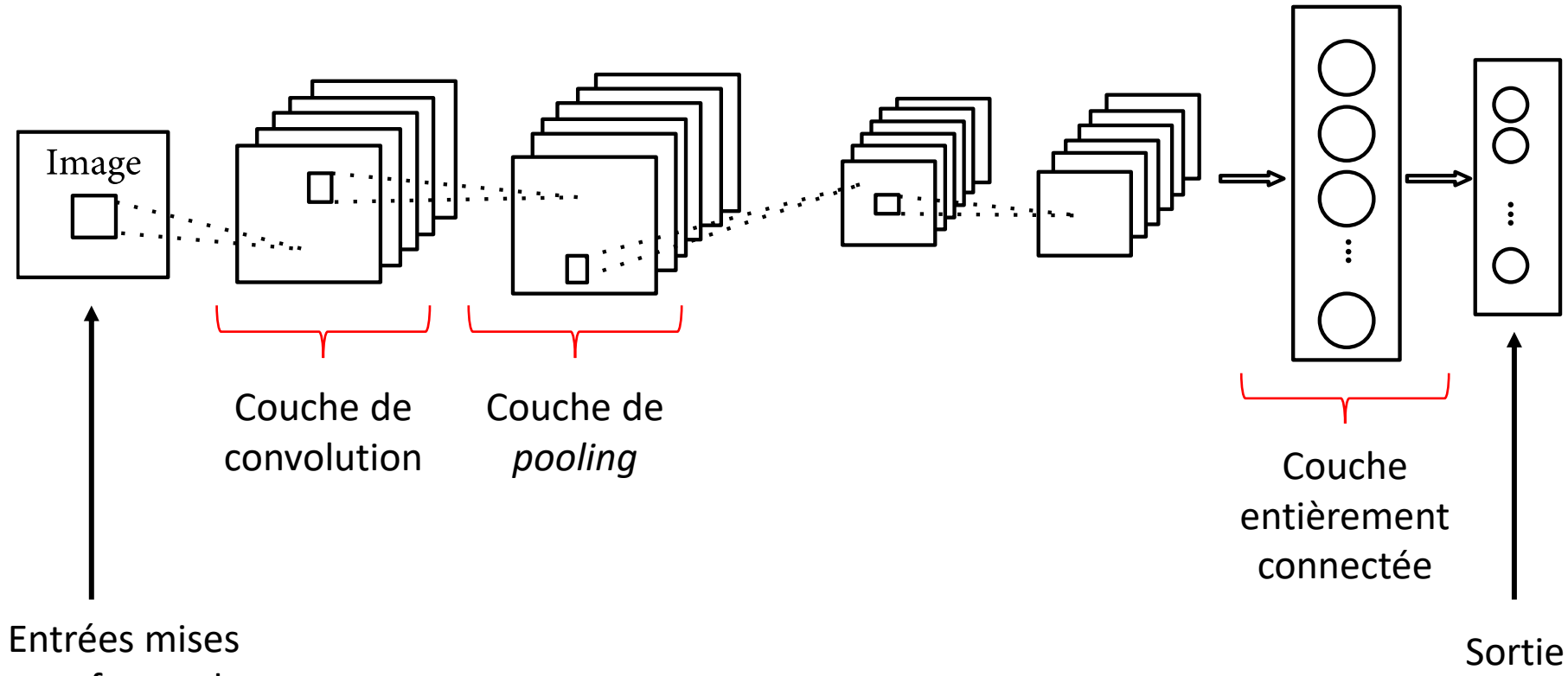


« Apprendre » pour un réseau neuronal :
ajuster les poids associés à chaque neurone de sorte à réduire l'erreur

Paramètres d'apprentissage :
- *Batch_size* : nombre de données traitées avant mise à jour des poids
- *Epochs* : nombre de passage du réseau sur l'entièreté des données
- Part d'aléatoire

Réseau neuronal convolutif (CNN)

Ajout de nouvelles couches en amont de la structure précédente
-> Les entrées sont scindés en morceaux au préalable



Source : https://static-01.hindawi.com/articles/js/volume-2019/6134610/figures/6134610.fig.001.svgz

Réseau neuronal convolutif (CNN)
Choix du type de réseau

Perceptron multicouches (MLP)

- Toutes les couches sont entièrement connectées
- Simplicité théorique
- Nombre de poids très grands

Réseau neuronal convolutif (CNN)

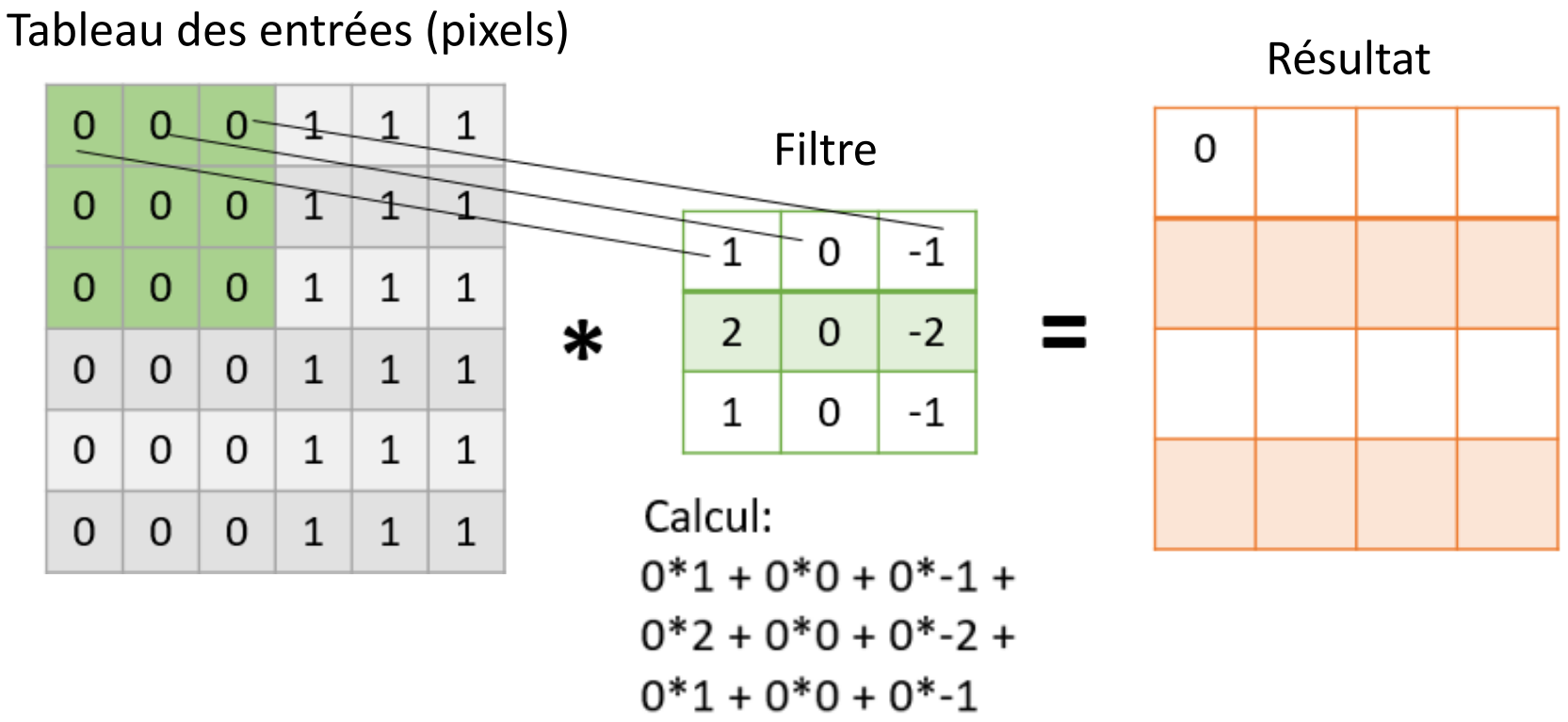
- Couches partiellement connectées
- Invariance de location des entrées
- Compréhension des dépendances spatiales locales
- Plus compliqué à implémenter

➔ CNN plus simple à entraîner et plus adapté au traitement d'images

Réseau neuronal convolutif (CNN)

Couches de convolution

Couche de convolution : assemblage de filtres qui s'appliquent aux entrées

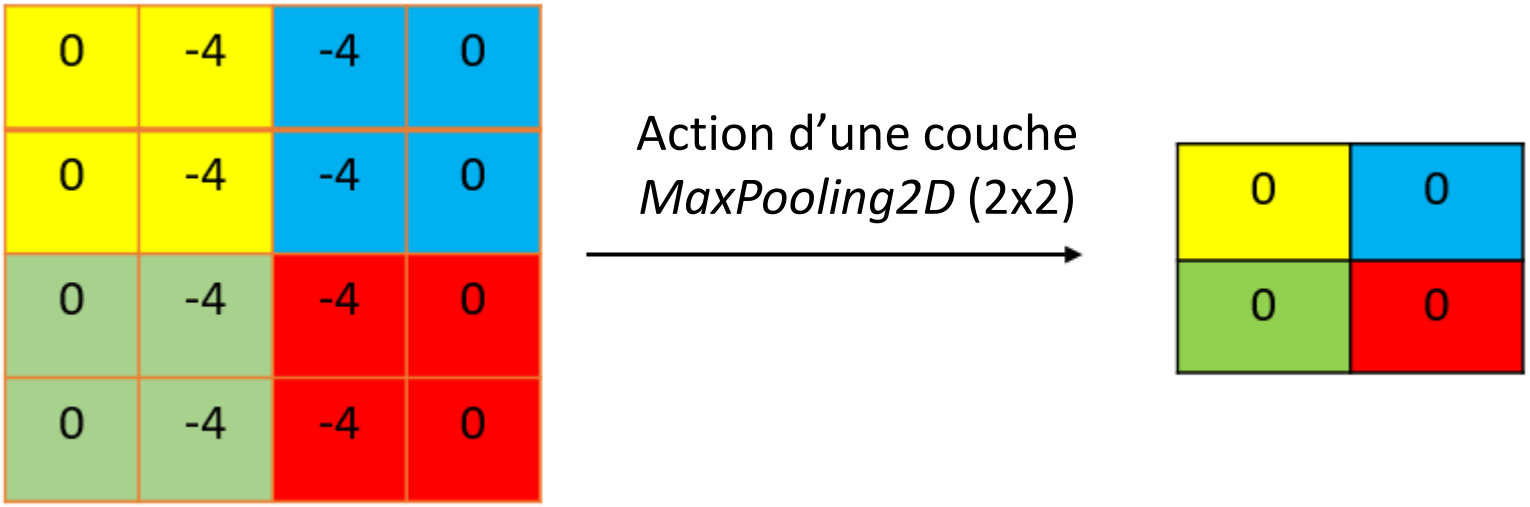


Paramètres spécifiques : nombre de filtres, dimensions des filtres

Réseau neuronal convolutif (CNN)
Couches de pooling

Réduit les dimensions du tableau transmis par la couche de convolution
-> conserve les informations utiles tout en diminuant les coûts en mémoire et en temps

Exemples : prendre le maximum / la moyenne des valeurs dans un sous-tableau



Source : https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/

Recherche du meilleur algorithme

Préparation d'une base de données adéquate

Base de données initiale (images)

Redimensionnement des images pour avoir un format standard

Conversion de l'image en tableau de pixels indiquant le niveau de gris

« Normalisation » des valeurs : nombres réduits dans [0,1]

Extrait du résultat (pour une image)

```
[[[0.22966821]
  [0.22567841]
  [0.22171263]
  ...
  [0.28569692]
  [0.2900497 ]
  [0.29350987]]

 [[0.21627219]
  [0.21395811]
  [0.21174352]
  ...
  [0.28218067]
  [0.2883576 ]
  [0.29015273]]

 [[0.21174352]
  [0.21174352]
  [0.21174352]
  ...
```

Recherche du meilleur algorithme
Prototype de modèle retenu

Certains paramètres sont particulièrement efficaces dans les problèmes de classification binaires

On prendra donc :

- *Binary_crossentropy* comme fonction de perte
- *adam* comme fonction d'optimisation
- *ReLu* pour les fonctions d'activations internes au réseau
- *Sigmoid* pour la fonction d'activation de la couche de sortie (valeur de sortie entre 0 et 1)

Recherche du meilleur algorithme
Essais préliminaires

Taille du lot : 8
Taille des filtres : 5x5
Nombre d'époques : 10

2 couches entièrement connectées
2 {couche de convolution + couche de pooling}

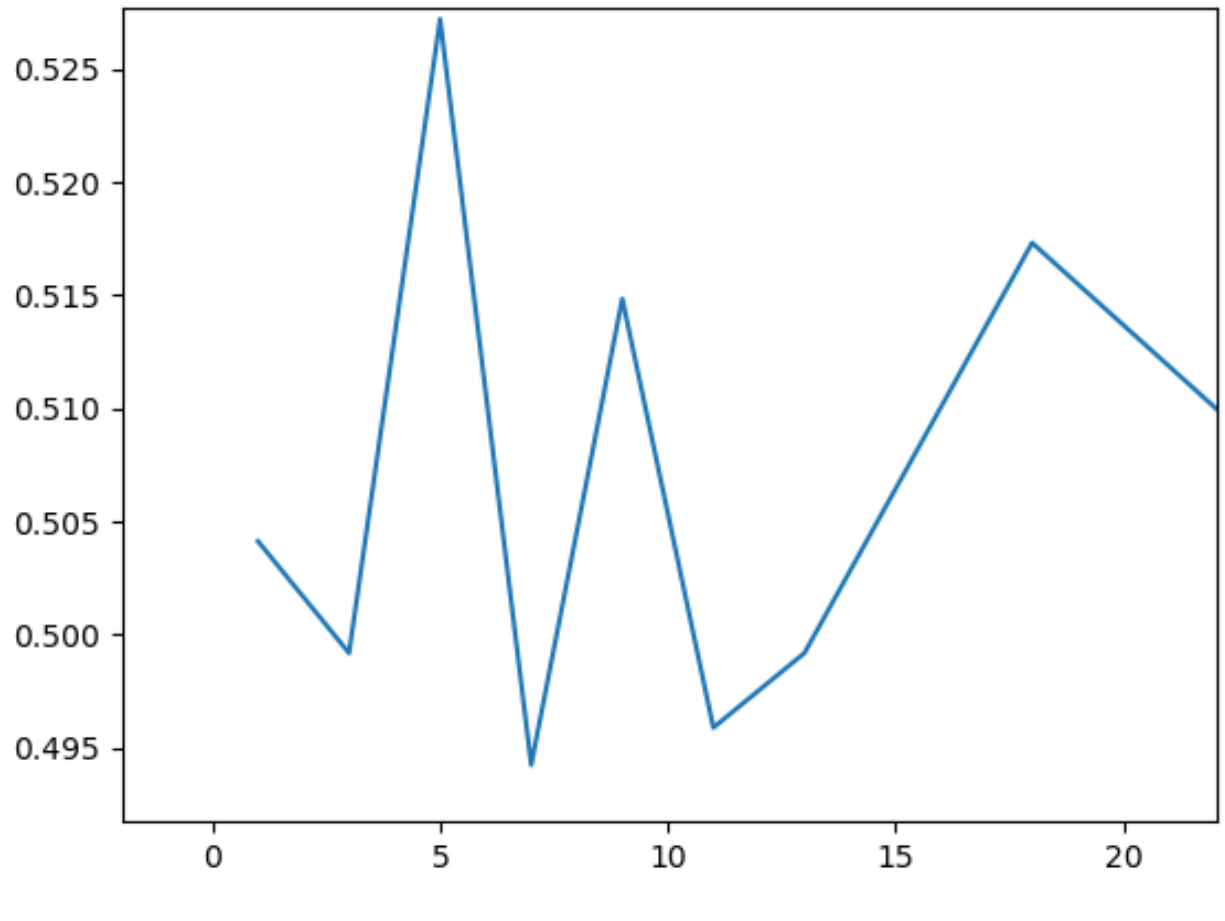
Nombre d'images	600	8376
Meilleure précision	0.613	0.689
Temps (1 époque)	60s	570s

Recherche du meilleur algorithme

Influence de la couche de convolution

Mesures sur 600 images ; 5 époques

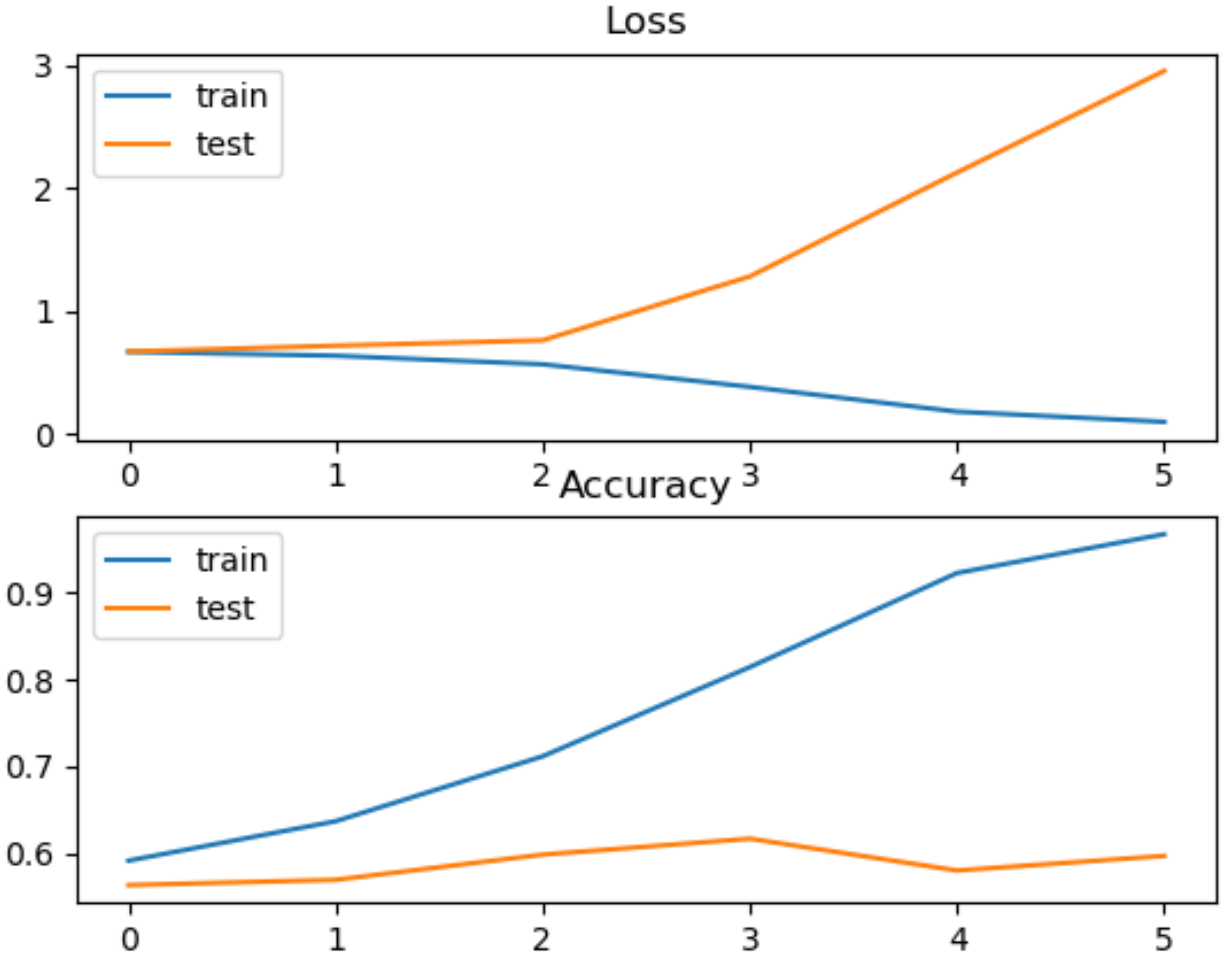
Précision



Dimension filtre

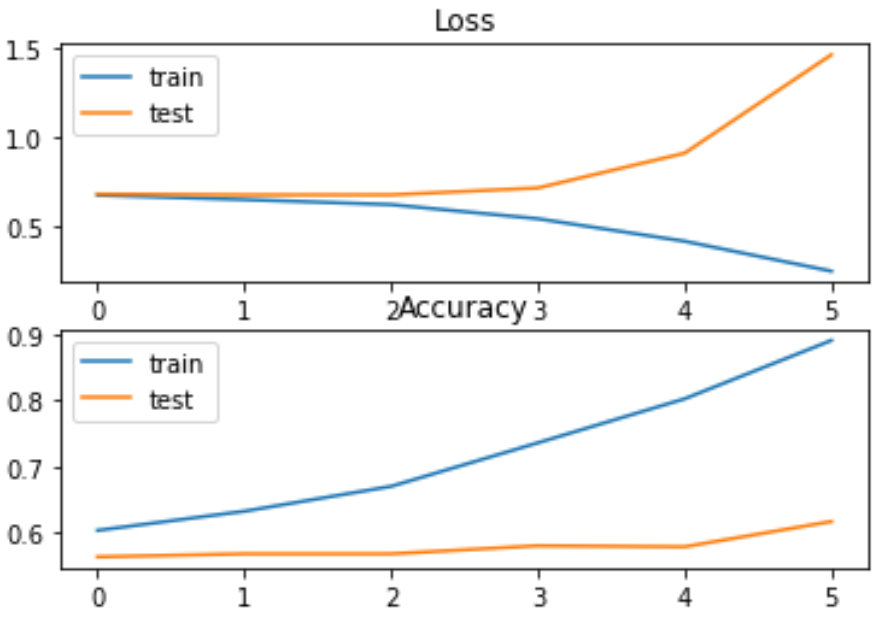
Recherche du meilleur algorithme
Overfitting

Mesures sur toutes les images ; 10 époques

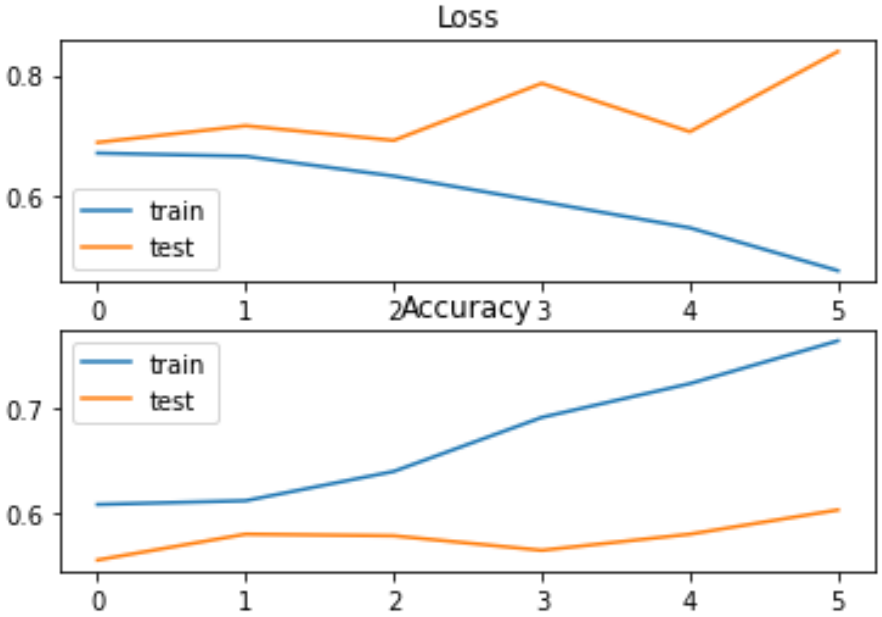


Recherche du meilleur algorithme
Ajout de couches

Ajout d'une couche de convolution et une couche entièrement connectée



Ajout de deux couches de convolution et deux couches entièrement connectées

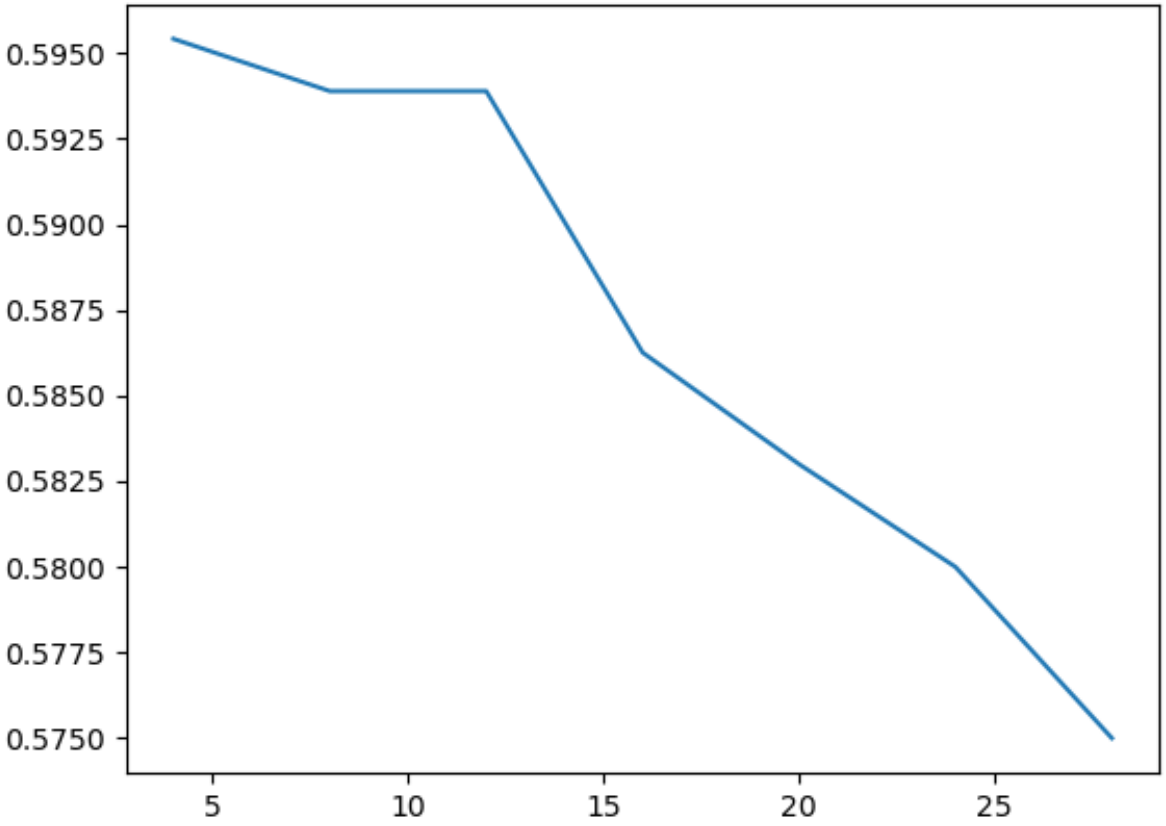


→ Précision légèrement supérieure mais temps d'entraînement plus important

Recherche du meilleur algorithme

Tailles des lots

Précision



Taille des lots

Conclusion

Meilleure précision obtenue : 0.738
➔Résultats insuffisants pour le domaine médical

Problème très complexe -> modèle difficile à ajuster

Nécessité d'une grande puissance de calcul et de beaucoup de mémoire

Axes d'amélioration :

- Meilleure qualité de la base de donnés
- *Data augmentation*
- S'intéresser à chaque paramètre plus en détail (nombre de couches, de neurones
- Moyens matériels bien plus importants

Annexes

Modules

```
1   from tensorflow.keras.models import Sequential
2   from tensorflow.keras.layers import Conv2D
3   from tensorflow.keras.layers import MaxPooling2D
4   from tensorflow.keras.layers import Dense
5   from tensorflow.keras.layers import Flatten
6   from tensorflow.keras.preprocessing.image import load_img
7   from tensorflow.keras.preprocessing.image import img_to_array
8   from tensorflow.image import rgb_to_grayscale
9   from tensorflow.image import resize
10  from matplotlib import pyplot
11  from os import listdir
12  import pandas as pd
13  import numpy as np
14  import random as rd
15
```

Annexes

Définition de fonctions

```
43  def charger_dataset(membre,tab_chem, longueur=100000):
44      """Charge le dataset correspondant à une certaine partie du corps
45      à partir des chemins données sous forme de liste ou d'array"""
46
47      dataset, label = [], []
48
49      i=0
50
51      for elem in tab_chem:
52          dossier = elem[0]
53          if not(membre in dossier):
54              continue
55
56          if i > longueur: break
57
58          for img in listdir(dossier):
59              dataset.append(charger_img(dossier+img))
60              label.append(elem[1])
61              i+=1
62              print(i)
63
64      dataset = np.asarray(dataset)
65      label = np.asarray(label)
66
67      return (dataset,label)
```

Annexes

Définition de fonctions

```
69  def charger_dataset_partiel(membre,tab_chem, longueur, plage):
70      """Charge le dataset correspondant à une certaine partie du corps
71      à partir des chemins données sous forme de liste ou d'array"""
72
73      dataset, label = [], []
74
75      indices = rd.sample(range(plage[0], plage[1]), longueur)
76
77      for i in range(longueur):
78          elem = tab_chem[indices[i]]
79          dossier = elem[0]
80          if not(membre in dossier):
81              i-=1
82              continue
83
84          for img in listdir(dossier):
85              dataset.append(charger_img(dossier+img))
86              label.append(elem[1])
87              print(i)
88
89      dataset = np.asarray(dataset)
90      label = np.asarray(label)
91
92      return (dataset,label)
```

Annexes

Définition de fonctions

```
95  def define_model(ker_size):
96      """Modèle CNN utilisé pour la prédiction des anomalies"""
97
98      model = Sequential()
99      model.add(Conv2D(32, ker_size, activation='relu', input_shape=(300, 300, 1)))
100     model.add(MaxPooling2D((2, 2)))
101     model.add(Conv2D(64, ker_size, activation='relu'))
102     model.add(MaxPooling2D((2, 2)))
103     model.add(Flatten())
104     model.add(Dense(64, activation='relu'))
105     model.add(Dense(64, activation='relu'))
106     model.add(Dense(1, activation='sigmoid'))
107     # compile model
108     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
109     return model
110
111
112 def prep_model(dataset, label, b_s, valX, valY, nb_e =5, ker_size = (5,5)):
113
114     model = define_model(ker_size)
115     history = model.fit(dataset, label, validation_data=(valX,valY), batch_size=b_s, epochs=nb_e)
116
117     return model, history
```

Annexes

Définition de fonctions

```
121  # normalize images
122  def prep_normalize(train, test):
123      """Les pixels sont linéairement distribués dans [0,1]"""
124      # convert from integers to floats
125      train_norm = train.astype('float32')
126      test_norm = test.astype('float32')
127      # normalize to range 0-1
128      train_norm = train_norm / 255.0
129      test_norm = test_norm / 255.0
130      # return normalized images
131      return train_norm, test_norm
```

Annexes

Exemple d'exécution d'un modèle

```
#Exécution du programme
train_ds, train_label = charger_dataset("SHOULDER", charger_chemins("MURA-v1.1", "train"))
v_ds, v_label = charger_dataset("SHOULDER", charger_chemins("MURA-v1.1", "valid"))

#pre-processing
train_ds, v_ds = prep_normalize(train_ds, v_ds)


#Entrainement
model, history = prep_model(train_ds, train_label, 8, v_ds, v_label, 10, (5,5))


#évaluation du modèle
_, train_acc = model.evaluate(train_ds, train_label)
_, test_acc = model.evaluate(v_ds, v_label)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

# plot loss during training
pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()

# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Accuracy')
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```