

Détection de fracture du col du fémur par un CNN

03.01.2022

Djeinaba Mamadou Ba

INSA Euro-Méditerranéenne de Fès

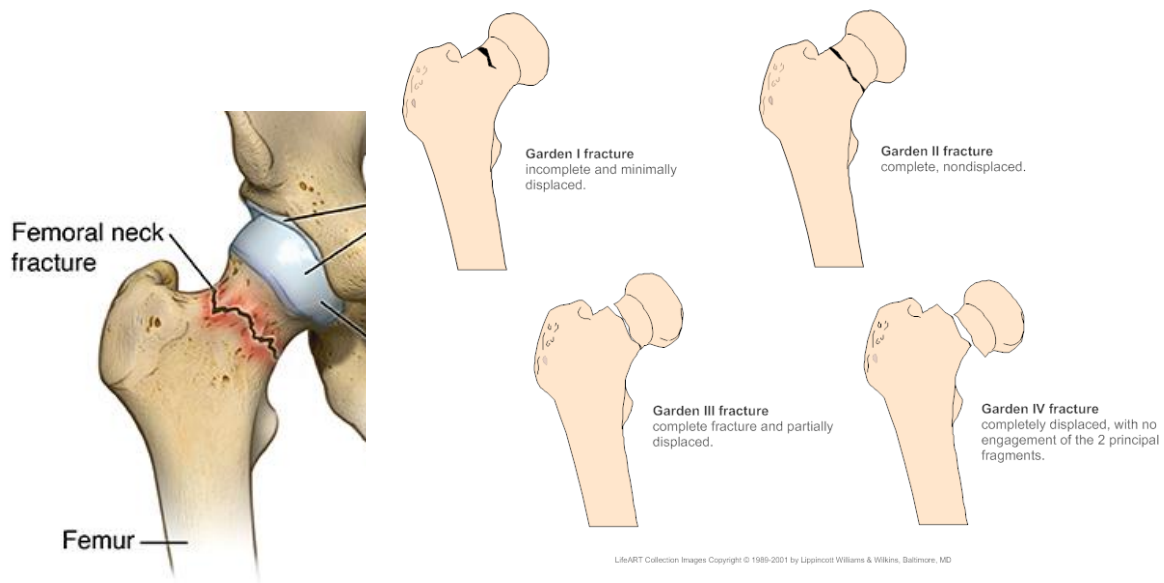
Introduction

La fracture du col du fémur est une fissure qui intervient près de l'articulation de la hanche suite à un choc.

il existe 4 types de fractures du col de fémur

1. **garden type I** : une fracture légère du col engrenée en valgus
2. **garden type II** : une fracture plus importante que le type 1 mais sans déplacement.
3. **garden III** : fracture avec déplacement
4. **garden IV**: fracture avec déplacement important et désolidarisation

L'objectif de ce projet est d'utiliser un réseau de neurones à convolutions afin de faire la détection de fractures du col de fémur.



1. Data Acquisition

La phase de collecte de données à constituer l'étape la plus difficile du projet étant donnée qu'il n'existe pas de dataset d'images radiographique de fracture du col de fémur disponible sur internet.

Les données ont été recueillies sur de multiple site internet traitant de sujet médicaux et dont le diagnostique des images à déjà été effectué, ce qui à permet de classer les xray des fractures en fonctions de leurs type.

plus de 40% des images proviennent du site de radiographie radiopedia.org où toutes les images sont diagnostiquées et classées selon leurs type.

2. Chargement des librairies

```
import numpy as np
import random
import pandas as pd
import os
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import cv2
import shutil
from glob import glob

import matplotlib.pyplot as plt
import math
%matplotlib inline
```

3. Chargement des données

```
dataset_dir = "Data"
fichier_gardenI= os.path.join('Data/Garden I/')
fichier_gardenII= os.path.join('Data/Garden II/')
fichier_gardenIII= os.path.join('Data/Garden III/')
fichier_gardenIV= os.path.join('Data/Garden IV/')
```

Les données sont classées selon leurs types sur 4 repertoires.

```
#nombre d'images contenues dans les fichiers
#récupération des images png, jpg, jpeg

#Garden I
images_gardenI = glob(os.path.join(fichier_gardenI, "*.png"))
images_gardenI.extend(glob(os.path.join(fichier_gardenI, "*.jpg")))
images_gardenI.extend(glob(os.path.join(fichier_gardenI, "*.jpeg")))

#GardenII
images_gardenII = glob(os.path.join(fichier_gardenII, "*.png"))
images_gardenII.extend(glob(os.path.join(fichier_gardenII, "*.jpg")))
images_gardenII.extend(glob(os.path.join(fichier_gardenII, "*.jpeg")))

#GardenIII
images_gardenIII = glob(os.path.join(fichier_gardenIII, "*.png"))
images_gardenIII.extend(glob(os.path.join(fichier_gardenIII, "*.jpg")))
images_gardenIII.extend(glob(os.path.join(fichier_gardenIII, "*.jpeg")))

#GardenIV
images_gardenIV = glob(os.path.join(fichier_gardenIV, "*.png"))
images_gardenIV.extend(glob(os.path.join(fichier_gardenIV, "*.jpg")))
images_gardenIV.extend(glob(os.path.join(fichier_gardenIV, "*.jpeg")))
```

Les images sont de types jpg, png et jpeg.

```
#nombre d'image de chaque type de fracture
total_gardenI = len(images_gardenI)
total_gardenII = len(images_gardenII)
total_gardenIII = len(images_gardenIII)
total_gardenIV = len(images_gardenIV)

print("le nombre d'image de Type garden 1 est de : {}".format(total_gardenI))
print("le nombre d'image de Type garden 2 est de : {}".format(total_gardenII))
print("le nombre d'image de Type garden 3 est de : {}".format(total_gardenIII))
print("le nombre d'image de Type garden 4 est de : {}".format(total_gardenIV))

le nombre d'image de Type garden 1 est de : 24
le nombre d'image de Type garden 2 est de : 18
le nombre d'image de Type garden 3 est de : 21
le nombre d'image de Type garden 4 est de : 38
```

On a 24 images de types garden 1, 18 garden II, 21 Garden 3 et 34 de types garden 4. un dataset de 101 images radiographiques.

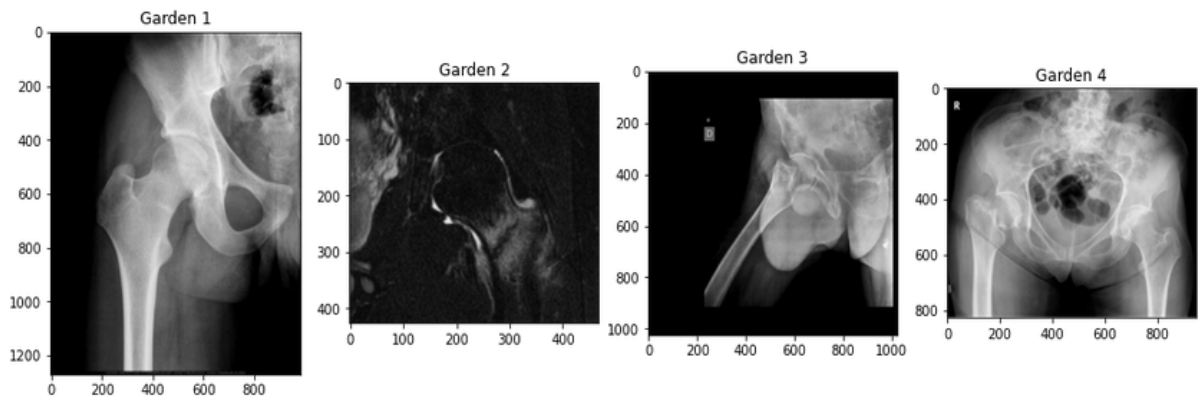
4. Traitements des données

```
#transformation des images en matrice
im_gardenI = cv2.imread(os.path.join(images_gardenI[9]))
im_gardenII = cv2.imread(os.path.join(images_gardenII[3]))
im_gardenIII = cv2.imread(os.path.join(images_gardenIII[14]))
im_gardenIV = cv2.imread(os.path.join(images_gardenIV[15]))
```

```
#Visualisation des images
f = plt.figure(figsize=(16, 16))
f.add_subplot(1, 4, 1)
plt.imshow(im_gardenI)
plt.title("Garden 1")
f.add_subplot(1,4, 2)
plt.imshow(im_gardenII)
plt.title("Garden 2")
f.add_subplot(1, 4, 3)
plt.imshow(im_gardenIII)
plt.title("Garden 3")
f.add_subplot(1, 4, 4)
plt.imshow(im_gardenIV)
plt.title("Garden 4")
```

Conversion des images en matrices avec la librairie opencv afin de les visualiser avec matplotlib.pyplot.

```
Out[6]: Text(0.5, 1.0, 'Garden 4')
```



```
#shape des images  
im_gardenI.shape
```

```
(1271, 979, 3)
```

```
im_gardenII.shape
```

```
(427, 468, 3)
```

```
im_gardenIII.shape
```

```
(1024, 1024, 3)
```

```
im_gardenIV.shape
```

```
(829, 951, 3)
```

Nos images sont de différentes shapes, il faut donc les redimensionner pour qu'ils aient le même shape.

```
# Create Train-Test Directory
subdirs = ['train/', 'test/']
for subdir in subdirs:
    labeldirs = ['Garden I', 'Garden II', 'Garden III', 'Garden IV']
    for labeldir in labeldirs:
        newdir = subdir + labeldir
        os.makedirs(newdir, exist_ok=True)
```

Création d'un répertoire train et test pour diviser les données d'entraînement des données de test.

Sélection des données pour le test

Sélection d'un ratio de 0.15 des données pour les données de test.

```
#division de l'ensemble des données en train et test

# seed random number generator
random.seed(123)

test_ratio = 0.15

for cases in [GardenI, GardenII, GardenIII, GardenIV]:
    total_cas = len(cases['images']) #nombre total de cas
    nombre_selectionner = int(test_ratio * total_cas) #nombre d'image pour le test

    print(cases['class'], f'le nombre de données sélectionnées pour le test est : {nombre_selectionner}')

    list_of_random_files = random.sample(cases['images'], nombre_selectionner) #random files selected

    for files in list_of_random_files:
        shutil.copy2(files, 'test/' + cases['class'])
```

Garden I le nombre de données sélectionnées pour le test est : 3
 Garden II le nombre de données sélectionnées pour le test est : 2
 Garden III le nombre de données sélectionnées pour le test est : 3
 Garden IV le nombre de données sélectionnées pour le test est : 5

Sélection des données pour l'apprentissage

```
#l'ensemble d'entraiment
for cases in [GardenI, GardenII, GardenIII, GardenIV]:
    image_test = os.listdir('test/' + cases['class'])
    for images in cases['images']:
        if images.split('/')[-1] not in image_test: #exclude test files from shutil.copy
            shutil.copy2(images, 'train/' + cases['class'])
```

```
total_train_GardenI = len(os.listdir('train/Garden I/'))
total_train_GardenII = len(os.listdir('train/Garden II/'))
total_train_GardenIII = len(os.listdir('train/Garden III/'))
total_train_GardenIV = len(os.listdir('train/Garden IV/'))

print("images d'entraînement Garden 1: {}".format(total_train_GardenI))
print("images d'entraînement Garden 2: {}".format(total_train_GardenII))
print("images d'entraînement Garden 3: {}".format(total_train_GardenIII))
print("images d'entraînement Garden 4: {}".format(total_train_GardenIV))
```

```
images d'entraînement Garden 1: 21
images d'entraînement Garden 2: 16
images d'entraînement Garden 3: 18
images d'entraînement Garden 4: 33
```

```

batch_size = 32
epochs = 10
#redimensionnement des pixels
IMG_HEIGHT = 512
IMG_WIDTH = 512

```

Initialement, le modèle à un batch size de 32 , 10 époques pour l'entraînement et redimensionnement des images.

5. Augmentation des données

Étant donné que les données ne sont pas volumineuses, il est nécessaire de passer par une phase d'augmentation des données pour pouvoir entraîner le modèle CNN.

```

train_image_generator = ImageDataGenerator(rescale=1./255) # generateur pour les données d'entrainement
test_image_generator = ImageDataGenerator(rescale=1./255) # generateur pour les données de test

```

```

train_dir = os.path.join('train')
test_dir = os.path.join('test')

```

```

#melange des données
train_data_gen = train_image_generator.flow_from_directory(batch_size=batch_size,
                                                         directory=train_dir,
                                                         shuffle=True,
                                                         target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         class_mode='binary')

```

Found 88 images belonging to 4 classes.

```

test_data_gen = test_image_generator.flow_from_directory(batch_size=batch_size,
                                                         directory=test_dir,
                                                         target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         class_mode='binary')

```

Found 13 images belonging to 4 classes.

Pour augmenter les données , on utilise la librairie **ImageDataGenerator**.

On a créer un générateur de données pour les données d'entraînement et les données de test.

6. Création du modèle CNN

```
model = Sequential([
    Conv2D(32, 3, padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(128, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='softmax')
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Le modèle CNN créé à au niveau de sa couche 1 , 32 filtres convolutifs, 64 filtres pour la deuxième couche et 128 pour la troisième couche de convolutions. après l'application des filtres de convolutions, on appelle à la fonction **Max Pooling 2D** pour récupérer les features les plus importantes des images puis la matrice est aplatie avant d'être envoyé vers le Réseau de neurones classiques composé de deux couches Dense.

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 512, 512, 32)	896
max_pooling2d (MaxPooling2D)	(None, 256, 256, 32)	0
conv2d_1 (Conv2D)	(None, 256, 256, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 64)	0
conv2d_2 (Conv2D)	(None, 128, 128, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 128)	0
flatten (Flatten)	(None, 524288)	0
dense (Dense)	(None, 128)	67108992
dense_1 (Dense)	(None, 1)	129
Total params: 67,202,369		
Trainable params: 67,202,369		
Non-trainable params: 0		

Résumé du modèle CNN.

```
history = model.fit(
    train_data_gen,
    epochs=epochs
)
```

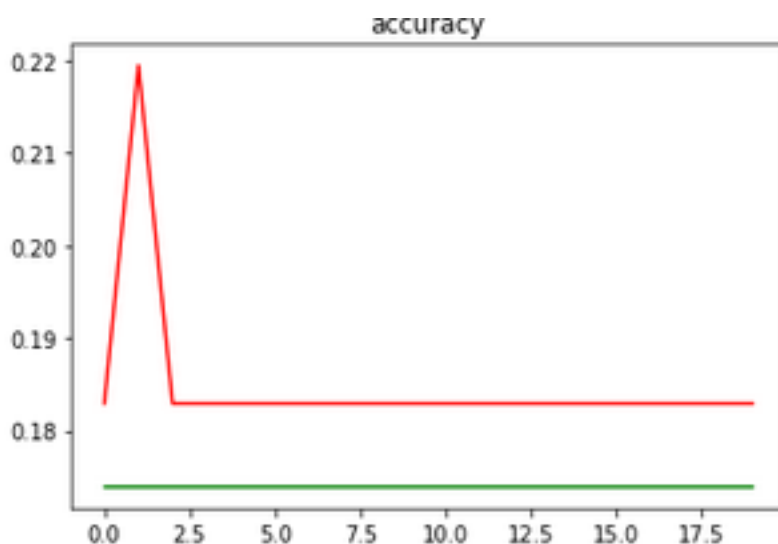
Epoch 1/10

/home/djeinaba/anaconda3/lib/python3.8/site-packages/keras/backend.py:4993: UserWarning: "`binary_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
warnings.warn(

```
3/3 [=====] - 24s 8s/step - loss: -86.4877 - accuracy: 0.1818
Epoch 2/10
3/3 [=====] - 29s 7s/step - loss: -842.1724 - accuracy: 0.1818
Epoch 3/10
3/3 [=====] - 37s 5s/step - loss: -3727.1123 - accuracy: 0.1818
Epoch 4/10
3/3 [=====] - 17s 6s/step - loss: -14661.8975 - accuracy: 0.1818
Epoch 5/10
3/3 [=====] - 17s 6s/step - loss: -45965.9023 - accuracy: 0.1818
Epoch 6/10
3/3 [=====] - 18s 5s/step - loss: -107226.2422 - accuracy: 0.1818
Epoch 7/10
3/3 [=====] - 18s 6s/step - loss: -250413.7656 - accuracy: 0.1818
Epoch 8/10
3/3 [=====] - 18s 6s/step - loss: -515401.1875 - accuracy: 0.1818
Epoch 9/10
3/3 [=====] - 18s 5s/step - loss: -1008737.6250 - accuracy: 0.1818
Epoch 10/10
3/3 [=====] - 18s 6s/step - loss: -1765462.1250 - accuracy: 0.1818
```

Le premier modèle a donné une accuracy très faible 18% du fait que les données sont extrêmement petites pour un CNN

après plusieurs entraînements du modèle et augmentation des paramètres du modèle, le meilleur score obtenu est de 21% avant de revenir à 18%



7. Comparaison avec un VGG19

VGG19 est une architecture de réseau neuronal à convolution (CNN). Elle est considérée comme l'une des excellentes architectures de modèle de vision à ce jour. La chose la plus unique à propos de VGG19 est qu'au lieu d'avoir un grand nombre d'hyper-paramètres, ils se sont concentrés sur des couches de convolution de filtre 3x3 et utilisaient toujours le même rembourrage et la même couche maxpool de filtre 2x2. Il suit cet arrangement de convolution et max pool couches de manière cohérente dans toute l'architecture. Au final, il dispose de 2 FC (couches entièrement connectées) suivis d'un softmax pour la sortie. Le 19 dans VGG19 fait référence à 19 couches qui ont des poids. Ce réseau est un assez grand réseau et il compte environ 138 millions (environ) de paramètres.

Afin de comparer la performance de notre CNN avec un autre modèle, on a choisi le **VGG19** dont l'architecture est la suivante :

II. Deuxième modèle: VGG

```
[50]: from tensorflow.keras.applications.vgg19 import VGG19
      from keras.utils.vis_utils import plot_model
      modelVGG = VGG19()
```

```
[51]: modelVGG.summary()
```

Model: "vgg19"

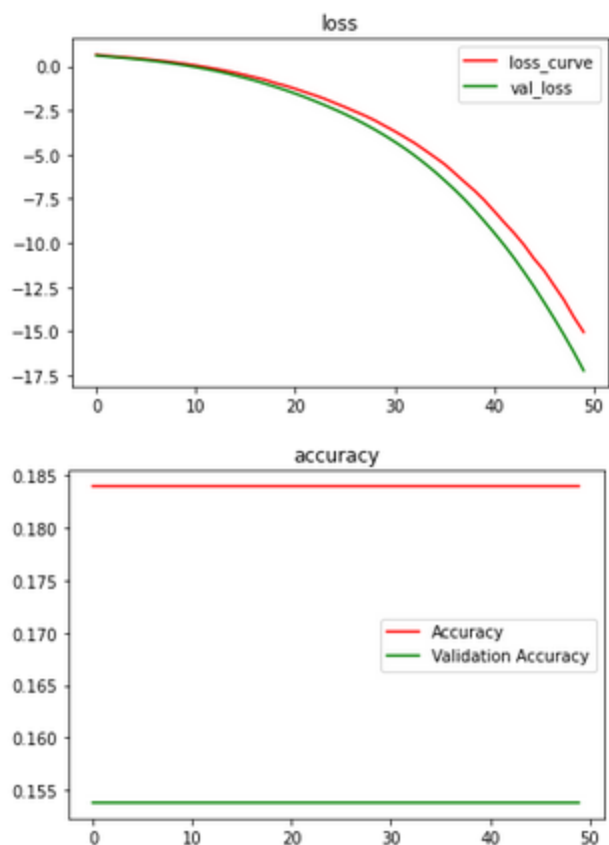
Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
Total params: 143,667,240		
Trainable params: 143,667,240		
Non-trainable params: 0		

Visualisation des résultats du VGG

```
loss_curve = history.history["loss"]
accuracy = history.history["accuracy"]
val_loss = history.history["val_loss"]
val_accuracy = history.history["val_accuracy"]


plt.plot(loss_curve, color='r', label='loss')
plt.plot(val_loss, color='g', label='val loss')
plt.title("loss")
plt.legend(["loss_curve", "val_loss"])
plt.show()

plt.plot(accuracy, color='r', label='loss')
plt.plot(val_accuracy, color='g', label='val loss')
plt.title("accuracy")
plt.legend(["Accuracy", "Validation Accuracy"])
plt.show()
```



Après entraînement du VGG 19, on constate que l'accuracy est de 18% exactement comme le CNN malgré que le VGG ait une architecture beaucoup plus robuste en termes de couches de convolution et plus de couche connecté (avec 4096 layers pour chacune des dernières couches).

Néanmoins lors des différents entraînement, le CNN à donner une accuracy de 21% , supérieure à celle du VGG. Ceci peut être expliqué par le fait que le



VGG demande plus de données (1.2 millions lors de son premier entraînement avec imagenet) pour l'entraînement que le CNN.

La contrainte principale de l'augmentation de l'accuracy réside dans le fait que les données d'entraînements sont petites empêchant ainsi d'avoir une bonne accuracy pour la classification du modèle.