CNN___Tuning___Covid_19

June 5, 2021

1 IRGUI ILYAS, NGAKAM TCHEUMBE PESCIANY LAFORTUNE

2 Détection du COVID-19 dans les images radiographiques avec CNN & Tuning des hyperparamètres

2.1 Importation des bibliothèques

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
from imutils import paths
import os
import keras

from keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
from keras.models import Sequential
from keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from keras.wrappers.scikit_learn import KerasClassifier

from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split, GridSearchCV
```

2.2 Chargement du dataset

2.2.1 Le dataset utilisé contient 25 images Normales et 25 images Covid en noir et blanc.

2.2.2 Nous avons importé le dataset dans le drive pour l'utiliser plus tard.

[2]:
```
imagePaths = list(paths.list_images("/content/drive/MyDrive/Colab Notebooks/
↪dataset"))
imgs = []
labels = []



for imagePath in imagePaths:
        label = imagePath.split(os.path.sep)[-2]

        image = cv2.imread(imagePath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Transformation en␣
 ↪RGB(rouge vert bleu)
        image = cv2.resize(image, (224, 224)) # 224x224 pixels

        imgs.append(image)
        labels.append(label)

imgs = np.array(imgs) / 255.0 # La Normalisation
labels = np.array(labels)
```

[3]:
```
# Encodage des labels en binaire
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels) # Convertion des classes vectorielles en␣
 ↪classes binaires

# Partition du dataset en training and testing splits avec 80% training et 20%␣
 ↪pour le testing
(trainX, testX, trainY, testY) = train_test_split(imgs, labels,
        test_size=0.20, stratify=labels, random_state=42)
```

2.3 Création du modèle

2.3.1 Dans notre cas, nous avons fait le tuning de tous les hyperparamètres (sauf le learning rate) en même temps.

2.3.2 Pour cela, nous avons donc définit les valeurs par défaut de certains hyperparamètres qui doivent être utilisés par d'autres.

[4]:
```
def create_model(activation = 'relu', neurons=32, optimizer='adam',␣
 ↪dropout_rate=0.0, weight_constraint=0):
   model = Sequential()
   model.add(Conv2D(neurons, kernel_size=(3,3), activation = activation,␣
 ↪input_shape=(224,224,3)))
   model.add(MaxPool2D(pool_size=(2, 2)))
   model.add(Flatten())
   model.add(Dropout(dropout_rate))
   model.add(Dense(neurons*2, activation= activation))
   model.add(Dense(2,activation='sigmoid'))
   model.compile(loss="binary_crossentropy", optimizer = optimizer,␣
 ↪metrics=["accuracy"])
   return model
```

2.4 Après plusieurs tests avec de nombreuses valeurs pour nos hyperparamètres, nous avons repéré la valeur idéale et avons ajusté nos listes à deux valeurs pour chacun des hyperparamètres pour gagner en temps.

```
[5]: model = KerasClassifier(build_fn=create_model, batch_size=5, epochs=10,␣
      ↪verbose=0) #Create model

     #Define grid search parameters
     batch_size=[8,10]
     epochs = [20,25]
     activation = ['softmax','relu']
     neurons = [25,32]
     optimizer = ['Adam', 'Adadelta']
     dropout_rate = [0.2,0.5]
```

2.5 Ensuite nous avons définit notre dictionnaire et appliqué GridSearchCV

```
[6]: param_grid = dict(batch_size=batch_size,
                       epochs=epochs,
                       activation = activation,
                       neurons = neurons,
                       optimizer = optimizer,
                       dropout_rate = dropout_rate)

     grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
     grid_result = grid.fit(trainX, trainY)
```

```
/usr/local/lib/python3.7/dist-
packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

2.6 Summurize Results

```
[7]: print("Best_parameters: %f,%s" %(grid_result.best_score_, grid_result.
      ↪best_params_))

     means = grid_result.cv_results_['mean_test_score']
     stds = grid_result.cv_results_['std_test_score']
     params = grid_result.cv_results_['params']

     for mean ,stdev, param in zip(means, stds, params):
       print("%f(%f) avec: %r" %(mean,stdev,param))
```

```
Best_parameters: 1.000000,{'activation': 'relu', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.628205(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.628205(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 8,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.500000(0.157019) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.628205(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.628205(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.371795(0.090655) avec: {'activation': 'softmax', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.950549(0.035039) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.743590(0.191880) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.950549(0.035039) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.822344(0.097913) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.642857(0.204457) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
1.000000(0.000000) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.794872(0.145048) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.565934(0.264003) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.950549(0.035039) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.769231(0.166173) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.598901(0.099206) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.974359(0.036262) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.747253(0.077704) avec: {'activation': 'relu', 'batch_size': 8, 'dropout_rate':
0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.719780(0.160442) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.771062(0.111376) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
0.924908(0.062861) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.771062(0.111376) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.822344(0.097913) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.2, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
0.950549(0.035039) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adam'}
0.540293(0.276344) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 25, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adam'}
0.565934(0.264003) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 20, 'neurons': 32, 'optimizer': 'Adadelta'}
1.000000(0.000000) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adam'}
0.668498(0.222706) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 25, 'optimizer': 'Adadelta'}
0.976190(0.033672) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adam'}
0.796703(0.098391) avec: {'activation': 'relu', 'batch_size': 10,
'dropout_rate': 0.5, 'epochs': 25, 'neurons': 32, 'optimizer': 'Adadelta'}
```

2.7 Le score optimal obtenu est de : 100% avec les hyperparamètres suivants :

2.8 {‘activation’: ‘relu’, ‘batch_size’: 8, ‘dropout_rate’: 0.2, ‘epochs’: 25, ‘neurons’: 32, ‘optimizer’: ‘Adam’}