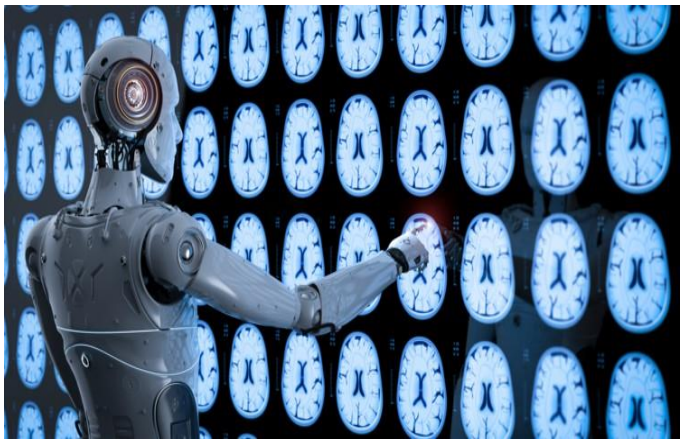
Groupe N° : 32

P2022 ECC

ÉCOLE CENTRALE CASABLANCA

LIVRABLE-3

ARCHITECTURE DU PROGRAMME TRAIN_COVID19.PY



Réalisé par :

Amine NAIT CHARIF
Doha FEDDOUL
Florian BOUBOUSSI DONGMO
Youssef CHOUHAIDI

Tuteur Entreprise :

M. Fayçal NOUSHI

Tuteur école :

M. Mohamed-Hassan KHALILI

SOMMAIRE

Le programme initial a été réalisé par Adrian Rosebrock

I-COLLECTE DES IMAGES ET CRÉATION DU DATASET

Le programme initial utilise 50 images : 25 de patients sains et 25 de patients atteins du covid-19 :

- Les 25 images des patients malades proviennent du dataset du docteur Joseph Cohen (posté sur GitHub et enrichi au fil du temps).
- Les 25 images des patients sains proviennent de la base de données de Kaggle sur les radiographies thoraciques (une base de données constituées d'images thoraciques de patients sains et atteints de pneumonie). Ces images ont été extraite grâce au programme **sample_kaggle_dataset.py**.

Après la collecte des images, un premier programme a permis la création du dataset (qui sera par la suite utilisé par le programme **train_covid19.py**) : **build_covid_dataset.py**.

II-ARCHITECTURE ET ANALYSE DU PROGRAMME :

Le programme est constitué de 8 principaux blocs :

1- L'importation des LIBRAIRIES et FONCTIONS nécessaires :

```
# USAGE
# python train.py --dataset dataset

# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
```

Figure 1 : Train_covid19

On remarque au tout début au fort emploi de TensorFlow.Keras.

TensorFlow est un outil libre de développement de programme d'apprentissage automatique compatible avec Python ; TensorFlow.Keras est son interface de programmation d'application (en anglais API pour Application Programming Interface) par excellence pour la rédaction et l'entrainement de modèle d'apprentissage.

On peut remarquer l'importation de VGG16. VGG16 est un réseau de neurone convolutif qui permettra d'entrainer le programme.

Scikit-learn (sklearn) est la bibliothèque de base de Python.

Matplotlib est la bibliothèque Python permettant le tracé des courbes. Elle permettra de tracer la courbe d'apprentissage du programme.

OpenCV (cv2) permettra le traitement préalable des images. En effet, VGG16 par exemple ne peut traiter que des images RGB de taille $224 * 224$.

Argparse est le module qui permettra à l'utilisateur de fournir des paramètres d'entrée (ici les images) ou de sortie au programme pour son exécution.

2- Initialisation des arguments :

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
        help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
        help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str, default="covid19.model",
        help="path to output loss/accuracy plot")
args = vars(ap.parse_args())

# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-3
EPOCHS = 25
BS = 8
```

Figure 2 : Train_covid19

Le programme devant traiter des images, il est nécessaire de fournir au programme les images en question au préalable. À l'aide de la commande ap.add_argument("-d", "--dataset", required=True, help="path to input dataset"), l'utilisateur pourra au début du programme, mentionner le nom du dossier où se trouve ses images : la mention se fait comme suit : ''-d nom du dossier'' ou ''-dataset nom du dossier''.

3- Chargement des images et prétraitement :

```
# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# loop over the image paths
for imagePath in imagePaths:
        # extract the class label from the filename
        label = imagePath.split(os.path.sep)[-2]

        # load the image, swap color channels, and resize it to be a fixed
        # 224x224 pixels while ignoring aspect ratio
        image = cv2.imread(imagePath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (224, 224))

        # update the data and labels lists, respectively
        data.append(image)
        labels.append(label)

# convert the data and labels to NumPy arrays while scaling the pixel
# intensities to the range [0, 255]
data = np.array(data) / 255.0
labels = np.array(labels)
```

Figure 3 : Train_covid19

Le chemin d'accès ayant été indiqué, les images peuvent être chargées par le programme :

Afin que l'utilisateur sache que le programme se trouve à ce niveau de l'exécution, le message ''[INFO] loading images …'' est affiché sur l'écran.

Toutes les images sont insérées dans la liste imagePaths.

Pour chaque image le traitement suivant est opéré :

- Extraction de son label et insertion de celui-ci dans la liste labels : le label renseigne si l'image est celle d'un patient sain ou malade.
- Transformation de l'image grace à la bibliothèque OpenCV en une image RGB.
- Redimensionnement de l'image en une image de taille $224 * 224$: comme mentionné plus haut, VGG16 ne peut traiter que des images de cette taille.
  Les nouvelles images sont mises dans la liste data et leur intensité mise sur une échelle de 0 à 1 (et non plus de 0 à 255 comme pour une image RGB classique).
  Enfin, les listes labels et data sont transformées en des tableaux.

4- <u>Partition des images :</u>

```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.20, stratify=labels, random_state=42)

# initialize the training data augmentation object
trainAug = ImageDataGenerator(
        rotation_range=15,
        fill_mode="nearest")
```

Figure 4 : Train_covid19

Tout d'abord un codage à chaud de nos données dans la liste labels est effectué. Le codage à chaud permet de coder une donnée avec un seul chiffre en binaire : par exemple, les images de patients sains auront un code $[1,0]$ et ceux de patients malades $[0,1]$.

Après ceci, les images sont partitionnées. 80% (soit 40 images, 20 de part et d'autre) sont utilisées pour l'entrainement du programme et le reste pour le test du programme.

Pour s'assurer que le modèle reste général, une rotation des images de 15 degré est effectuée dans le sens contraire ou non des aiguilles d'une montre.

5- <u>Initialisation et réglage fin de VGG16 :</u>

```
# load the VGG16 network, ensuring the head FC layer sets are left
# off
baseModel = VGG16(weights="imagenet", include_top=False,
        input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
        layer.trainable = False
```

Figure 5 : Train_covid19

Les images étant fin prêtes, on procède par la suite à l'appel du réseau de neurones VGG16 et à des réglages fins de celui-ci ; VGG16 est constitué de plusieurs couches de réseau de neurones (une couche

de neurone étant un ensemble de neurone mis en parallèles) et dans ce cas, toutes les couches ne seront pas utilisées (entrainées). Le réglage fin consiste donc en quelque sorte à geler les couches qui ne seront pas utilisées.

6- Compilation et entrainement du programme :

a- Compilation :

```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])
```

Figure 6 : Train_covid19

Afin que l'utilisateur sache que le programme se trouve à ce niveau de l'exécution, le message ''[INFO] compiling model …'' est affiché sur l'écran.

b- Entrainement :

```
# train the head of the network
print("[INFO] training head...")
H = model.fit_generator(
        trainAug.flow(trainX, trainY, batch_size=BS),
        steps_per_epoch=len(trainX) // BS,
        validation_data=(testX, testY),
        validation_steps=len(testX) // BS,
        epochs=EPOCHS)
```

Figure 7 : Train_covid19

Afin que l'utilisateur sache que le programme se trouve à ce niveau de l'exécution, le message ''[INFO] training head …'' est affiché sur l'écran.

Pour lancer l'entrainement du modèle, la commande model.fit_generator est utilisée : le modèle est entrainé avec les images améliorées (pivotées de 15 dégrés).

7- Évaluation du modèle :

Afin que l'utilisateur sache que le programme se trouve à ce niveau de l'exécution, le message ''[INFO] evaluating network …'' est affiché sur l'écran.

- Tout d'abord, on analyse et affiche la prédiction du modèle sur chacune des images tests (est-ce que le modèle la classe comme image saine ou ''malade'').

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
        target_names=lb.classes_))
```

Figure 8 : Train_covid19

- Ensuite nous élaborons une matrice de confusion. La matrice de confusion permet de voir les résultats du programme, c.-à-d. que pour les images saines, combien ont bel et bien été prédites comme étant saines par le programme et de même avec les images de patients

malades. La matrice de confusion permettra de déterminer la précision (accuracy), la sensibilité (sensitivity) et la spécificité (specificity). La précision représente le taux de précision générale du programme sur les 10 images tests. La sensibilité représente la précision du programme sur les images des patients atteints et la spécificité sur les images des patients sains.

```
# compute the confusion matrix and and use it to derive the raw
# accuracy, sensitivity, and specificity
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])

# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

Figure 9 : Train_covid19

- Enfin, le programme affiche (enregistre) la courbe représentant l'évolution de l'entrainement du programme. Cette dernière est enregistrée sous plot.png. Toutefois, si l'utilisateur souhaite l'enregistrer sous un autre nom, il suffit lors du lancement du programme de mentionner comme argument ''-p nom du fichier'' ou ''—plot nom du fichier''.

```
# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Figure 10 : Train_covid19

8- Enregistrement du modèle :

Afin que l'utilisateur sache que le programme se trouve à ce niveau de l'exécution, le message ''[INFO] saving COVID-19 detector model …'' est affiché sur l'écran.

- Pour terminer, le programme enregistre le modèle sous le nom covid19.model. Toutefois, si l'utilisateur souhaite l'enregistrer sous un autre nom, il suffit lors du lancement du programme de mentionner comme argument ''-m nom du fichier'' ou ''—model nom du fichier''.

```
# serialize the model to disk
print("[INFO] saving COVID-19 detector model...")
model.save(args["model"], save_format="h5")
```

Figure 11 : Train_covid19

Au final, le programme peut se résumer comme suit :

- ✓ IMPORTATION DES BIBLIOTHÈQUES
- ✓ CHARGEMENT DES IMAGES ET PRÉTRAITEMENT DES CES DERNIÈRES
- ✓ OUVERTURE ET RÉGLAGE FIN DU RÉSEAU DE NEURONE VGG16
- ✓ ENTRAINEMENT DU MODÈLE
- ✓ AFFICHAGE DES RÉSULTATS DU MODÈLE
- ✓ SAUVEGARDE.