Compte Rendu de TP 06

Diagnostic par apprentissage

Réseau de neurones convolutifs pour la classification binaire d'images

Nom : BOUARICHE

Prénom : Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

Encadré par : Mme. Anissa MOKRAOUI

Prise en main de l'environnement de travail

```
from psutil import *
cpu_count() # indique le nombre de CPU
!lscpu |grep 'Model name' # CPU mode and speed
!df –h / | awk '{print $4}' # available Hard disk space
!free –h –si | awk '/Mem:/{print $2}' # Usuable memory
!nvidia-smi –L # GPU specifications.
```

```
grep: name': No such file or directory
df: –h: No such file or directory
awk: 1: unexpected character 0xe2
awk: line 2: missing } near end of file
awk: 1: unexpected character 0xe2
awk: line 2: missing } near end of file

Usage:
 free [options]

Options:
 -b, --bytes         show output in bytes
     --kilo          show output in kilobytes
     --mega          show output in megabytes
     --giga          show output in gigabytes
     --tera          show output in terabytes
     --peta          show output in petabytes
 -k, --kibi          show output in kibibytes
 -m, --mebi          show output in mebibytes
 -g, --gibi          show output in gibibytes
     --tebi          show output in tebibytes
     --pebi          show output in pebibytes
 -h, --human         show human-readable output
     --si            use powers of 1000 not 1024
 -l, --lohi          show detailed low and high memory statistics
 -t, --total         show total for RAM + swap
 -s N, --seconds N   repeat printing every N seconds
 -c N, --count N     repeat printing N times, then exit
 -w, --wide          wide output

     --help     display this help and exit
 -V, --version  output version information and exit

For more details see free(1).
ERROR: Option –L is not recognized. Please run 'nvidia-smi -h'.
```

Commentaire:

from psutil impor*: Cette ligne importe toutes les fonctions et classes de la bibliothèque psutil, qui est une bibliothèque Python permettant d'obtenir des informations sur l'utilisation du système.

cpu_count(): Cette ligne utilise la fonction cpu_count() de la bibliothèque psutil pour obtenir le nombre de processeurs logiques sur le système.

!lscpu |grep 'Model name': Cette ligne utilise une commande shell pour afficher les détails du processeur en utilisant lscpu et filtre la sortie avec grep pour récupérer la ligne contenant le modèle du CPU.

!df –h / | awk '{print $4}': Cette ligne utilise une commande shell pour afficher l'espace disque disponible en utilisant df.

!free –h –si | awk '/Mem:/{print $2}': Cette ligne utilise une commande shell pour afficher les informations sur la mémoire en utilisant free.

!nvidia-smi –L: Cette ligne utilise une commande shell pour afficher des informations sur les GPU Nvidia en utilisant nvidia-smi, en listant les dispositifs GPU disponibles.

A) Augmentation de la taille du jeu de données d'entrainement

```
from keras.preprocessing.image import ImageDataGenerator
from skimage import io
```

Commentaire

L'augmentation de données est une technique qui génère de nouvelles images en appliquant des transformations mineures (rotation, zoom, retournement, etc.) aux images existantes, aidant ainsi à augmenter la diversité des données d'entraînement.

from keras.preprocessing.image import ImageDataGenerator: Cette ligne importe la classe ImageDataGenerator de la bibliothèque Keras, qui est utilisée pour augmenter les données d'images.

from skimage import io: Cette ligne importe le module io de la bibliothèque scikit-image (skimage). Cela suggère l'utilisation de la fonction io de scikit-image pour lire et manipuler des images.

```
datagen = ImageDataGenerator(rotation_range=45, width_shift_range=0.2,
height_shift_range=0.2, shear_range=0.2, horizontal_flip=True,
zoom_range=0.2, fill_mode="constant",cval=125)
```

Commentaire:

Ce code crée une instance de la classe ImageDataGenerator de Keras avec plusieurs paramètres spécifiés pour augmenter les données d'images.

rotation_range=45: Indique la plage de valeurs pour la rotation aléatoire des images, avec une plage de 45 degrés.

width_shift_range=0.2: Spécifie la plage de décalage horizontal aléatoire des images en fraction de la largeur totale, avec une plage de 0.2 (20% de la largeur).

height_shift_range=0.2: Définit la plage de décalage vertical aléatoire des images en fraction de la hauteur totale, avec une plage de 0.2 (20% de la hauteur).

shear_range=0.2: Indique la plage de cisaillement aléatoire des images.

horizontal_flip=True: Permet les retournements horizontaux aléatoires des images, ce qui peut augmenter la variabilité des données.

zoom_range=0.2: Détermine la plage de zoom aléatoire des images, avec une plage de 0.2 (zoom de 20%).

fill_mode="constant": Spécifie la stratégie de remplissage utilisée lors de l'application de transformations. Dans ce cas, on utilise le mode "constant", ce qui signifie que les pixels nouvellement créés lors des transformations auront une valeur constante spécifiée.

cval=125: Valeur de remplissage constante utilisée lorsque fill_mode est "constant". Dans ce cas, les pixels nouvellement créés seront remplis avec la valeur 125.

```
x =io.imread("./monalisa.jpg")
x.shape
```

```
(256, 256, 3)
```

Commentaire:

La première ligne **x = io.imread("./monalisa.jpg")** lit l'image depuis le fichier spécifié et la stocke dans la variable x. La deuxième ligne **x.shape** renvoie la forme de l'image sous la forme d'un tuple, où chaque élément du tuple représente la taille de la dimension correspondante de l'image. On peut remarquer que la taille de notre image est de hauteur de 256 pixels et de largeur de 256 pixels et de 3 couches RGB, car on a une image en couleur.

```
x = x.reshape(1,256, 256, 3)
```

Commentaire:

Dans cette ligne de code, une modification de la dimension de l'image est effectuée, et une nouvelle dimension est ajoutée. Cette dimension est utilisée pour créer un lot (batch) d'images.

```
x.shape
```

```
(1, 256, 256, 3)
```

Commentaire:

Cette instruction affiche la forme actuelle de l'image après la modification de la dimension. on remarque qu'on a une nouvelle dimension de valeur 1.

```
!mkdir augmented1
!mkdir augmented2
!mkdir augmented3
!mkdir augmented4
```

Commentaire:

Ces lignes de code utilisent des commandes shell pour créer quatre répertoires différents dans le système de fichiers.

- !mkdir augmented1: Crée un répertoire nommé "augmented1".
- !mkdir augmented2: Crée un répertoire nommé "augmented2".
- !mkdir augmented3: Crée un répertoire nommé "augmented3".
- !mkdir augmented4: Crée un répertoire nommé "augmented4".

```
i=0
for batch in datagen.flow(x, batch_size=16, save_to_dir =
"augmented1", save_prefix ="aug", save_format ="png"):
  i+=1
  if i>20:
    break
```

Commentaire:

Ce code applique les transformations définies par le générateur datagen à l'image x et sauvegarde les images augmentées dans le répertoire "augmented1". La boucle s'arrête après avoir généré 20 lots d'images.

```
datagen = ImageDataGenerator(rotation_range=45, width_shift_range=0.2,
height_shift_range=0.2, shear_range=0.2, horizontal_flip=True,
zoom_range=0.2, fill_mode="nearest")
i=0
for batch in datagen.flow(x, batch_size=16, save_to_dir =
"augmented2", save_prefix ="aug", save_format ="png"):
  i+=1
  if i>20:
    break
```

Commentaire:

Ce code applique les transformations définies par le générateur datagen à l'image x et sauvegarde les images augmentées dans un autre répertoire "augmented2". La seul difference

est le mode de remplissage "nearest" qui signifie que les pixels nouvellement créés prendront la valeur du pixel le plus proche dans l'image d'origine.

```
datagen = ImageDataGenerator(rotation_range=45, width_shift_range=0.2,
height_shift_range=0.2, shear_range=0.2, horizontal_flip=True,
zoom_range=0.2, fill_mode="reflect")
i=0
for batch in datagen.flow(x, batch_size=16, save_to_dir =
"augmented3", save_prefix ="aug", save_format ="png"):
  i+=1
  if i>20:
    break
```

Commentaire:

Ce code applique la meme chose que le code précédent mais ici on a un mode de remplissage different, on a le mode de remplissage "reflect" qui remplit les pixels nouvellement créés en reflétant les valeurs des bords de l'image. Cela signifie que si un pixel est à proximité du bord de l'image et doit être rempli, sa valeur sera déterminée en réfléchissant les valeurs des pixels voisins du bord. Cela peut aider à créer une transition plus douce entre les pixels existants et les pixels nouvellement générés.

```
datagen = ImageDataGenerator(rotation_range=45, width_shift_range=0.2,
height_shift_range=0.2, shear_range=0.2, horizontal_flip=True,
zoom_range=0.2, fill_mode="wrap")
i=0
for batch in datagen.flow(x, batch_size=16, save_to_dir =
"augmented4", save_prefix ="aug", save_format ="png"):
  i+=1
  if i>20:
    break
```

Commentaire:

Ce code applique les transformations définies par le générateur datagen à l'image x et sauvegarde les images augmentées dans un repertoire, la seul difference est pour le mode de remplissage qui est "wrap" (enroulement ou répétition), ce mode s'applique si un pixel doit être rempli à l'extérieur des limites de l'image d'origine, ses valeurs seront obtenues en répétant les valeurs de pixels de l'autre côté de l'image. Cela crée une sorte d'effet de répétition ou d'enroulement.

B) Classification binaire (infectée, non-infectée) d'images cellulaires

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.layers import Activation, Dropout, Flatten, Dense
import os
import cv2
from PIL import Image
import numpy as np
```

matplotlib.pyplot et **matplotlib.image :** Utilisées pour la visualisation des images et des graphiques.

keras.preprocessing.image.ImageDataGenerator : Pour augmenter les données d'entraînement en temps réel et améliorer la généralisation du modèle.

keras.models.Sequential : Pour créer un modèle séquentiel de couches.

keras.layers : Différentes couches pour construire le modèle, y compris **Conv2D, MaxPooling2D, BatchNormalization, Activation, Dropout, Flatten et Dense.**

os : Pour les opérations liées au système d'exploitation.

cv2 : OpenCV, utilisé pour le traitement d'images.

PIL.Image : La bibliothèque Python Imaging Library pour le traitement d'images.

numpy : Pour les opérations numériques.

Exercice 01: Lecture et labelisation des données

```
! rm -r augmented1
! rm -r augmented2
! rm -r augmented3
! rm -r augmented4
```

Commentaire: Ce script shell a pour but de supprimer les anciens dossiers de données afin de libérer de l'espace disque et de la mémoire.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Commentaire:

Ce code utilise la bibliothèque Google Colab pour monter le lecteur Google Drive dans l'environnement de Colab. Cela permet d'accéder aux fichiers stockés sur Google Drive et de les utiliser dans notre environnement Colab.

```
!unzip ./drive/MyDrive/cell_images.zip
```

```
Archive:  ./drive/MyDrive/cell_images.zip
replace
cell_images/Parasitized/C100P61ThinF_IMG_20150918_144104_cell_162.png?
[y]es, [n]o, [A]ll, [N]one, [r]ename: N
```

Commentaire : Cette ligne de code shell est utilisée pour la décompression d'un fichier ZIP qui contient les données (images pour l'apprentissage).

```
!ls

cell_images  drive  MODEL  monalisa.jpg  sample_data
```

Commentaire: Cette ligne de code shell est pour la vérification de données qui sont dans le repertoire de travail.

```
import os
import cv2
import numpy as np

SIZE = 150
dataset = []
label = []

parasitized_images = os.listdir("cell_images/Parasitized/")

for i, image_name in enumerate(parasitized_images):
    if image_name.split(".")[1] == "png":
        image_path = os.path.join("cell_images/Parasitized/",
image_name)
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (SIZE, SIZE))
        dataset.append(np.array(image))
        label.append(1)
```

Commentaire:

Ce code charge des images de cellules parasitées dans un tableau (dataset) et leur attribuer l'étiquette 1 dans un tableau de labels (label).

```
import os
import cv2
import numpy as np

SIZE = 150

parasitized_images = os.listdir("cell_images/Uninfected/")

for i, image_name in enumerate(parasitized_images):
```

```
    if image_name.split(".")[1] == "png":
        image_path = os.path.join("cell_images/Uninfected/",
image_name)
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (SIZE, SIZE))
        dataset.append(np.array(image))
        label.append(0)
```

Commentaire:

On fais la meme chose que le code précédent mais dans cette partie on charge des images de cellules infectées dans un tableau (dataset) et leur attribuer l'étiquette 0 dans un tableau de labels (label).

```
dataset = np.array(dataset)
label = np.array(label)

np.shape(dataset)
```

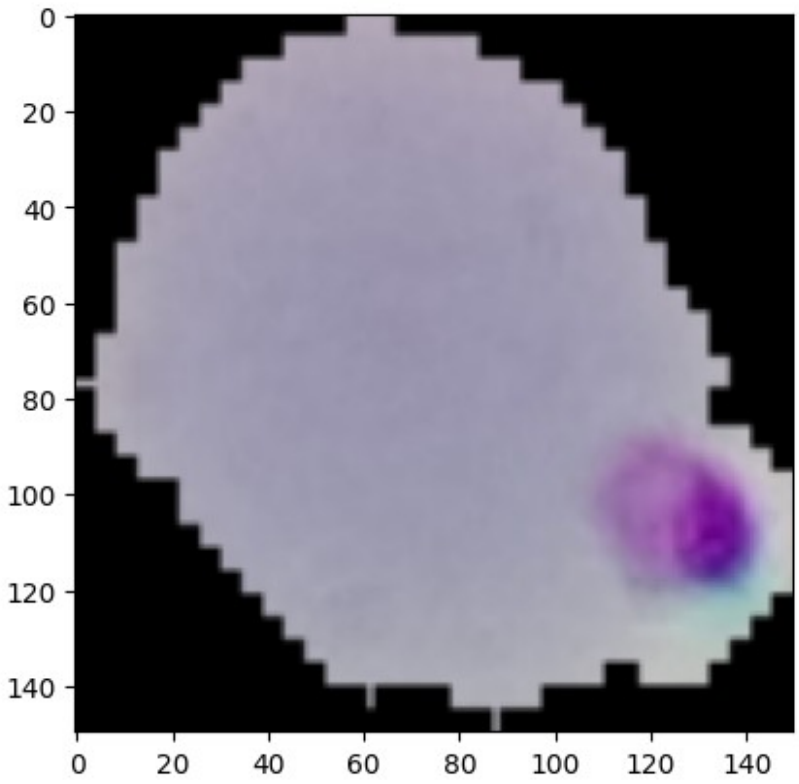
```
(27558, 150, 150, 3)
```

```
np.shape(label)
```

```
(27558,)
```

Commentaire: On remarque qu'on a 27558 images de taille 150x150x3. et on a aussi 27558 labels.

```
import random
import numpy as np
import matplotlib.pyplot as plt

image_number = random.randint(0, len(dataset) - 1)
plt.imshow(np.reshape(dataset[image_number], (150, 150, 3)))
plt.show()
print("Label de cette image : ", label[image_number])
```



Commentaire: Dans ce code on affiche une image et son label. par example on a obtenu cette image et on a obtenu le label "1" (n'est pas infectée).

Exercice 02: Construction et entrainement du modèle

```
import psutil

ram_info = psutil.virtual_memory()

print(f"Total RAM: {ram_info.total / (1024 ** 3):.2f} GB")
print(f"Used RAM: {ram_info.used / (1024 ** 3):.2f} GB")
print(f"Free RAM: {ram_info.available / (1024 ** 3):.2f} GB")

Label de cette image :  1
Total RAM: 12.67 GB
Used RAM: 8.12 GB
Free RAM: 4.20 GB
```

Commentaire:

Avec ces lignes de code on peut voire la taille occupé par le jeu de données.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataset, label,
test_size=0.20, random_state=0)
```

Commentaire:

On peut obtenir avec ces lignes de codes un jeu de données séparés de deux parties, une partie pour l'entrainement qui prend 80% de données et le reste est pour la validation.

```
X_train.dtype
```

```
dtype('uint8')
```

```
X_train = X_train.astype('float16') / 255
X_test = X_test.astype('float16') / 255

import psutil

ram_info = psutil.virtual_memory()

print(f"Total RAM: {ram_info.total / (1024 ** 3):.2f} GB")
print(f"Used RAM: {ram_info.used / (1024 ** 3):.2f} GB")
print(f"Free RAM: {ram_info.available / (1024 ** 3):.2f} GB")
```

```
Total RAM: 12.67 GB
Used RAM: 10.24 GB
Free RAM: 2.10 GB
```

Commentaire:

En raison de la capacité limitée de la RAM et afin de pouvoir utiliser le jeu de données pour l'apprentissage, on a modifié le type de données en "float16".

```
INPUT_SHAPE =(SIZE, SIZE, 3)
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = INPUT_SHAPE))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3), kernel_initializer = "he_uniform"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3), kernel_initializer = "he_uniform"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation("sigmoid"))

model.compile(loss="binary_crossentropy", optimizer = "rmsprop",
metrics="accuracy")

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 activation (Activation)     (None, 148, 148, 32)      0

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        9248

 activation_1 (Activation)   (None, 72, 72, 32)        0

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 activation_2 (Activation)   (None, 34, 34, 64)        0

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 64)        0
 g2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 64)                1183808

 activation_3 (Activation)   (None, 64)                0

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

 activation_4 (Activation)   (None, 1)                 0

=================================================================
Total params: 1212513 (4.63 MB)
Trainable params: 1212513 (4.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Commentaire:

Dans le code precédent et avec l'exécution de ce code on a une représentation d'une architecture de réseau de neurones convolutifs (CNN) en utilisant le framework Keras. Les couches utilisés sont:

Conv2D (couche de convolution):

- Nombre de filtres : 32
- Taille du noyau : (3, 3)
- Fonction d'activation : ReLU

MaxPooling2D (couche de max pooling):

- Taille de la fenêtre de pooling : (2, 2)

Couche de convolution suivante:

- Nombre de filtres : 32
- Taille du noyau : (3, 3)
- Fonction d'activation : ReLU

MaxPooling2D (couche de max pooling):

- Taille de la fenêtre de pooling : (2, 2)

Couche de convolution suivante:

- Nombre de filtres : 64
- Taille du noyau : (3, 3)
- Fonction d'activation : ReLU

MaxPooling2D (couche de max pooling):

- Taille de la fenêtre de pooling : (2, 2)

Flatten (aplatir):Transforme les données en un vecteur 1D avant de passer à la couche dense.

Dense (couche dense):

- Nombre de neurones : 64
- Fonction d'activation : ReLU

Dropout (couche de dropout): Applique une régularisation de type dropout avec un taux de 0.5 (ce qui signifie que 50% de neurones serons exclu pendant l'entraînement).

Dense (couche dense de sortie):

- Un seul neurone, indiquant une tâche de classification binaire.

Fonction d'activation : Sigmoid (typique pour la classification binaire).

Paramètres du modèle:

- Nombre total de paramètres : 1,212,513
- Paramètres entraînables : 1,212,513
- Paramètres non entraînables : 0

Remarques:

Le modèle est destiné à une tâche de classification binaire, comme indiqué par la couche de sortie avec une seule unité et une activation sigmoid. Le dropout est utilisé pour réduire le surajustement pendant l'entraînement. La fonction d'activation ReLU est utilisée dans les couches cachées, sauf pour la couche de sortie qui utilise une activation sigmoid.

```
import psutil

ram_info = psutil.virtual_memory()

print(f"Total RAM: {ram_info.total / (1024 ** 3):.2f} GB")
print(f"Used RAM: {ram_info.used / (1024 ** 3):.2f} GB")
print(f"Free RAM: {ram_info.available / (1024 ** 3):.2f} GB")
```

```
Total RAM: 12.67 GB
Used RAM: 10.24 GB
Free RAM: 2.10 GB
```

Commentaire:

En raison de l'espace limité dans la RAM, il n'est pas possible de charger l'intégralité du jeu de données en mémoire lors de l'entraînement, ce qui est généralement effectué par la fonction model.fit. Pour remédier à cette contrainte, une solution consiste à charger les données directement depuis le disque dur. À chaque itération d'entraînement, un lot (ou batch) de données est chargé, permettant ainsi de travailler avec des ensembles de données volumineux. Bien que cette technique puisse être plus lente que l'utilisation de la fonction model.fit, elle est efficace pour les jeux de données de grande taille, offrant l'avantage d'exploiter l'ensemble des informations présentes dans le jeu de données.

```
from keras.utils import Sequence
import numpy as np

class DataGenerator(Sequence):
    def __init__(self, x_data, y_data, batch_size):
        self.x_data = x_data
        self.y_data = y_data
        self.batch_size = batch_size
        self.indexes = np.arange(len(self.x_data))

    def __len__(self):
        return int(np.ceil(len(self.x_data) / self.batch_size))

    def __getitem__(self, index):
        start = index * self.batch_size
        end = (index + 1) * self.batch_size
        batch_x = self.x_data[start:end]
        batch_y = self.y_data[start:end]

        # You may need to preprocess your data here if required

        return batch_x, batch_y
```

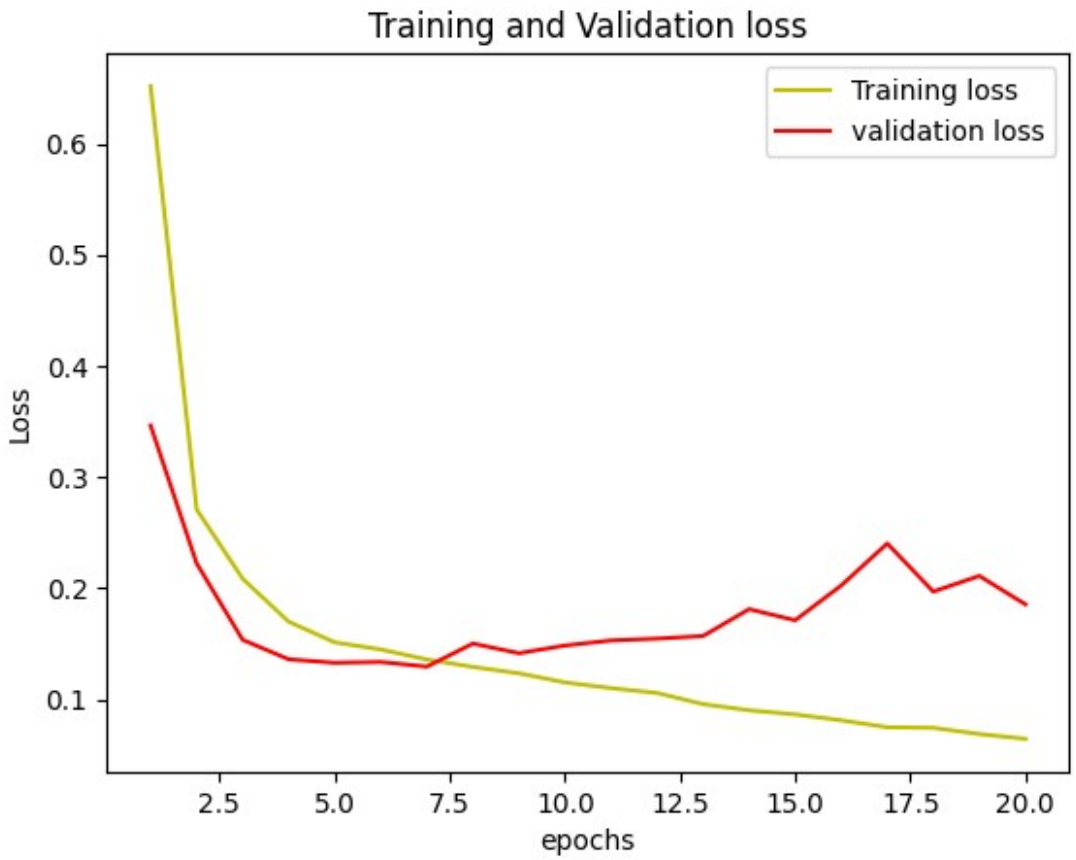
```
# Create an instance of the data generator
batch_size = 64
train_generator = DataGenerator(X_train,y_train, batch_size)
validation_generator = DataGenerator(X_test,y_test, batch_size)

# Use fit_generator instead of fit
History =
model.fit_generator(generator=train_generator,validation_data=validati
on_generator, epochs=20, verbose=1, shuffle=False)
```

```
Epoch 1/20

<ipython-input-46-dd1a3841a92b>:30: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use
`Model.fit`, which supports generators.
  History =
model.fit_generator(generator=train_generator,validation_data=validati
on_generator, epochs=20, verbose=1, shuffle=False)

345/345 [==============================] - 24s 54ms/step - loss:
0.6517 - accuracy: 0.6219 - val_loss: 0.3464 - val_accuracy: 0.8768
Epoch 2/20
345/345 [==============================] - 14s 40ms/step - loss:
0.2710 - accuracy: 0.9103 - val_loss: 0.2224 - val_accuracy: 0.9263
Epoch 3/20
345/345 [==============================] - 14s 41ms/step - loss:
0.2086 - accuracy: 0.9371 - val_loss: 0.1536 - val_accuracy: 0.9483
Epoch 4/20
345/345 [==============================] - 14s 40ms/step - loss:
0.1699 - accuracy: 0.9484 - val_loss: 0.1361 - val_accuracy: 0.9565
Epoch 5/20
345/345 [==============================] - 14s 41ms/step - loss:
0.1513 - accuracy: 0.9542 - val_loss: 0.1330 - val_accuracy: 0.9561
Epoch 6/20
345/345 [==============================] - 14s 41ms/step - loss:
0.1450 - accuracy: 0.9546 - val_loss: 0.1337 - val_accuracy: 0.9557
Epoch 7/20
345/345 [==============================] - 15s 43ms/step - loss:
0.1360 - accuracy: 0.9563 - val_loss: 0.1295 - val_accuracy: 0.9566
Epoch 8/20
345/345 [==============================] - 14s 41ms/step - loss:
0.1293 - accuracy: 0.9593 - val_loss: 0.1503 - val_accuracy: 0.9554
Epoch 9/20
345/345 [==============================] - 14s 41ms/step - loss:
0.1235 - accuracy: 0.9619 - val_loss: 0.1416 - val_accuracy: 0.9575
Epoch 10/20
345/345 [==============================] - 14s 41ms/step - loss:
0.1153 - accuracy: 0.9637 - val_loss: 0.1485 - val_accuracy: 0.9545
Epoch 11/20
```

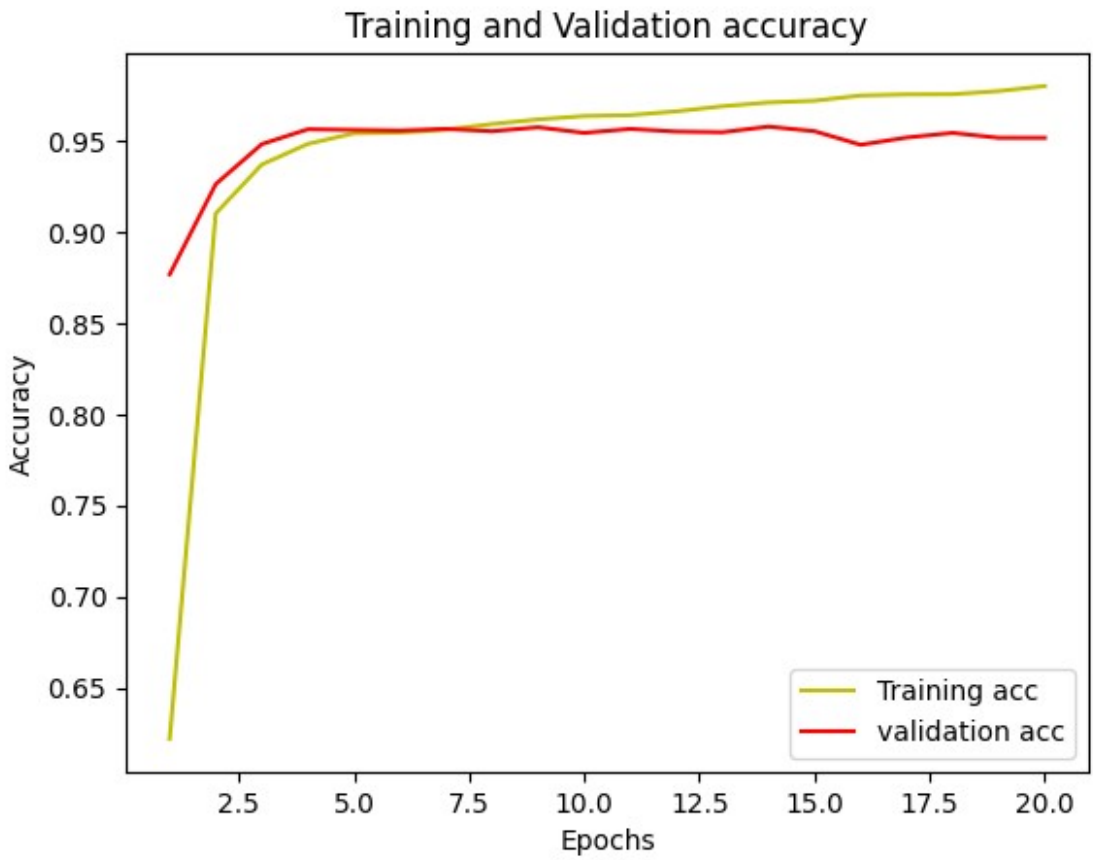
```
345/345 [==============================] - 14s 40ms/step - loss:
0.1102 - accuracy: 0.9642 - val_loss: 0.1531 - val_accuracy: 0.9566
Epoch 12/20
345/345 [==============================] - 16s 47ms/step - loss:
0.1059 - accuracy: 0.9663 - val_loss: 0.1547 - val_accuracy: 0.9552
Epoch 13/20
345/345 [==============================] - 15s 42ms/step - loss:
0.0958 - accuracy: 0.9691 - val_loss: 0.1571 - val_accuracy: 0.9548
Epoch 14/20
345/345 [==============================] - 18s 51ms/step - loss:
0.0903 - accuracy: 0.9712 - val_loss: 0.1813 - val_accuracy: 0.9579
Epoch 15/20
345/345 [==============================] - 14s 41ms/step - loss:
0.0866 - accuracy: 0.9721 - val_loss: 0.1712 - val_accuracy: 0.9554
Epoch 16/20
345/345 [==============================] - 15s 43ms/step - loss:
0.0813 - accuracy: 0.9750 - val_loss: 0.2025 - val_accuracy: 0.9479
Epoch 17/20
345/345 [==============================] - 16s 47ms/step - loss:
0.0751 - accuracy: 0.9756 - val_loss: 0.2403 - val_accuracy: 0.9519
Epoch 18/20
345/345 [==============================] - 14s 42ms/step - loss:
0.0745 - accuracy: 0.9757 - val_loss: 0.1971 - val_accuracy: 0.9545
Epoch 19/20
345/345 [==============================] - 14s 40ms/step - loss:
0.0689 - accuracy: 0.9774 - val_loss: 0.2111 - val_accuracy: 0.9517
Epoch 20/20
345/345 [==============================] - 14s 42ms/step - loss:
0.0646 - accuracy: 0.9802 - val_loss: 0.1856 - val_accuracy: 0.9517
```

Exercice 03: Analyse des performance du modèle

```
loss = History.history["loss"]
val_loss = History.history["val_loss"]
epochs = range(1,len(loss)+1)
plt.plot(epochs,loss,"y", label ="Training loss")
plt.plot(epochs,val_loss,"r", label ="validation loss")
plt.title ("Training and Validation loss")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
acc = History.history["accuracy"]
val_accu = History.history["val_accuracy"]
plt.plot(epochs,acc,"y", label ="Training acc")
plt.plot(epochs,val_accu,"r", label ="validation acc")
plt.title ("Training and Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Commentaire:

Le modèle de réseau de neurones a été entraîné sur 20 époques avec des générateurs de données. La perte d'entraînement diminue progressivement, indiquant une amélioration, mais la perte de validation augmente après quelques époques, suggérant un possible surajustement. L'exactitude d'entraînement atteint 98%, tandis que la précision de validation semble plafonner autour de 95%. Des ajustements, tels que l'ajout de régularisation, pourraient être nécessaires pour améliorer les performances sur les données de validation.

Application de la Regularisaiton:

Pour améliorer les performances de modele on a deux solutions soit on arrete l'entrainement avant l'augmentation de la marge entre la perte de validation et d'entrainement, ou on ajoute une regularisation sur l'architecture de reseau de neurone.

```
INPUT_SHAPE =(SIZE, SIZE, 3)
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape =
INPUT_SHAPE,activation="relu",kernel_regularizer='l2'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3), kernel_initializer =
"he_uniform",activation="relu",kernel_regularizer='l2'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Conv2D(64,(3,3), kernel_initializer =
"he_uniform",activation="relu",kernel_regularizer='l2'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(64,activation="relu",kernel_regularizer='l2'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation("sigmoid"))

model.compile(loss="binary_crossentropy", optimizer = "rmsprop",
metrics="accuracy")

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_3 (MaxPoolin  (None, 74, 74, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 72, 72, 32)        9248

 max_pooling2d_4 (MaxPoolin  (None, 36, 36, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 17, 17, 64)        0
 g2D)

 flatten_1 (Flatten)         (None, 18496)             0

 dense_2 (Dense)             (None, 64)                1183808

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

 activation_5 (Activation)   (None, 1)                 0

=================================================================
Total params: 1212513 (4.63 MB)
Trainable params: 1212513 (4.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Commentaire :

Dans cette partie, la régularisation a été ajoutée à chaque couche du réseau, ce qui va être utilisé pour pénaliser les poids du modèle. Cette approche contribue à prévenir le surapprentissage en limitant les coefficients des paramètres, favorisant ainsi une meilleure généralisation du modèle sur de nouvelles données.

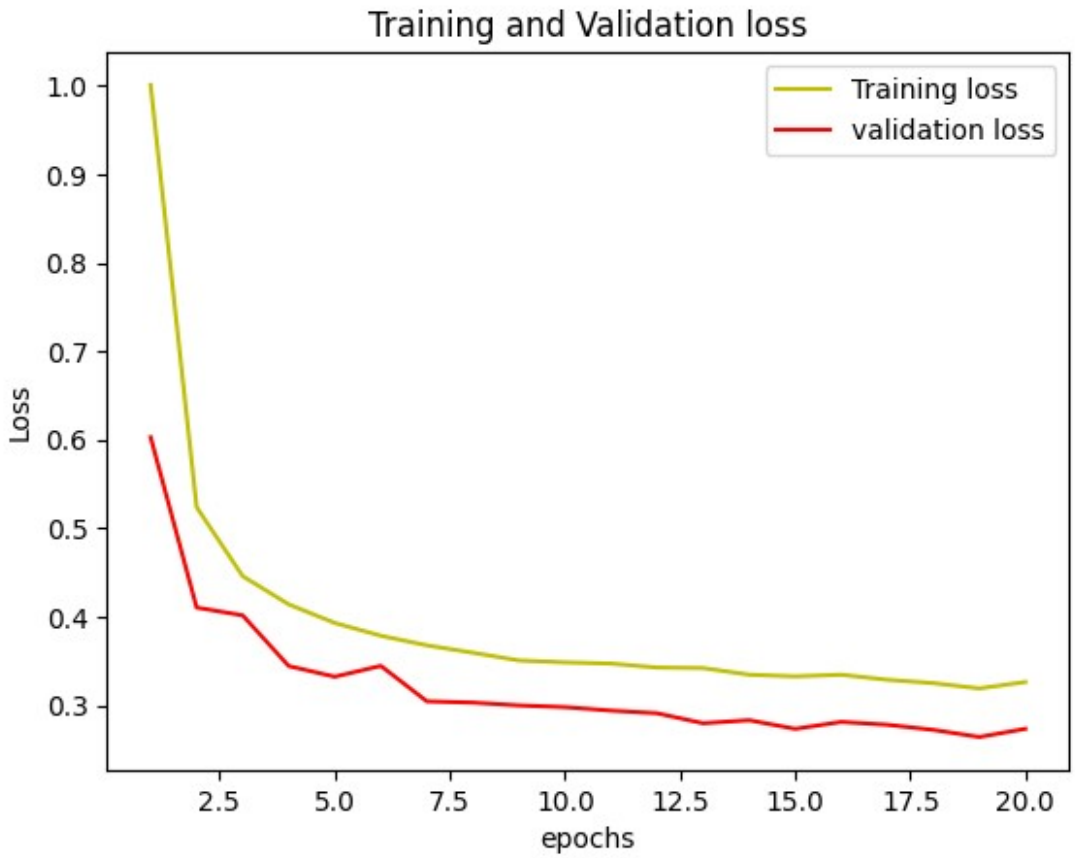
```
from keras.utils import Sequence
import numpy as np

class DataGenerator(Sequence):
    def __init__(self, x_data, y_data, batch_size):
        self.x_data = x_data
        self.y_data = y_data
        self.batch_size = batch_size
        self.indexes = np.arange(len(self.x_data))

    def __len__(self):
        return int(np.ceil(len(self.x_data) / self.batch_size))

    def __getitem__(self, index):
        start = index * self.batch_size
        end = (index + 1) * self.batch_size
        batch_x = self.x_data[start:end]
        batch_y = self.y_data[start:end]

        # You may need to preprocess your data here if required

        return batch_x, batch_y

# Create an instance of the data generator
batch_size = 64
train_generator = DataGenerator(X_train,y_train, batch_size)
validation_generator = DataGenerator(X_test,y_test, batch_size)

# Use fit_generator instead of fit
History =
model.fit_generator(generator=train_generator,validation_data=validati
on_generator, epochs=20, verbose=1, shuffle=False)
```

```
Epoch 1/20

<ipython-input-52-dd1a3841a92b>:30: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use
`Model.fit`, which supports generators.
  History =
model.fit_generator(generator=train_generator,validation_data=validati
on_generator, epochs=20, verbose=1, shuffle=False)

345/345 [==============================] - 20s 55ms/step - loss:
1.0008 - accuracy: 0.6486 - val_loss: 0.6029 - val_accuracy: 0.7661
```

```
Epoch 2/20
345/345 [==============================] - 14s 42ms/step - loss:
0.5238 - accuracy: 0.8697 - val_loss: 0.4105 - val_accuracy: 0.9029
Epoch 3/20
345/345 [==============================] - 13s 39ms/step - loss:
0.4459 - accuracy: 0.9028 - val_loss: 0.4017 - val_accuracy: 0.9098
Epoch 4/20
345/345 [==============================] - 15s 43ms/step - loss:
0.4141 - accuracy: 0.9062 - val_loss: 0.3444 - val_accuracy: 0.9102
Epoch 5/20
345/345 [==============================] - 15s 43ms/step - loss:
0.3931 - accuracy: 0.9099 - val_loss: 0.3324 - val_accuracy: 0.9303
Epoch 6/20
345/345 [==============================] - 17s 50ms/step - loss:
0.3787 - accuracy: 0.9120 - val_loss: 0.3447 - val_accuracy: 0.9058
Epoch 7/20
345/345 [==============================] - 15s 42ms/step - loss:
0.3678 - accuracy: 0.9106 - val_loss: 0.3046 - val_accuracy: 0.9267
Epoch 8/20
345/345 [==============================] - 16s 45ms/step - loss:
0.3594 - accuracy: 0.9123 - val_loss: 0.3032 - val_accuracy: 0.9224
Epoch 9/20
345/345 [==============================] - 16s 45ms/step - loss:
0.3507 - accuracy: 0.9116 - val_loss: 0.3000 - val_accuracy: 0.9332
Epoch 10/20
345/345 [==============================] - 16s 47ms/step - loss:
0.3486 - accuracy: 0.9120 - val_loss: 0.2980 - val_accuracy: 0.9231
Epoch 11/20
345/345 [==============================] - 23s 65ms/step - loss:
0.3472 - accuracy: 0.9149 - val_loss: 0.2941 - val_accuracy: 0.9305
Epoch 12/20
345/345 [==============================] - 18s 53ms/step - loss:
0.3428 - accuracy: 0.9129 - val_loss: 0.2911 - val_accuracy: 0.9214
Epoch 13/20
345/345 [==============================] - 15s 43ms/step - loss:
0.3422 - accuracy: 0.9113 - val_loss: 0.2798 - val_accuracy: 0.9274
Epoch 14/20
345/345 [==============================] - 14s 42ms/step - loss:
0.3346 - accuracy: 0.9122 - val_loss: 0.2833 - val_accuracy: 0.9258
Epoch 15/20
345/345 [==============================] - 14s 41ms/step - loss:
0.3327 - accuracy: 0.9120 - val_loss: 0.2735 - val_accuracy: 0.9309
Epoch 16/20
345/345 [==============================] - 14s 41ms/step - loss:
0.3346 - accuracy: 0.9129 - val_loss: 0.2813 - val_accuracy: 0.9291
Epoch 17/20
345/345 [==============================] - 15s 42ms/step - loss:
0.3289 - accuracy: 0.9155 - val_loss: 0.2781 - val_accuracy: 0.9343
Epoch 18/20
```

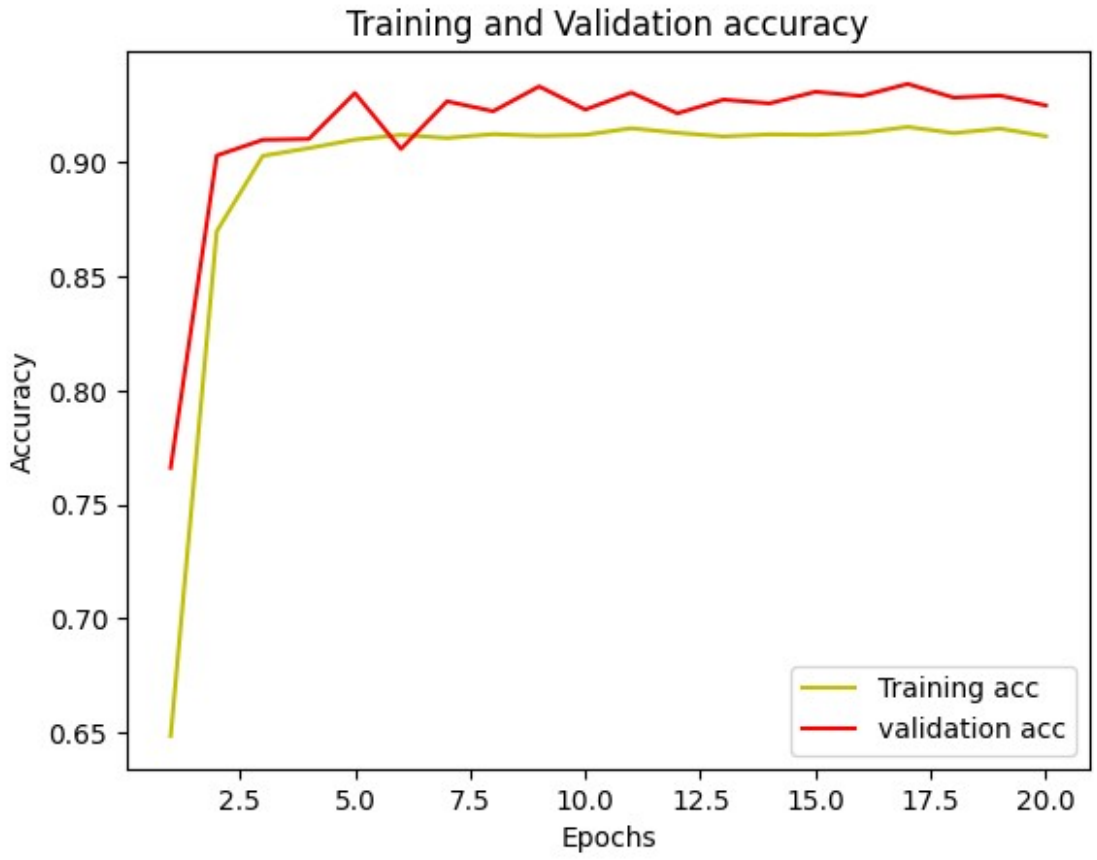
```
345/345 [==============================] - 15s 43ms/step - loss:
0.3251 - accuracy: 0.9128 - val_loss: 0.2723 - val_accuracy: 0.9283
Epoch 19/20
345/345 [==============================] - 15s 42ms/step - loss:
0.3191 - accuracy: 0.9148 - val_loss: 0.2643 - val_accuracy: 0.9292
Epoch 20/20
345/345 [==============================] - 14s 41ms/step - loss:
0.3264 - accuracy: 0.9114 - val_loss: 0.2735 - val_accuracy: 0.9249
```

```
loss = History.history["loss"]
val_loss = History.history["val_loss"]
epochs = range(1,len(loss)+1)
plt.plot(epochs,loss,"y", label ="Training loss")
plt.plot(epochs,val_loss,"r", label ="validation loss")
plt.title ("Training and Validation loss")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
acc = History.history["accuracy"]
val_accu = History.history["val_accuracy"]
```

```
plt.plot(epochs,acc,"y", label ="Training acc")
plt.plot(epochs,val_accu,"r", label ="validation acc")
plt.title ("Training and Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

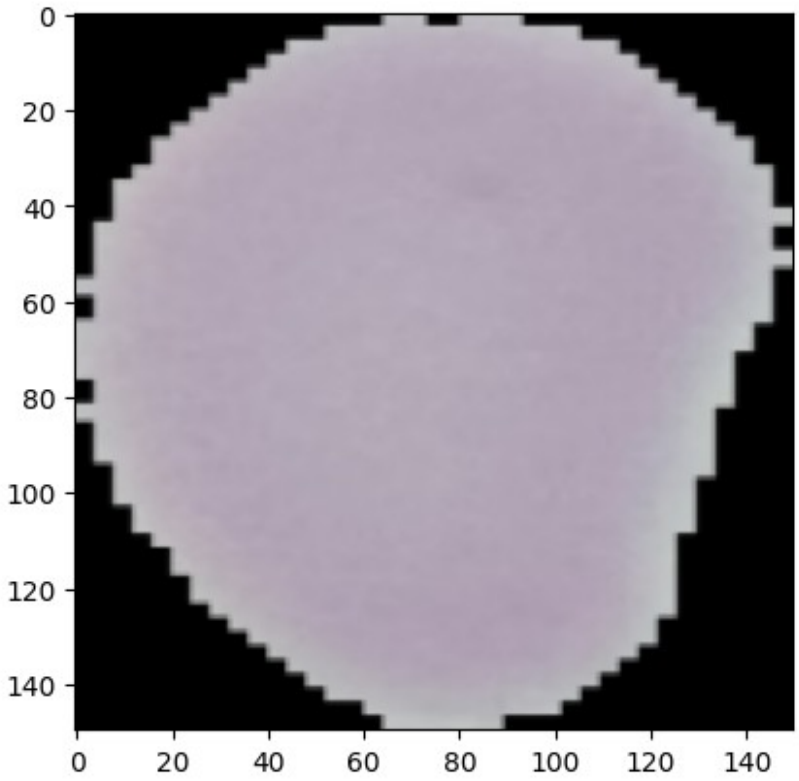


Commentaire :

Après l'ajout de la régularisation dans les couches du réseau de neurone, on remarque qu'on a pu éviter le surapprentissage (overfitting). Les performances d'entraînement et de validation sont désormais plus équilibrées, indiquant une meilleure généralisation du modèle sur de nouvelles images.

```
n =random.randint(0,len(X_test)-1)
img = X_test[n]
plt.imshow((img*255).astype("uint8"))
input_img = np.expand_dims(img,axis=0)
print(" Label de cette image : ", y_test[n])
print("Prédiction de cette image :", model.predict(input_img))
print("Le label de cette image est : ", y_test[n])
```

```
 Label de cette image :  0
1/1 [==============================] - 0s 395ms/step
Prédiction de cette image : [[0.051792]]
Le label de cette image est :  0
```



Commentaire :

On remarque que notre modèle entraîné a correctement classifié l'image. Avec une valeur prédicté de 0.05 et une étiquette de 1, l'arrondissement indique que le modèle prédit la classe 0 avec confiance.

```
model.save("MODEL")
!cp -r MODEL/ "/content/drive/MyDrive/"
```

Commentaire: Avec ces deux lignes de code, on a sauvgarder notre model dans un repertoire au Drive.

```
from tensorflow.keras.models import load_model
model = load_model("MODEL")
```

Commentaire: Avec ces deux lignes de code, on a telecharger notre model entrainé.

```
del dataset, label, X_train, y_train
```

Commentaire: On supprime les variables qui ne seront pas utilisées ultérieurement afin de libérer la mémoire.

```
acc = model.evaluate(X_test, y_test)
```

```
173/173 [==============================] - 2s 12ms/step - loss: 0.2735
- accuracy: 0.9249
```

```
print("Accuracy =", (acc[1]*100.0), "%")
```

```
Accuracy = 92.48911738395691 %
```

Commentaire: On a une précision de model qui égale à 92.48%

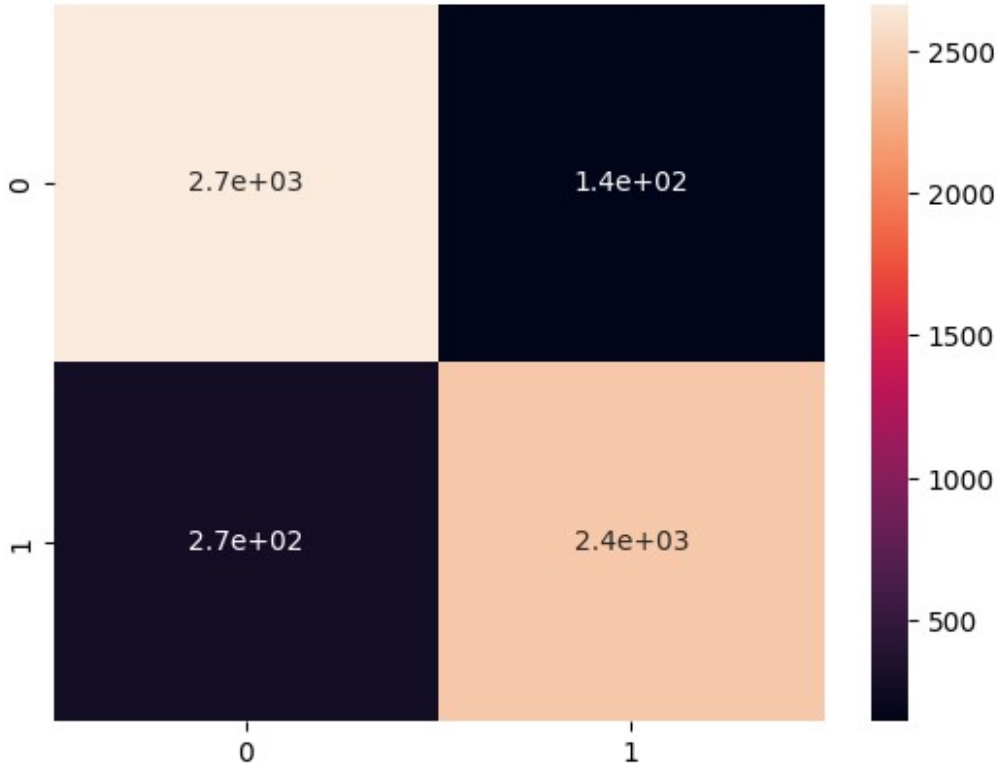
Exercice 04: Analyse de la matrice de confusion

```
threshold = 0.5
from sklearn.metrics import confusion_matrix
import seaborn as sns
y_pred = (model.predict(X_test)>=threshold).astype(int)
```

```
173/173 [==============================] - 2s 8ms/step
```

```
cm = confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
```

```
<Axes: >
```

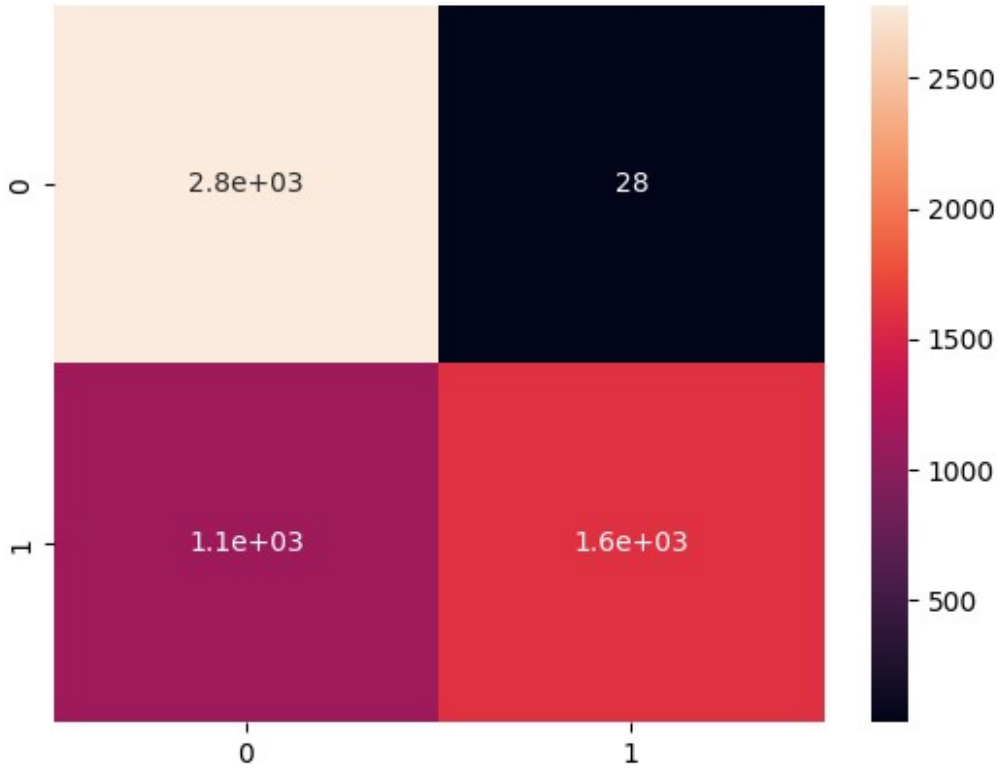


Commentaire : D'après les résultats obtenus, la matrice de confusion présente les vraies prédictions sur la diagonale. On peut observer que notre modèle est bien entraîné, car les grandes valeurs se trouvent en diagonale. On peut utilisé cette matrice pour calculer la sensibilité et la précision et d'autres parametres.

```
threshold = 0.9
from sklearn.metrics import confusion_matrix
import seaborn as sns
y_pred = (model.predict(X_test)>=threshold).astype(int)
cm = confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
```

```
173/173 [==============================] - 2s 9ms/step
```

```
<Axes: >
```



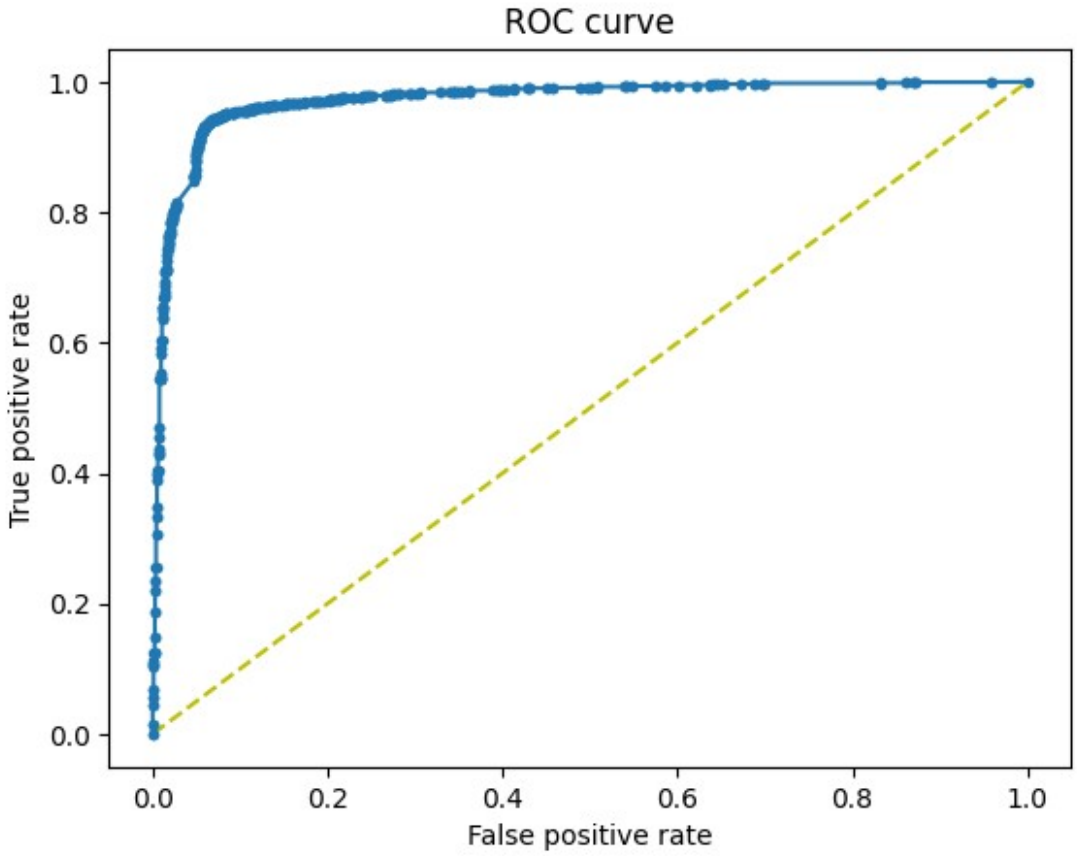
Commebtaire:

Après avoir modifié la valeur du seuil, on remarque que notre modèle confond les vrais "un" en les prédisant comme des "zéros" car notre seuil est très élevé. De même, la même situation se produirait pour les vrais "zéros" si l'on choisissait une valeur de seuil de classification trop basse.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
y_preds = model.predict(X_test).ravel()
fpr, tpr, thresold = roc_curve(y_test, y_preds)
```

```
plt.figure(1)
plt.plot([0,1],[0,1],"y--")
plt.plot(fpr,tpr,marker=".")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("ROC curve")
plt.show()
```

```
173/173 [==============================] - 4s 9ms/step
```



Commentaire:

roc_curve(y_test, y_preds): calcule les taux de faux positifs (fpr), les taux de vrais positifs (tpr) et les seuils (thresholds) pour différents points de seuil de classification.

fpr: représente le taux de faux positifs, c'est-à-dire la proportion de vrais négatifs incorrectement classifiés en tant que positifs.

tpr: représente le taux de vrais positifs, indiquant la proportion de vrais positifs correctement classifiés en tant que positifs.

thresholds contient les seuils de classification associés à chaque paire de taux de faux positifs et de taux de vrais positifs.

Ce script permet de visualiser la performance d'un modèle de classification binaire en représentant graphiquement la relation entre les taux de faux positifs et les taux de vrais positifs pour différents seuils de classification. Cette courbe est utile pour évaluer la capacité du modèle à discriminer entre les classes. Dans notre graphe on peut voire que le choix de la seuil

Conclision:

Ce TP a débuté par une exploration de la préparation des données et la manière d'augmenter les données lorsque celles-ci sont limitées. Dans la deuxième partie, On a travaillé sur l'apprentissage d'un modèle CNN pour la détection des infections à partir d'images. On a résolu des problèmes liés à la saturation de mémoire et on a également traité le surapprentissage en introduisant des paramètres de régularisation. Enfin, on a évalué notre modèle et examiné l'impact du point de seuil sur la précision des prédictions de notre modèle.