BASKARAN

Amalan

N°12001105

19/11/2023

--TP 6 Réseau de neurones convolutifs pour la classification binaire d'images--

Entrée [116]:

```
from keras.preprocessing.image import ImageDataGenerator # les Libraries Necessaires
from skimage import io
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
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
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import roc_curve
```

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

Entrée [117]:

```
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)
```

Entrée [118]:

```
x =io.imread("/content/monalisa.jpg")
x = x.reshape((1,) + x.shape)
```

Entrée [119]:

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

Augmentation de la taille du jeu de données d'entrainement qui consiste a crées des données a aprtir d'une données en referance. ImageDataGenerator fournit plusieurs paramètres pour contrôler le processus d'augmentation comme

- rotation_range : rotations aléatoires à appliquer aux images.
- width_shift_range et height_shift_range :la plage de décalages horizontaux et verticaux aléatoires à appliquer aux images
- horizontal_flip : s'il faut inverser les images horizontalement

A partir d'image monalisa, nous créerons des nouvelles images qui sont des modification d'image de monalisa comma par exemple monalisa décalé ou inversé. L'image généré sera stocké dans le dossier augmented

Exercice 1 : Lecture et labellisation du jeu de données

Entrée [120]:

```
#!unzip /content/drive/MyDrive/cell_test2.zip
```

Entrée [121]:

```
image_directory = "/content/cell_test2/" # Ici nous utiliserons 2000 données de Parasitized et 2000 Uninfected
```

Entrée [122]:

```
SIZE = 150 # taille d'image
dataset = [] # les images comme les features
label = [] # 1 ou 0, si il est parasitized ou Uninfected
parasitized_images = os.listdir(image_directory + 'Parasitized/') # le dossier qui contient images Parasitized
for i, image_name in enumerate(parasitized_images) : # on regard chaque element
      if (image_name.split('.')[1] == 'png') : # uniquement les images png
        image= cv2.imread(image_directory + 'Parasitized/' +image_name) # lecture d'images
        image = Image.fromarray(image,'RGB') # array to image
        image = image.resize((SIZE,SIZE))
        dataset.append(np.array(image)) # ajouter dans les feautures
        label.append(1) # ajout label qui est 1

parasitized_images = os.listdir(image_directory + 'Uninfected/')  # le dossier qui contient images Uninfected
for i, image_name in enumerate(parasitized_images) :
      if (image_name.split('.')[1] == 'png') :
        image= cv2.imread(image_directory + 'Uninfected/' +image_name)
        image = Image.fromarray(image,'RGB')
        image = image.resize((SIZE,SIZE))
        dataset.append(np.array(image))
        label.append(0)
```

Entrée [123]:

```
dataset = np.array(dataset) # array to numpy array
label = np.array(label)
```

Entrée [124]:

```
dataset.shape
```

Out[124]:

```
(4000, 150, 150, 3)
```

4000 élements de 150x150 avec 3 couches R,G et B

Entrée [125]:

```
label.shape
```

Out[125]:

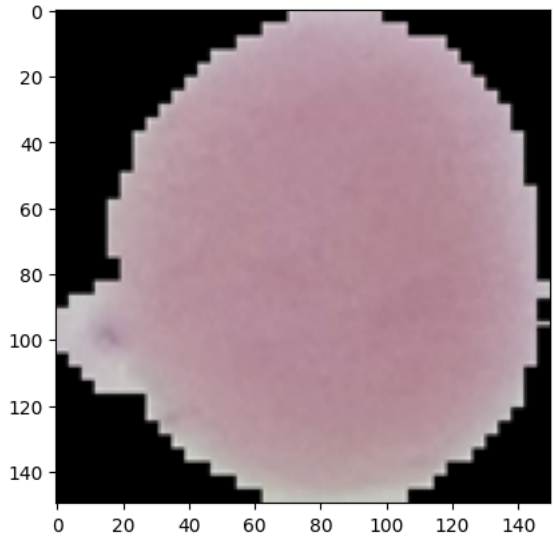
```
(4000,)
```

Zero ou un pour chaque dataset

Entrée [126]:

```
import random
import numpy as np
image_number =random.randint(0,len(dataset)-1) # choisir un chiffre entre 0 et 3999
plt.imshow(np.reshape(dataset[image_number],(150,150,3))) # afficher
print("Label de cette image : ", label[image_number])
```

```
Label de cette image :  0
```



On peut voir que les images infecté possedent une ou plusieurs taches violet et non infecté ne possede pas de tache.

Exercice 2 : Construction et entrainement du modèle

Entrée [127]:

```
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)
```

découpe des données

Entrée [128]:

```
from keras.utils import normalize
X_train = normalize(X_train, axis =1) # on normalise, pour éviter les differance d'amplitude
X_test = normalize(X_test, axis =1)
```

Entrée [129]:

```
INPUT_SHAPE =(SIZE, SIZE, 3)
model = Sequential() # création d'un modele neutral networks
model.add(Conv2D(32,(3,3),input_shape = INPUT_SHAPE)) # ajout de couche convolution 32 filtre de 3X3
model.add(Activation('relu'))# activation relu
model.add(MaxPooling2D(pool_size=(2,2))) #réduire la dimensionnalité de l'image
model.add(Conv2D(32,(3,3), kernel_initializer = 'he_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2))) #réduire la dimensionnalité de l'image
model.add(Conv2D(64,(3,3), kernel_initializer = 'he_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2))) #réduire la dimensionnalité de l'image
model.add(Flatten()) #convertir en une dimention
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5)) # permet d'eviter sur-apprentissage
model.add(Dense(1)) # prédiction finale du modèle
model.add(Activation('sigmoid'))
```

Entrée [130]:

```
model.compile(loss='binary_crossentropy', optimizer = 'rmsprop', metrics=['accuracy']) #compilation du modele
```

Entrée [131]:

```
print(model.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 148, 148, 32)      896

 activation_20 (Activation)  (None, 148, 148, 32)      0

 max_pooling2d_12 (MaxPooli  (None, 74, 74, 32)        0
 ng2D)

 conv2d_13 (Conv2D)          (None, 72, 72, 32)        9248

 activation_21 (Activation)  (None, 72, 72, 32)        0

 max_pooling2d_13 (MaxPooli  (None, 36, 36, 32)        0
 ng2D)

 conv2d_14 (Conv2D)          (None, 34, 34, 64)        18496

 activation_22 (Activation)  (None, 34, 34, 64)        0

 max_pooling2d_14 (MaxPooli  (None, 17, 17, 64)        0
 ng2D)

 flatten_4 (Flatten)         (None, 18496)             0

 dense_8 (Dense)             (None, 64)                1183808

 activation_23 (Activation)  (None, 64)                0

 dropout_4 (Dropout)         (None, 64)                0

 dense_9 (Dense)             (None, 1)                 65

 activation_24 (Activation)  (None, 1)                 0

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

Un résumé de notre model compilé avec les different details concernant les couches et les activation utlisé.

Lorsqu'on compile un modele, on passe par plusieurs étapes :

- la fonction loss permettre de mesurer de l'erreur entre les prédictions du modèle et les valeurs réelles
- on ajoute un optimiseur qui sera responsable de la mise à jour des poids du modèle afin de minimiser la fonction de perte
- des métriques sont utilisées pour évaluer les performances du modèle pendant l'entraînement et l'évaluation une fois ces parametres sont fixé, on aura une modele.

Entrée [132]:

```
history = model.fit(X_train, y_train, batch_size=64, verbose = 1, epochs =20, validation_data =(X_test, y_test), shuffle = False)
```

```
50/50 [==============================] - 4s 60ms/step - loss: 0.6918 - accuracy: 0.5366 - val_loss: 0.6893 - val_accurac
y: 0.5213
Epoch 2/20
50/50 [==============================] - 2s 42ms/step - loss: 0.6817 - accuracy: 0.5653 - val_loss: 0.6828 - val_accurac
y: 0.5412
Epoch 3/20
50/50 [==============================] - 2s 39ms/step - loss: 0.6728 - accuracy: 0.5803 - val_loss: 0.6815 - val_accurac
y: 0.5362
Epoch 4/20
50/50 [==============================] - 2s 39ms/step - loss: 0.6642 - accuracy: 0.5950 - val_loss: 0.6775 - val_accurac
y: 0.5337
Epoch 5/20
50/50 [==============================] - 2s 36ms/step - loss: 0.6355 - accuracy: 0.6253 - val_loss: 0.6314 - val_accurac
y: 0.6587
Epoch 6/20
50/50 [==============================] - 2s 36ms/step - loss: 0.6081 - accuracy: 0.6697 - val_loss: 0.5563 - val_accurac
y: 0.7038
Epoch 7/20
50/50 [==============================] - 2s 36ms/step - loss: 0.5135 - accuracy: 0.7609 - val_loss: 0.3377 - val_accurac
```

Exercice 3 : Analyse des performances du modèle

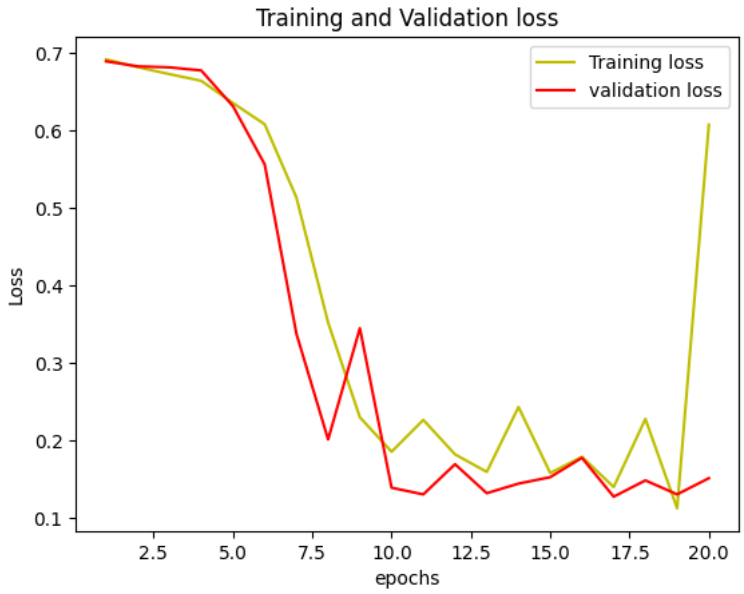
Entrée [133]:

```
print(history.history.keys()) # list all data in history
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Entrée [134]:

```
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()
```



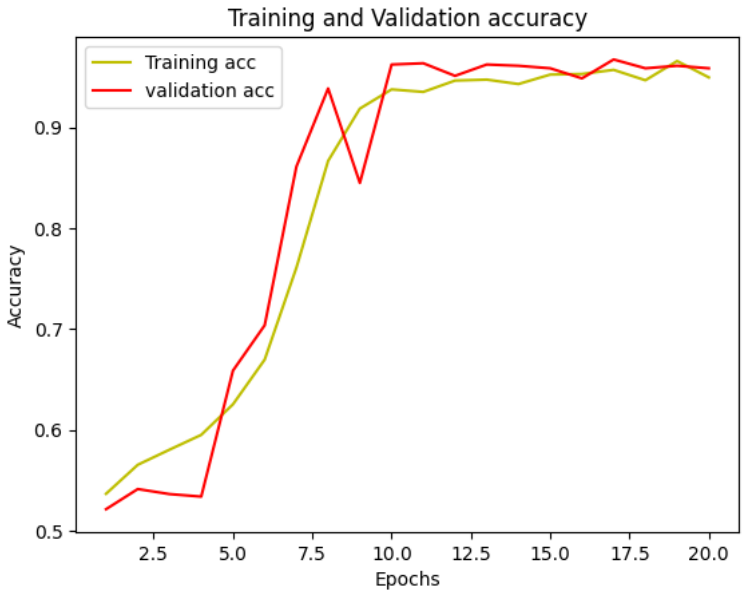
Nous effectuons plusieurs epochs(itérations). À chaque epochs, le modèle est entraîné sur les données d'entraînement et les prédictions sont comparées aux labels. La différence entre les prédictions et les valeurs attendues qui quantifie l'erreur du modèle.

La courbe de loss montre comment la perte sur l'ensemble d'entraînement diminue au fur et à mesure que le modèle apprend. Plus le modèle s'améliore, plus la perte diminue donc il faudra au moins 10 epochs pour que le modele stablise. L'objectif de l'entraînement est de minimiser cette perte jusqu'à ce qu'elle atteigne un minimum.

on note aussi que l'erreur d'entainement augmente apres 19 épochs donc il faudra choisir un épochs entre 10 et 19.

Entrée [135]:

```
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()
```



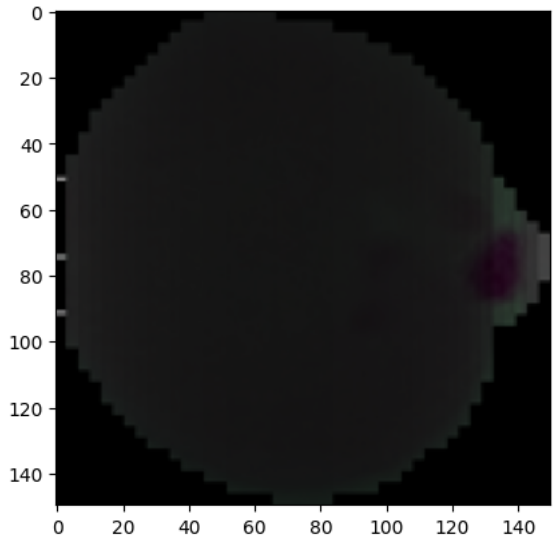
La courbe de Validation accuracy qui indique comment la performance du modèle varie au cours des epochs. Une courbe proche de 1 signifie que le modèle est très performant.

Il arrive parfois que la courbe de validation soit supérieure à celle d'entraînement, ce qui suggère que le modèle a fait du sur-apprentissage.

Entrée [136]:

```
n =random.randint(0,len(X_test)-1)
img = X_test[n]
plt.imshow(img)
input_img = np.expand_dims(img,axis=0)
print('Label de cette image : ', Y_test[image_number]) # label concernant l'image
print('Prédiction de cette image :', model.predict(input_img)) # prediction
print('Le label de cette image est : ', model.predict(input_img))
```

```
Label de cette image :  1
1/1 [==============================] - 0s 84ms/step
Prédiction de cette image : [[0.9978441]]
1/1 [==============================] - 0s 20ms/step
Le label de cette image est :  [[0.9978441]]
```



Nous avons sélectionné au hasard une image dans x_test, qui n'a pas été utilisée pour entraîner le modèle, et nous lui avons demandé de prédire si elle correspond à un cas infecté ou non.

L'image a un label initial de 1, ce qui signifie qu'elle provient d'un patient infecté, et on peut remarquer une tache anormale sur l'image. Notre modèle a correctement identifié cette caractéristique et a attribué une probabilité de 99,7% à l'hypothèse d'une infection.

Entrée [137]:

```
acc = model.evaluate(X_test, y_test)
print('Accuracy=', (acc*100), '%')
```

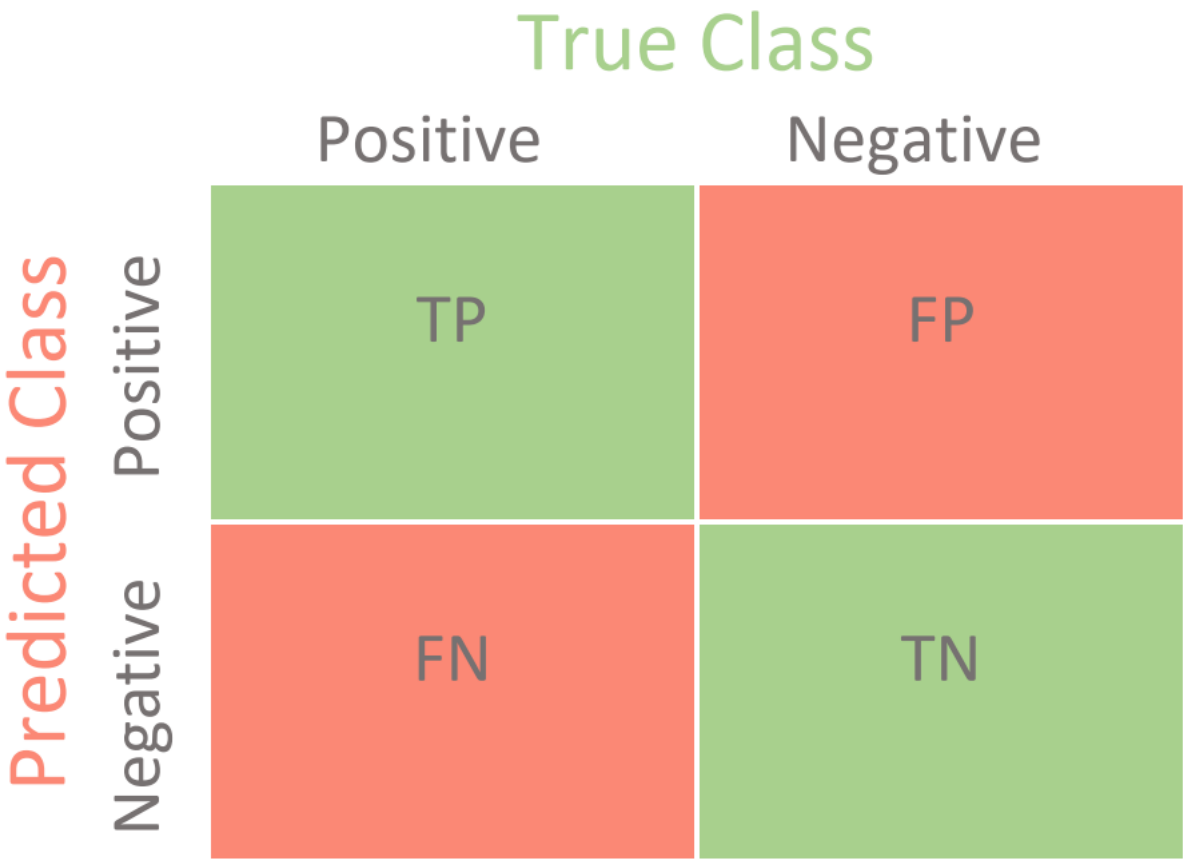
```
25/25 [==============================] - 0s 8ms/step - loss: 0.1511 - accuracy: 0.9588
Accuracy= [0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095
367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367
432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.958750009536743
2, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432,
0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1
511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511
293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293
351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351
650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650
238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.151129335165023
8, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238,
0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9
587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587
500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.958750
095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095
367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367
432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.958750009536743
2, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432,
0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1
511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511
293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293
351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351
650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650
238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.151129335165023
8, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238,
0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9
587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587
500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.958750
095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095
367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367
432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.958750009536743
2, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432,
0.1511293351650238, 0.9587500095367432, 0.1511293351650238, 0.9587500095367432] %
```

Pour tester la qualité d'un modèle, on utilise la fonction model.evaluate() qui prend X_test et Y_test (les données de test et les labels associé), et renvoie une

liste de métriques comme accuracy ou loss. Ces métriques permettent de mesurer les performances du modèle sur les données de test.

Exercice 4 : Analyse de la matrice de confusion

Dans cette partie, nous allons analyser la modèle avec une matrice de confusion. Comme on sait depuis plusieurs TP, la fonctionnement de matrice de confusion :



Entrée [138]:

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

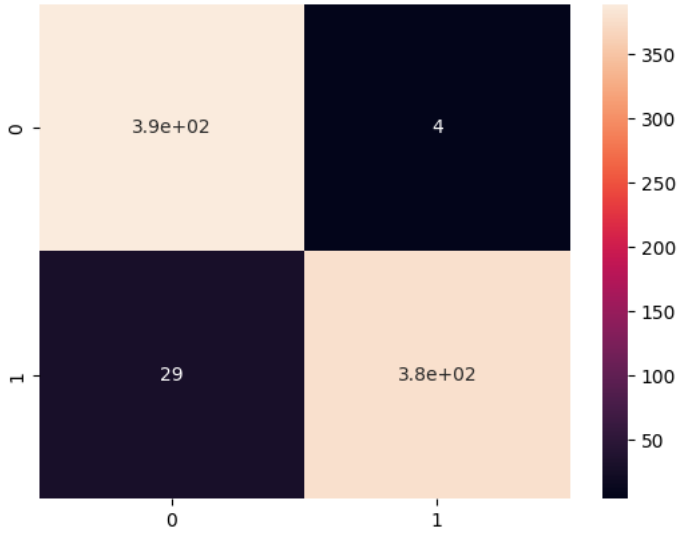
```
25/25 [==============================] - 0s 8ms/step
```

Entrée [139]:

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

Out[139]:

<Axes: >

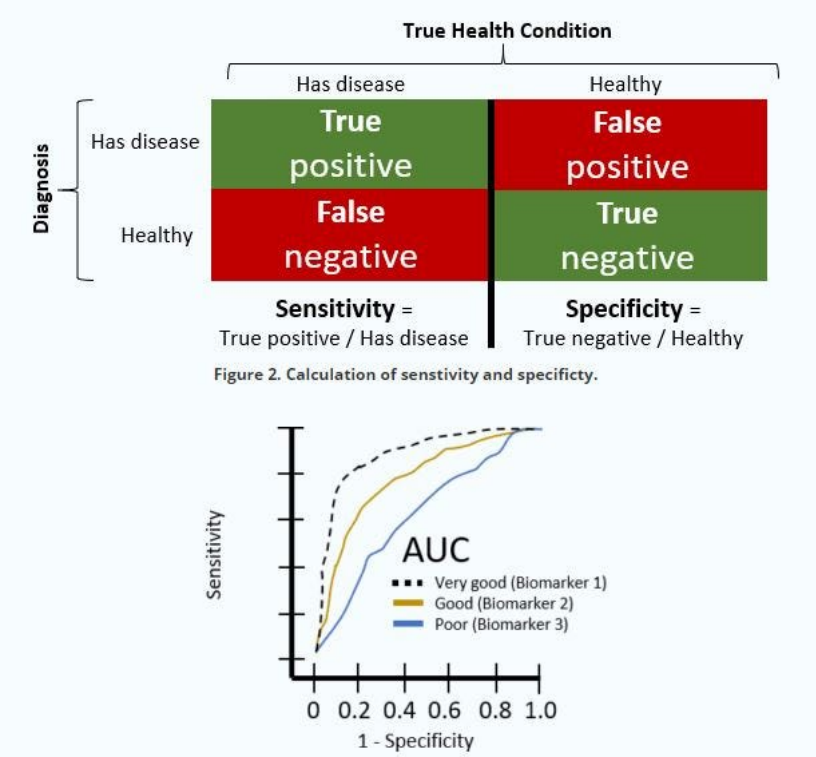


Sur les 20% des données totales (4000x0,20 = 800).Sur un échantillon de 800 données testées, notre modèle a correctement classé 767 données et en a mal classé 33. Le taux d'erreur est donc de 4%, ce qui est assez faible. Nous pouvons en conclure que notre modèle est bien entraîné et performant.

ROC curve va compléter notre analyse avec la mtrice de confusion

```
ROC curve utlise deux indice qu'on peut retrouver sur la matrice de confusion
- Spécificité en abscisse qui se calcule comme (VRAI POSITIF / CAS non infecté) [taux de faux positifs]
- Sensibilité en ordonnées qui se calcule comme (VRAI NEGATIVE / CAS  infecté) [taux de vrais positifs]
```

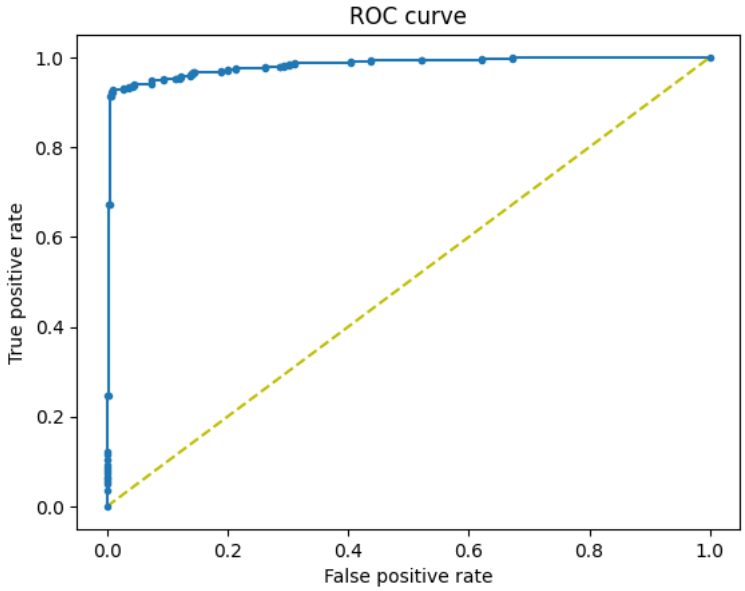
voici le plus clairement :



Entrée [140]:

```
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()
```

```
25/25 [==============================] - 0s 7ms/step
```



D'apres le graphique d'haut dessous, on note que notre modele est tres performant.

Conclusion :

On peut en conclure que les réseaux de neurones sont des outils puissants pour analyser et traiter des données médicales complexes. Durant ce TP nous avons appris comment utiliser un réseau de neurones pour une application type avec les images RGB.