Module - Application Clinique en Radiodiagnostic -M2

December 26, 2024

Réalisé par : **Zakaria Taoubi**

Vous pouvez trouver plus d'informations sur mon travail sur mon GitHub :

[1]:
```
from IPython.display import Image, display
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('Taoubi_Zakaria.png')
fig, ax = plt.subplots(figsize=(2, 2))
ax.imshow(img)
ax.axis('off')
plt.subplots_adjust(left=0.5 - 0.25, right=2, top=0.5 + 0.25, bottom=0.5 - 0.25)
plt.show()
```

1 Introduction

Dans le cadre de mon master en physique médicale à l'ISSS de Settat. **Dans le cadre du module de cours intitulé Application clinique en radiodiagnostics**, j'ai développé un modèle d'intelligence artificielle (IA) pour aider à la détection de tumeurs sur des images médicales. L'IA, et en particulier le deep learning, offre un potentiel considérable pour améliorer l'efficacité du diagnostic en radiodiagnostics, en aidant les professionnels de santé à analyser plus rapidement et avec plus de précision les images médicales.

Il s'agit d'un mini-projet que j'ai développé sur Jupyter Notebook, et qui repose sur la **classification des tumeurs cérébrales**. Pour entraîner le modèle, j'ai utilisé un IRM dataset disponible sur Kaggle. Ce mini-projet a été réalisé dans le cadre de mon module **Application clinique en radiodiagnostic** sous la direction de mon professeur **Pr. Halimi** et **Pr. Walid ALLIOUI**.

1.1 Dataset : Classification IRM des tumeurs cérébrales

J'ai trouvé un dataset sur Kaggle qui contient des images IRM pour la classification des tumeurs cérébrales. Ce dataset est intéressant pour des projets d'apprentissage automatique en imagerie médicale, surtout pour détecter la présence ou l'absence de tumeurs dans des scans IRM.

1.1.1 Source des données

- **Plateforme** : Kaggle
- **Dataset** : IRM de tumeurs cérébrales

1.1.2 Description du dataset

- **Nombre total d'images** : 253
- **Classes** :
  - **Sans Tumeur** : 98 images
  - **Avec Tumeur** : 155 images (cette catégorie inclure différents types de tumeurs cérébrales)
  - **Dimensions des images** : Les dimensions des images varient, donc il est possible que je doive les redimensionner pour les rendre compatibles avec le modèle.

Objectif du projet :

L'objectif principal de ce projet était de créer un modèle capable de classer des images médicales en déterminant si elles montrent une tumeur ou non. Pour ce faire, j'ai utilisé le modèle EfficientNetB3, une architecture de réseau de neurones convolutifs (CNN) qui est bien adaptée à la classification d'images grâce à son efficacité et à ses bonnes performances.

Méthodologie :

Le processus de développement a été divisé en plusieurs étapes :

- **Collecte des données** : J'ai utilisé un ensemble d'images médicales correspondant à des cas de tumeurs et de non-tumeurs.
- **Prétraitement des données** : Les images ont été normalisées et redimensionnées pour être compatibles avec le modèle.
- **Entraînement et test du modèle** : J'ai entraîné le modèle EfficientNetB3 sur l'ensemble de données pendant **30 époques**. Bien que cet entraînement ait été relativement court, il a permis d'obtenir une première évaluation des performances du modèle.

1.2 Tumeur cérébrale

Une tumeur cérébrale est une masse de cellules anormales qui se développe dans ou autour du cerveau. Les tumeurs cérébrales peuvent être bénignes (non cancéreuses) ou malignes (cancéreuses) et, selon leur taille et leur emplacement, elles peuvent causer des symptômes graves comme des maux de tête, des troubles neurologiques, et même menacer la vie du patient. Ce projet fait partie de mon module d'application clinique en radiodiagnostic, dans lequel je cherche à développer une solution de classification automatique des tumeurs à partir d'images IRM, en utilisant des techniques d'apprentissage automatique et d'intelligence artificielle (IA).

[11]:
```
from IPython.display import display, Image, Markdown
image_path = "images.jpeg"
display(Image(filename=image_path))
caption = "Figure 1: tumeur cérébrale"
display(Markdown(f"**{caption}**"))
```

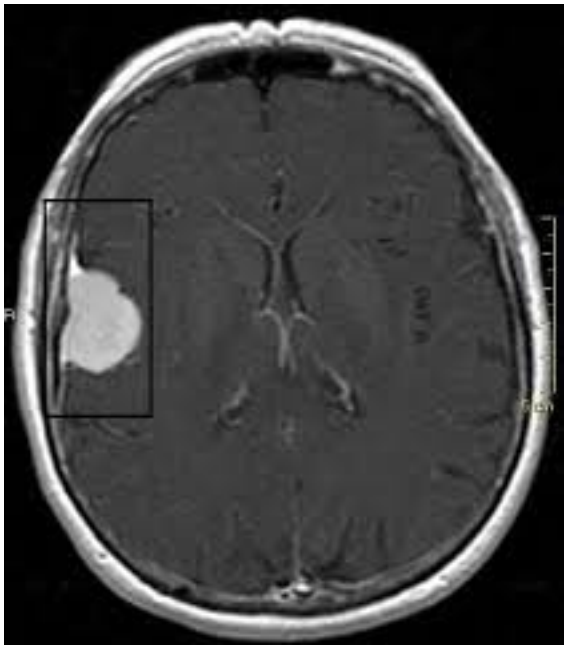


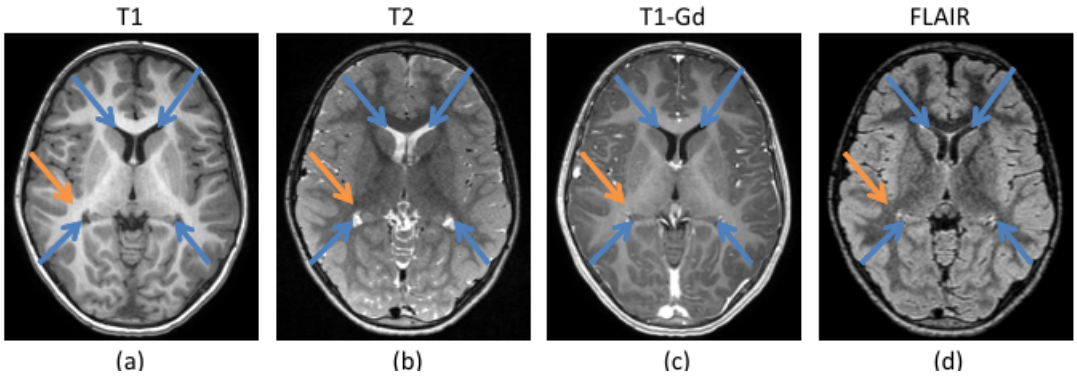
Figure 1: tumeur cérébrale

1.2.1 Imagerie par Résonance Magnétique (IRM)

L'imagerie par résonance magnétique (IRM) est une technique d'imagerie médicale non invasive permettant de produire des images anatomiques et fonctionnelles détaillées du corps humain. Cette technologie repose sur les propriétés de résonance magnétique nucléaire (RMN) des atomes d'hydrogène, très présents dans le corps humain. Lorsqu'ils sont exposés à un champ magnétique intense (B), les noyaux d'hydrogène s'alignent avec ce champ et absorbent des impulsions de radiofréquences (RF), modifiant temporairement leur état magnétique. Après l'arrêt des impulsions RF, ces noyaux reviennent progressivement à leur état initial en émettant un signal mesurable, utilisé pour générer des images. Différentes séquences IRM, telles que T Un(a), T Deux (b) et FLAIR(d) , permettent de distinguer les tissus sains des anomalies comme les tumeurs ou l'œdème. Par exemple, les images pondérées en T Un mettent en avant les structures anatomiques, tandis que les séquences FLAIR suppriment le signal du liquide céphalorachidien pour mieux visualiser les lésions. De plus, l'injection de gadolinium, un produit de contraste, accentue le signal des tumeurs, facilitant ainsi le diagnostic et la planification des traitements. **L'exemple ci-dessous illustre différentes séquences IRM d'une tumeur cérébrale** :

[5]:
```
from IPython.display import display, Image
image_path = "ImagesT.png"
```

```
display(Image(filename=image_path))
```



1.3 Importance de la détection précoce des tumeurs cérébrales

La détection précoce des tumeurs cérébrales est essentielle pour améliorer le pronostic et offrir des traitements moins invasifs. Plus la tumeur est identifiée tôt, plus il est possible de limiter les dommages causés aux tissus cérébraux environnants. Une détection rapide permet aussi d'élaborer des plans de traitement plus précis, qui augmentent les chances de réussite thérapeutique et de qualité de vie pour le patient.

1.4 L'apport de l'IA dans les applications cliniques en radiodiagnostic

L'IA, et en particulier les modèles d'apprentissage profond, a le potentiel de révolutionner le diagnostic des tumeurs cérébrales en radiodiagnostic. En analysant de grandes quantités d'images médicales (IRM, échographies, images TEP), les algorithmes d'IA peuvent repérer des signes de tumeurs difficiles à détecter par l'œil humain, ce qui améliore la précision et la rapidité du diagnostic.

Dans ce projet, j'utilise des images IRM, mais les techniques de détection assistée par l'IA peuvent également être appliquées aux échographies et aux images TEP (Tomographie par Émission de Positons) pour fournir une analyse plus complète et multi-modale. L'objectif est de démontrer **comment une approche basée sur l'IA peut soutenir les radiologues en facilitant un diagnostic précoce, fiable et reproductible dans le contexte clinique**.

[3]:
```
!pip install opendatasets
```

```
Requirement already satisfied: opendatasets in d:\anaconda\lib\site-packages
(0.1.22)
Requirement already satisfied: tqdm in d:\anaconda\lib\site-packages (from
opendatasets) (4.65.0)
Requirement already satisfied: kaggle in d:\anaconda\lib\site-packages (from
opendatasets) (1.6.17)
Requirement already satisfied: click in d:\anaconda\lib\site-packages (from
opendatasets) (8.0.4)
Requirement already satisfied: colorama in d:\anaconda\lib\site-packages (from
```

```
click->opendatasets) (0.4.6)
Requirement already satisfied: six>=1.10 in d:\anaconda\lib\site-packages (from
kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in d:\anaconda\lib\site-
packages (from kaggle->opendatasets) (2023.11.17)
Requirement already satisfied: python-dateutil in d:\anaconda\lib\site-packages
(from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in d:\anaconda\lib\site-packages (from
kaggle->opendatasets) (2.31.0)
Requirement already satisfied: python-slugify in d:\anaconda\lib\site-packages
(from kaggle->opendatasets) (5.0.2)
Requirement already satisfied: urllib3 in d:\anaconda\lib\site-packages (from
kaggle->opendatasets) (1.26.16)
Requirement already satisfied: bleach in d:\anaconda\lib\site-packages (from
kaggle->opendatasets) (4.1.0)
Requirement already satisfied: packaging in d:\anaconda\lib\site-packages (from
bleach->kaggle->opendatasets) (23.1)
Requirement already satisfied: webencodings in d:\anaconda\lib\site-packages
(from bleach->kaggle->opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in d:\anaconda\lib\site-
packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\anaconda\lib\site-
packages (from requests->kaggle->opendatasets) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in d:\anaconda\lib\site-packages
(from requests->kaggle->opendatasets) (3.4)
```

[4]:
```
import opendatasets as od
```

[5]:
```
dataset = 'https://www.kaggle.com/datasets/navoneel/
  ↪brain-mri-images-for-brain-tumor-detection'
```

[6]:
```
od.download(dataset)
```

```
Skipping, found downloaded files in ".\brain-mri-images-for-brain-tumor-
detection" (use force=True to force download)
```

[7]:
```
import os
```

[8]:
```
data_dir = r'.\brain-mri-images-for-brain-tumor-detection'
```

[9]:
```
os.listdir(data_dir)
```

[9]:
```
['brain_tumor_dataset', 'no', 'yes']
```

[10]:
```
pip install tensorflow==2.14.0
```

```
Requirement already satisfied: tensorflow==2.14.0 in d:\anaconda\lib\site-
packages (2.14.0)
Requirement already satisfied: tensorflow-intel==2.14.0 in d:\anaconda\lib\site-
```

```
packages (from tensorflow==2.14.0) (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
d:\anaconda\lib\site-packages (from tensorflow-
intel==2.14.0->tensorflow==2.14.0) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (18.1.1)
Requirement already satisfied: ml-dtypes==0.2.0 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (0.2.0)
Requirement already satisfied: numpy>=1.23.5 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (1.26.4)
Requirement already satisfied: opt-einsum>=2.3.2 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (3.4.0)
Requirement already satisfied: packaging in d:\anaconda\lib\site-packages (from
tensorflow-intel==2.14.0->tensorflow==2.14.0) (23.1)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in d:\anaconda\lib\site-packages (from tensorflow-
intel==2.14.0->tensorflow==2.14.0) (4.25.5)
Requirement already satisfied: setuptools in d:\anaconda\lib\site-packages (from
tensorflow-intel==2.14.0->tensorflow==2.14.0) (68.0.0)
Requirement already satisfied: six>=1.12.0 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in d:\anaconda\lib\site-packages
(from tensorflow-intel==2.14.0->tensorflow==2.14.0) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (4.7.1)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
d:\anaconda\lib\site-packages (from tensorflow-
intel==2.14.0->tensorflow==2.14.0) (0.31.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (1.67.1)
Requirement already satisfied: tensorboard<2.15,>=2.14 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in
d:\anaconda\lib\site-packages (from tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2.14.0)
```

```
Requirement already satisfied: keras<2.15,>=2.14.0 in d:\anaconda\lib\site-
packages (from tensorflow-intel==2.14.0->tensorflow==2.14.0) (2.14.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in d:\anaconda\lib\site-
packages (from astunparse>=1.6.0->tensorflow-intel==2.14.0->tensorflow==2.14.0)
(0.38.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in d:\anaconda\lib\site-
packages (from tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2.36.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
d:\anaconda\lib\site-packages (from tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in d:\anaconda\lib\site-packages
(from tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow==2.14.0)
(3.4.1)
Requirement already satisfied: requests<3,>=2.21.0 in d:\anaconda\lib\site-
packages (from tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
d:\anaconda\lib\site-packages (from tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in d:\anaconda\lib\site-packages
(from tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow==2.14.0)
(2.2.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in d:\anaconda\lib\site-
packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (5.5.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in d:\anaconda\lib\site-
packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in d:\anaconda\lib\site-packages
(from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in d:\anaconda\lib\site-
packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\anaconda\lib\site-
packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in d:\anaconda\lib\site-packages
(from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\anaconda\lib\site-
packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in d:\anaconda\lib\site-
packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2023.11.17)
```

```
Requirement already satisfied: MarkupSafe>=2.1.1 in d:\anaconda\lib\site-
packages (from werkzeug>=1.0.1->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in d:\anaconda\lib\site-
packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in d:\anaconda\lib\site-packages
(from requests-oauthlib>=0.7.0->google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow-
intel==2.14.0->tensorflow==2.14.0) (3.2.2)
Note: you may need to restart the kernel to use updated packages.
```

[11]:
```
pip install numpy==1.23.5 pandas==2.1.3 pip  scipy==1.9.3
```

```
Collecting numpy==1.23.5
  Obtaining dependency information for numpy==1.23.5 from https://files.pythonho
sted.org/packages/19/0d/b8c34e4baf258d77a8592bdce45183e9a12874c167f5966c7dd467b7
4ea9/numpy-1.23.5-cp311-cp311-win_amd64.whl.metadata
  Using cached numpy-1.23.5-cp311-cp311-win_amd64.whl.metadata (2.3 kB)
Requirement already satisfied: pandas==2.1.3 in d:\anaconda\lib\site-packages
(2.1.3)
Requirement already satisfied: pip in d:\anaconda\lib\site-packages (23.2.1)
Collecting scipy==1.9.3
  Obtaining dependency information for scipy==1.9.3 from https://files.pythonhos
ted.org/packages/42/81/0a64d2204c3b261380ac96c6d61f018528108b62c0e21e6153a58cebf
4f6/scipy-1.9.3-cp311-cp311-win_amd64.whl.metadata
  Downloading scipy-1.9.3-cp311-cp311-win_amd64.whl.metadata (58 kB)
     ---------------------------------------- 0.0/58.5 kB ? eta -:--:--
     ------------------ -------------------- 30.7/58.5 kB 660.6 kB/s eta 0:00:01
     ---------------------------------------- 58.5/58.5 kB 778.0 kB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.8.2 in d:\anaconda\lib\site-
packages (from pandas==2.1.3) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in d:\anaconda\lib\site-packages
(from pandas==2.1.3) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in d:\anaconda\lib\site-packages
(from pandas==2.1.3) (2023.3)
Requirement already satisfied: six>=1.5 in d:\anaconda\lib\site-packages (from
python-dateutil>=2.8.2->pandas==2.1.3) (1.16.0)
Using cached numpy-1.23.5-cp311-cp311-win_amd64.whl (14.6 MB)
Downloading scipy-1.9.3-cp311-cp311-win_amd64.whl (39.9 MB)
   ---------------------------------------- 0.0/39.9 MB ? eta -:--:--
   ---------------------------------------- 0.1/39.9 MB 1.5 MB/s eta 0:00:27
   ---------------------------------------- 0.3/39.9 MB 3.3 MB/s eta 0:00:13
    --------------------------------------- 0.6/39.9 MB 4.4 MB/s eta 0:00:09
    --------------------------------------- 0.8/39.9 MB 4.9 MB/s eta 0:00:08
   - -------------------------------------- 1.1/39.9 MB 5.2 MB/s eta 0:00:08
   - -------------------------------------- 1.4/39.9 MB 5.4 MB/s eta 0:00:08
```

```
- ---------------------------------- 1.6/39.9 MB 5.7 MB/s eta 0:00:07
- ---------------------------------- 1.9/39.9 MB 5.8 MB/s eta 0:00:07
-- ---------------------------------- 2.3/39.9 MB 6.3 MB/s eta 0:00:06
-- ---------------------------------- 2.6/39.9 MB 6.4 MB/s eta 0:00:06
--- ---------------------------------- 3.1/39.9 MB 6.9 MB/s eta 0:00:06
--- ---------------------------------- 3.6/39.9 MB 7.1 MB/s eta 0:00:06
--- ---------------------------------- 4.0/39.9 MB 7.2 MB/s eta 0:00:05
---- ---------------------------------- 4.2/39.9 MB 7.3 MB/s eta 0:00:05
---- ---------------------------------- 4.7/39.9 MB 7.5 MB/s eta 0:00:05
---- ---------------------------------- 4.9/39.9 MB 7.5 MB/s eta 0:00:05
----- ---------------------------------- 5.6/39.9 MB 7.7 MB/s eta 0:00:05
----- ---------------------------------- 5.9/39.9 MB 7.9 MB/s eta 0:00:05
------ ---------------------------------- 6.3/39.9 MB 7.9 MB/s eta 0:00:05
------ ---------------------------------- 6.7/39.9 MB 8.0 MB/s eta 0:00:05
------- ---------------------------------- 7.1/39.9 MB 8.0 MB/s eta 0:00:05
------- ---------------------------------- 7.5/39.9 MB 8.1 MB/s eta 0:00:05
------- ---------------------------------- 7.9/39.9 MB 8.1 MB/s eta 0:00:04
-------- ---------------------------------- 8.3/39.9 MB 8.1 MB/s eta 0:00:04
-------- ---------------------------------- 8.7/39.9 MB 8.2 MB/s eta 0:00:04
--------- ---------------------------------- 9.1/39.9 MB 8.3 MB/s eta 0:00:04
--------- ---------------------------------- 9.5/39.9 MB 8.2 MB/s eta 0:00:04
---------- ---------------------------------- 10.0/39.9 MB 8.3 MB/s eta 0:00:04
---------- ---------------------------------- 10.3/39.9 MB 8.6 MB/s eta 0:00:04
---------- ---------------------------------- 10.7/39.9 MB 8.7 MB/s eta 0:00:04
----------- ---------------------------------- 11.0/39.9 MB 8.8 MB/s eta 0:00:04
----------- ---------------------------------- 11.4/39.9 MB 9.0 MB/s eta 0:00:04
----------- ---------------------------------- 11.8/39.9 MB 9.1 MB/s eta 0:00:04
------------ ---------------------------------- 12.2/39.9 MB 9.1 MB/s eta 0:00:04
------------ ---------------------------------- 12.6/39.9 MB 9.1 MB/s eta 0:00:03
------------ ---------------------------------- 12.8/39.9 MB 9.0 MB/s eta 0:00:04
------------- ---------------------------------- 13.4/39.9 MB 9.0 MB/s eta 0:00:03
------------- ---------------------------------- 13.8/39.9 MB 9.0 MB/s eta 0:00:03
-------------- ---------------------------------- 14.1/39.9 MB 9.0 MB/s eta 0:00:03
-------------- ---------------------------------- 14.4/39.9 MB 9.0 MB/s eta 0:00:03
-------------- ---------------------------------- 14.8/39.9 MB 9.0 MB/s eta 0:00:03
--------------- ---------------------------------- 15.2/39.9 MB 9.4 MB/s eta 0:00:03
--------------- ---------------------------------- 15.6/39.9 MB 9.0 MB/s eta 0:00:03
---------------- ---------------------------------- 16.0/39.9 MB 8.8 MB/s eta 0:00:03
---------------- ---------------------------------- 16.3/39.9 MB 9.0 MB/s eta 0:00:03
---------------- ---------------------------------- 16.7/39.9 MB 8.8 MB/s eta 0:00:03
----------------- ---------------------------------- 17.1/39.9 MB 8.8 MB/s eta 0:00:03
----------------- ---------------------------------- 17.6/39.9 MB 8.8 MB/s eta 0:00:03
------------------ ---------------------------------- 18.0/39.9 MB 8.7 MB/s eta 0:00:03
------------------ ---------------------------------- 18.4/39.9 MB 8.8 MB/s eta 0:00:03
------------------ ---------------------------------- 18.5/39.9 MB 8.6 MB/s eta 0:00:03
------------------ ---------------------------------- 18.9/39.9 MB 8.6 MB/s eta 0:00:03
------------------- ---------------------------------- 19.2/39.9 MB 8.6 MB/s eta 0:00:03
------------------- ---------------------------------- 19.5/39.9 MB 8.5 MB/s eta 0:00:03
```

```
------------------ -------------------- 19.7/39.9 MB 8.4 MB/s eta 0:00:03
------------------ -------------------- 19.9/39.9 MB 8.2 MB/s eta 0:00:03
------------------- ------------------- 20.2/39.9 MB 8.1 MB/s eta 0:00:03
------------------- ------------------- 20.5/39.9 MB 8.1 MB/s eta 0:00:03
------------------- ------------------- 20.7/39.9 MB 8.0 MB/s eta 0:00:03
-------------------- ------------------ 21.0/39.9 MB 7.9 MB/s eta 0:00:03
-------------------- ------------------ 21.2/39.9 MB 7.9 MB/s eta 0:00:03
-------------------- ------------------ 21.5/39.9 MB 7.8 MB/s eta 0:00:03
-------------------- ------------------ 21.8/39.9 MB 7.8 MB/s eta 0:00:03
--------------------- ----------------- 22.1/39.9 MB 7.8 MB/s eta 0:00:03
--------------------- ----------------- 22.5/39.9 MB 7.8 MB/s eta 0:00:03
--------------------- ----------------- 22.7/39.9 MB 7.7 MB/s eta 0:00:03
---------------------- ---------------- 23.0/39.9 MB 7.9 MB/s eta 0:00:03
---------------------- ---------------- 23.4/39.9 MB 7.6 MB/s eta 0:00:03
---------------------- ---------------- 23.7/39.9 MB 7.6 MB/s eta 0:00:03
----------------------- --------------- 24.0/39.9 MB 7.6 MB/s eta 0:00:03
----------------------- --------------- 24.4/39.9 MB 7.7 MB/s eta 0:00:03
----------------------- --------------- 24.6/39.9 MB 7.5 MB/s eta 0:00:03
------------------------ -------------- 25.0/39.9 MB 7.5 MB/s eta 0:00:02
------------------------ -------------- 25.3/39.9 MB 7.5 MB/s eta 0:00:02
------------------------ -------------- 25.7/39.9 MB 7.4 MB/s eta 0:00:02
------------------------- ------------- 26.0/39.9 MB 7.4 MB/s eta 0:00:02
------------------------- ------------- 26.4/39.9 MB 7.4 MB/s eta 0:00:02
------------------------- ------------- 26.7/39.9 MB 7.4 MB/s eta 0:00:02
-------------------------- ------------ 27.0/39.9 MB 7.4 MB/s eta 0:00:02
-------------------------- ------------ 27.4/39.9 MB 7.4 MB/s eta 0:00:02
-------------------------- ------------ 27.6/39.9 MB 7.4 MB/s eta 0:00:02
--------------------------- ----------- 27.9/39.9 MB 7.4 MB/s eta 0:00:02
--------------------------- ----------- 28.2/39.9 MB 7.3 MB/s eta 0:00:02
--------------------------- ----------- 28.5/39.9 MB 7.2 MB/s eta 0:00:02
--------------------------- ----------- 28.9/39.9 MB 7.3 MB/s eta 0:00:02
---------------------------- ---------- 29.1/39.9 MB 7.2 MB/s eta 0:00:02
---------------------------- ---------- 29.5/39.9 MB 7.2 MB/s eta 0:00:02
---------------------------- ---------- 29.8/39.9 MB 7.4 MB/s eta 0:00:02
----------------------------- --------- 30.1/39.9 MB 7.4 MB/s eta 0:00:02
----------------------------- --------- 30.5/39.9 MB 7.5 MB/s eta 0:00:02
----------------------------- --------- 30.9/39.9 MB 7.5 MB/s eta 0:00:02
------------------------------ -------- 31.2/39.9 MB 7.6 MB/s eta 0:00:02
------------------------------ -------- 31.2/39.9 MB 7.6 MB/s eta 0:00:02
------------------------------ -------- 31.5/39.9 MB 7.4 MB/s eta 0:00:02
------------------------------ -------- 31.9/39.9 MB 7.4 MB/s eta 0:00:02
------------------------------- ------- 32.3/39.9 MB 7.5 MB/s eta 0:00:02
------------------------------- ------- 32.7/39.9 MB 7.5 MB/s eta 0:00:01
-------------------------------- ------ 33.1/39.9 MB 7.5 MB/s eta 0:00:01
-------------------------------- ------ 33.4/39.9 MB 7.5 MB/s eta 0:00:01
-------------------------------- ------ 33.7/39.9 MB 7.5 MB/s eta 0:00:01
--------------------------------- ----- 34.1/39.9 MB 7.6 MB/s eta 0:00:01
--------------------------------- ----- 34.4/39.9 MB 7.5 MB/s eta 0:00:01
```

```
   ---------------------------------- ----- 34.7/39.9 MB 7.5 MB/s eta 0:00:01
   ---------------------------------- ----- 35.1/39.9 MB 7.5 MB/s eta 0:00:01
   ---------------------------------- ----- 35.4/39.9 MB 7.5 MB/s eta 0:00:01
   ---------------------------------- ----- 35.8/39.9 MB 7.5 MB/s eta 0:00:01
   ---------------------------------- ----- 36.1/39.9 MB 7.6 MB/s eta 0:00:01
   ---------------------------------- ----- 36.5/39.9 MB 7.6 MB/s eta 0:00:01
   ---------------------------------- ----- 36.9/39.9 MB 7.6 MB/s eta 0:00:01
   ---------------------------------- ----- 37.2/39.9 MB 7.6 MB/s eta 0:00:01
   ---------------------------------- ----- 37.6/39.9 MB 7.7 MB/s eta 0:00:01
   ---------------------------------- ----- 37.8/39.9 MB 7.5 MB/s eta 0:00:01
   ---------------------------------- ----- 38.2/39.9 MB 7.7 MB/s eta 0:00:01
   ---------------------------------- ----- 38.6/39.9 MB 7.8 MB/s eta 0:00:01
   ---------------------------------- ----- 38.9/39.9 MB 7.8 MB/s eta 0:00:01
   ---------------------------------------  39.3/39.9 MB 7.9 MB/s eta 0:00:01
   ---------------------------------------  39.6/39.9 MB 8.0 MB/s eta 0:00:01
   ---------------------------------------  39.9/39.9 MB 7.9 MB/s eta 0:00:01
   ---------------------------------------  39.9/39.9 MB 7.9 MB/s eta 0:00:01
   ---------------------------------------- 39.9/39.9 MB 7.4 MB/s eta 0:00:00
Installing collected packages: numpy, scipy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.4
    Uninstalling numpy-1.26.4:
      Successfully uninstalled numpy-1.26.4
Note: you may need to restart the kernel to use updated packages.

ERROR: Could not install packages due to an OSError: [WinError 5] Access is
denied: 'D:\\Anaconda\\Lib\\site-
packages\\~.mpy.libs\\libopenblas64__v0.3.23-293-gc2f4bdbb-
gcc_10_3_0-2bde3a66a51006b2b53eb373ff767a3f.dll'
Consider using the `--user` option or check the permissions.
```

1.5 Importer les modules nécessaires

[13]:
```python
# Importation de la bibliothèque PIL pour le traitement des images
from PIL import Image   # Pour ouvrir, manipuler et enregistrer des images avec␣
 ↪Python.

# Importation des outils de handling des données
import cv2     # OpenCV, utilisé pour le traitement et l'analyse d'images et de␣
 ↪vidéos.
import numpy as np    # Numpy, pour le calcul numérique et les tableaux␣
 ↪multidimensionnels.
import pandas as pd    # Pandas, pour la manipulation de données sous forme de␣
 ↪tableaux (DataFrames).
import seaborn as sns     # Seaborn, pour des visualisations avancées basées␣
 ↪sur Matplotlib.
```

```
sns.set_style('darkgrid')     # Applique un style de grille sombre aux␣
 ↪graphiques de Seaborn pour une meilleure visibilité.

# Importation des outils de scikit-learn pour l'évaluation des modèles
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
D:\Anaconda\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning:
Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5'
currently installed).
  from pandas.core import (
```

2 Prétraitement des données

2.1 Lire les données et les stocker dans un dataframe (df)

[14]:
```
# # Générer les chemins de données avec les labels
data_dir = r'.\brain-mri-images-for-brain-tumor-detection' # Définir le␣
 ↪répertoire contenant les images de MRIs
filepaths = []    # Liste vide pour stocker les chemins des fichiers d'images.
labels = []      # Liste vide pour stocker les Labels correspondant à chaque␣
 ↪image

folds = os.listdir(data_dir)
for fold in folds:
    foldpath = os.path.join(data_dir, fold)
    filelist = os.listdir(foldpath)
    for file in filelist:
        fpath = os.path.join(foldpath, file)

        filepaths.append(fpath)
        labels.append(fold)

# Concaténer les chemins de fichiers avec les labels dans un DataFrame
Fseries = pd.Series(filepaths, name= 'filepaths') # Créer une série pandas␣
 ↪contenant les chemins de fichiers.
Lseries = pd.Series(labels, name='labels')      # Créer une série pandas␣
 ↪contenant les étiquettes.
df = pd.concat([Fseries, Lseries], axis= 1)      # Combiner les deux séries en␣
 ↪un DataFrame sur l'axe des colonnes.
```

[15]:
```
df
```

[15]:

	filepaths	labels
0	.\brain-mri-images-for-brain-tumor-detection\b…	brain_tumor_dataset
1	.\brain-mri-images-for-brain-tumor-detection\b…	brain_tumor_dataset

```
2    .\brain-mri-images-for-brain-tumor-detection\n…    no
3    .\brain-mri-images-for-brain-tumor-detection\n…    no
4    .\brain-mri-images-for-brain-tumor-detection\n…    no
..                                                  …    …
250  .\brain-mri-images-for-brain-tumor-detection\y…    yes
251  .\brain-mri-images-for-brain-tumor-detection\y…    yes
252  .\brain-mri-images-for-brain-tumor-detection\y…    yes
253  .\brain-mri-images-for-brain-tumor-detection\y…    yes
254  .\brain-mri-images-for-brain-tumor-detection\y…    yes

[255 rows x 2 columns]
```

2.2 Diviser les données en ensembles d'entraînement, de validation et de test

[16]:
```
# Définir la variable 'strat' qui contient les étiquettes
strat = df['labels']  # 'labels' contient les (yes , no) des images.

# Diviser le DataFrame en ensembles d'entraînement et de test
train_df, test_df = train_test_split(df,  # Diviser le DataFrame 'df' en deux␣
 ↪sous-ensembles : 'train_df' et 'test_df'.
                                     train_size= 0.8,  # 80% des données␣
 ↪seront utilisées pour l'entraînement.
                                     shuffle= True,  # Mélanger les données␣
 ↪avant de les diviser.
                                     random_state= 123,
                                    # Fixer la graine aléatoire pour garantir␣
 ↪la reproductibilité des résultats.
                                     stratify= strat)
                        # Assurer que la répartition des labels dans les ensembles␣
 ↪d'entraînement et de test est équilibrée.
```

[17]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.18.0
```

2.3 Ici, je continue d'importer les modules nécessaires

[18]:
```
# Importer les bibliothèques nécessaires
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential # permet de créer des réseaux de␣
 ↪neurones couche par couche.
from tensorflow.keras.optimizers import Adam, Adamax # utilisés pour ajuster␣
 ↪les poids du modèle lors de l'entraînement.
from tensorflow.keras.preprocessing.image import ImageDataGenerator # effectuer␣
 ↪des augmentations sur les données d'images.
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers
# Ignorer les avertissements
import warnings
warnings.filterwarnings("ignore") # Ignorer les avertissements pour éviter leur␣
 ↪affichage pendant l'exécution du code.

print ('modules loaded')
```

```
modules loaded
```

2.4 Créer un générateur de données d'image

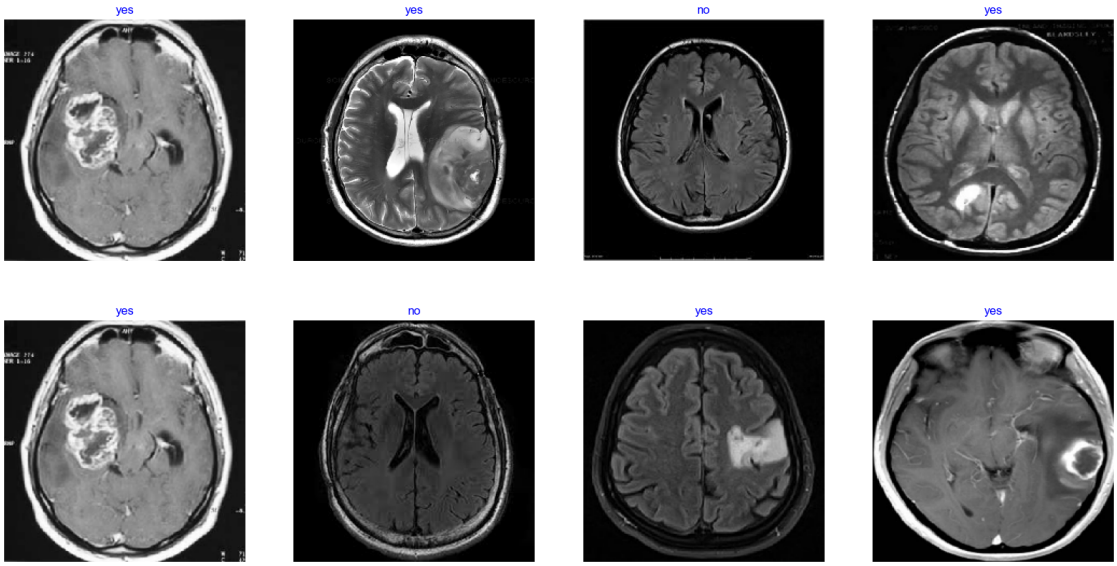
[19]:
```
# Taille des images redimensionnées
batch_size = 8  # Le nombre d'images traitées par lot (batch) pendant␣
 ↪l'entraînement. Ici, 8 images par itération.
img_size = (224, 224)  # La taille de l'image redimensionnée, ici 224x224␣
 ↪pixels.
channels = 3  # Nombre de canaux de couleur (3 pour RGB : rouge, vert, bleu).
img_shape = (img_size[0], img_size[1], channels)  # Forme de l'image (224, 224,␣
 ↪3).

# Création d'un générateur d'images pour l'entraînement et le test
tr_gen = ImageDataGenerator()  # Création d'un générateur pour l'entraînement␣
 ↪(sans transformation ici).
ts_gen = ImageDataGenerator()  # Création d'un générateur pour le test (sans␣
 ↪transformation ici).

# Générateur pour les données d'entraînement
train_gen = tr_gen.flow_from_dataframe(  # Crée un générateur de données à␣
 ↪partir du DataFrame d'entraînement.
    train_df,  # DataFrame contenant les données d'entraînement.
    x_col='filepaths',  # Colonne contenant les chemins des fichiers d'images.
    y_col='labels',  # Colonne contenant les étiquettes des images (labels).
    target_size=img_size,  # Redimensionnement des images à 224x224 pixels.
    class_mode='categorical',  # Mode des classes (ici, multi-classes avec␣
 ↪étiquettes catégorielles).
    color_mode='rgb',          # Mode de couleur (ici, RGB avec 3 canaux).
    shuffle=True,          # Mélanger les données avant chaque époque␣
 ↪d'entraînement.
    batch_size=batch_size  # Taille des lots d'images pendant l'entraînement.
)

# Générateur pour les données de test
test_gen = ts_gen.flow_from_dataframe(  # Crée un générateur de données à␣
 ↪partir du DataFrame de test.
```

```
    test_df,  # DataFrame contenant les données de test.
    x_col='filepaths',  # Colonne contenant les chemins des fichiers d'images.
    y_col='labels',  # Colonne contenant les labels des images .
    target_size=img_size,  # Redimensionnement des images à 224x224 pixels.
    class_mode='categorical',  # Mode des classes .
    color_mode='rgb',  # Mode de couleur (ici, RGB avec 3 canaux).
    shuffle=False,  # Ne pas mélanger les données de test pour préserver␣
 ↪l'ordre.
    batch_size=batch_size  # Taille des lots d'images pendant l'évaluation.
)
```

```
Found 202 validated image filenames belonging to 2 classes.
Found 51 validated image filenames belonging to 2 classes.
```

2.5 Afficher 8 images d'entraînement

[23]:
```
g_dict = train_gen.class_indices      # Définit un dictionnaire {'classe':␣
 ↪index} qui mappe chaque classe à un indice
classes = list(g_dict.keys())
images, labels = next(train_gen)      # Récupère un lot d'images et leurs␣
 ↪étiquettes (labels) à partir du train generator

plt.figure(figsize=(20, 10))          # Crée une figure avec une taille de␣
 ↪20x10 pouces pour afficher les images

# Boucle pour afficher 10 images (car batch_size=10)
for i in range(8):
    plt.subplot(2, 4, i + 1)
    image = images[i] / 255           # Redimensionne les pixels de l'image␣
 ↪dans la plage [0, 1] pour normaliser les données
    plt.imshow(image)                 # Affiche l'image dans le sous-graphe␣
 ↪actuel
    index = np.argmax(labels[i])      # Récupère l'indice de la classe à␣
 ↪partir des étiquettes (encodées en one-hot)
    class_name = classes[index]       # Récupère le nom de la classe en␣
 ↪utilisant l'indice précédemment obtenu
    plt.title(class_name, color='blue', fontsize=12)  # Affiche le nom de la␣
 ↪classe comme titre de l'image, avec couleur bleue
    plt.axis('off')

plt.show()                            # Affiche toutes les images et leurs␣
 ↪titres dans une fenêtre de visualisation
```



3 Structure du modèle

3.1 Création d'un modèle générique

[24]:
```
# Définition de la taille des images et des paramètres d'entrée
img_size = (224, 224)  # Définir la taille des images d'entrée (224x224 pixels).
channels = 3  # Nombre de canaux pour l'image (3 pour RGB : Rouge, Vert, Bleu).
img_shape = (img_size[0], img_size[1], channels)  # La forme de l'image pour le␣
 ↪modèle (224, 224, 3) pour des images RGB.

# Définir le nombre de classes basé sur les labels des données d'entraînement
class_count = len(list(train_gen.class_indices.keys()))  # Calcule le nombre de␣
 ↪classes.

# Créer le modèle de base pré-entraîné EfficientNetB3
base_model = tf.keras.applications.EfficientNetB3(
    include_top=False,  # Exclut la partie "top" (couches finales de␣
 ↪classification), pour ajouter notre propre couche de sortie.
    weights="imagenet",  # Utilise les poids pré-entraînés sur ImageNet.
    input_shape=img_shape,  # Définir la forme des images d'entrée.
    pooling='max'  # Utilise un pooling de type "max" pour réduire la␣
 ↪dimensionnalité des caractéristiques extraites.
)

# Créer le modèle séquentiel en ajoutant des couches supplémentaires
model = tf.keras.Sequential([
    base_model,  # Ajoute le modèle EfficientNetB3 pré-entraîné.
```

```
    tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001), ␣
 ↪# Normalisation des lots pour améliorer la stabilité.
    tf.keras.layers.Dense(
        256,  # Ajoute une couche dense avec 256 neurones.
        kernel_regularizer=tf.keras.regularizers.l2(0.016), # la régularisation␣
 ↪L2 sur les poids pour éviter l'overfitting.
        activity_regularizer=tf.keras.regularizers.l1(0.006),  # Applique la␣
 ↪régularisation L1 sur les activations.
        bias_regularizer=tf.keras.regularizers.l1(0.006),  # Applique la␣
 ↪régularisation L1 sur les biais.
        activation='relu'  # Fonction d'activation ReLU (Rectified Linear Unit)␣
 ↪pour introduire de la non-linéarité.
    ),  # Ajoute une couche Dropout avec un taux de 45% pour prévenir le␣
 ↪sur-apprentissage.
    tf.keras.layers.Dropout(rate=0.45, seed=123),
    # Couche de sortie avec activation softmax pour la classification␣
 ↪multi-classes.
    tf.keras.layers.Dense(class_count, activation='softmax')
])
# Compiler le modèle avec l'optimiseur et la fonction de perte appropriés
model.compile(
    optimizer=tf.keras.optimizers.Adamax(learning_rate=0.001),  # l'optimiseur␣
 ↪Adamax avec un taux d'apprentissage de 0.001.
    loss='categorical_crossentropy',  # Fonction de perte pour la␣
 ↪classification multi-classes.
    metrics=['accuracy']  # Utilise la précision comme métrique pour évaluer␣
 ↪les performances du modèle.
)

# Afficher un résumé du modèle pour visualiser sa structure
model.summary()  # Affiche la structure du modèle, les couches, le nombre de␣
 ↪paramètres, etc.
```

```
Model: "sequential"
```

```
 Layer (type)                        Output Shape                      ␣
 ↪Param #

 efficientnetb3 (Functional)         (None, 1536)                      ␣
 ↪10,783,535

 batch_normalization                 (None, 1536)                      ␣
 ↪6,144
 (BatchNormalization)                                                  ␣
 ↪
```

```
dense (Dense)                        (None, 256)                 ␣
↪393,472

dropout (Dropout)                    (None, 256)                 ␣
↪  0

dense_1 (Dense)                      (None, 2)                   ␣
↪514


Total params: 11,183,665 (42.66 MB)

Trainable params: 11,093,290 (42.32 MB)

Non-trainable params: 90,375 (353.03 KB)
```

3.2 Résultats du Modèle

Le modèle utilise **EfficientNetB3** pré-entraîné comme base, suivi de couches supplémentaires pour la régularisation et la classification. Voici une explication des résultats du modèle :

3.2.1 Couches du modèle

- **EfficientNetB3** : Cette couche est responsable de l'extraction des caractéristiques à partir des images d'entrée. Elle réduit la taille des images tout en conservant les informations importantes pour la classification, produisant une sortie de forme `(None, 1536)`, ce qui signifie que chaque image est représentée par un vecteur de 1536 caractéristiques.
- **BatchNormalization** : Après l'extraction des caractéristiques, la normalisation par lot est appliquée pour stabiliser l'entraînement en ajustant les activations. Cela aide à accélérer l'apprentissage et à rendre le modèle plus stable.
- **Dense (256 neurones)** : Cette couche entièrement connectée avec 256 neurones apprend à combiner les caractéristiques extraites pour détecter des patterns complexes dans les données d'entrée. Le nombre de paramètres associés à cette couche est de **393,472**.
- **Dropout** : Le taux de **Dropout** de 45 % est appliqué pour éviter le surapprentissage en désactivant aléatoirement certaines connexions entre les neurones pendant l'entraînement.
- **Dense (sortie)** : Enfin, la couche de sortie avec 2 neurones utilise la fonction **softmax** pour prédire la probabilité d'appartenir à chaque classe (par exemple, présence ou absence de tumeur). La sortie de cette couche est un vecteur de forme `(None, 2)`.

3.2.2 Conclusion

Le modèle est capable de classifier les images en deux catégories : **tumeur** et **non-tumeur**. La structure du modèle, en combinant un réseau pré-entraîné avec des techniques de régularisation

comme la normalisation par lot et le dropout, permet de capturer des caractéristiques complexes tout en évitant le surapprentissage. Ce modèle devrait être efficace pour la détection de tumeurs sur des images de cerveau, en particulier grâce à l'utilisation de **EfficientNetB3**, un modèle pré-entraîné performant.

4 Entraînement du modèle

[25]:
```
epochs = 30  # Nombre total d'époques pour l'entraînement.

history = model.fit(x=train_gen,  # est le générateur d'images qui fournit les␣
 ↪données d'entraînement par lots.
                    epochs=epochs,  # Le modèle sera entraîné pendant 30␣
 ↪époques.
                    verbose=1,
# affiche une barre de progression avec des informations sur la perte et␣
 ↪l'exactitude du modèle pendant chaque époque.
                    validation_data=test_gen,
# utilisé comme ensemble de validation pour évaluer les performances du modèle␣
 ↪sur des données non vues pendant l'entraînement.

                    validation_steps=None,  #  signifie que le nombre d'étapes␣
 ↪de validation est automatiquement calculé.
                    shuffle=False) # 'shuffle=False' indique que les données ne␣
 ↪seront pas mélangées pendant l'entraînement
```

```
Epoch 1/30
26/26             89s 2s/step -
accuracy: 0.5392 - loss: 14.1243 - val_accuracy: 0.7059 - val_loss: 16.0328
Epoch 2/30
26/26             40s 2s/step -
accuracy: 0.8224 - loss: 12.3312 - val_accuracy: 0.7255 - val_loss: 11.9535
Epoch 3/30
26/26             41s 2s/step -
accuracy: 0.8466 - loss: 10.3324 - val_accuracy: 0.8039 - val_loss: 11.1556
Epoch 4/30
26/26             41s 2s/step -
accuracy: 0.9012 - loss: 8.5292 - val_accuracy: 0.9216 - val_loss: 9.6338
Epoch 5/30
26/26             42s 2s/step -
accuracy: 0.8444 - loss: 7.4529 - val_accuracy: 0.7647 - val_loss: 8.5084
Epoch 6/30
26/26             39s 2s/step -
accuracy: 0.7725 - loss: 6.7981 - val_accuracy: 0.8039 - val_loss: 7.2478
Epoch 7/30
26/26             43s 2s/step -
accuracy: 0.7664 - loss: 6.4048 - val_accuracy: 0.8824 - val_loss: 6.5443
Epoch 8/30
```

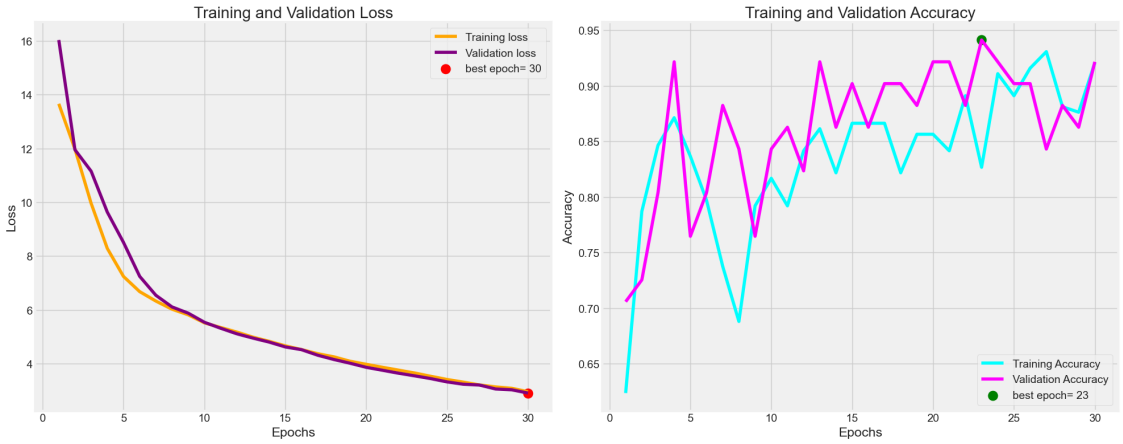
```
26/26             43s 2s/step -
accuracy: 0.6998 - loss: 6.0765 - val_accuracy: 0.8431 - val_loss: 6.1104
Epoch 9/30
26/26             44s 2s/step -
accuracy: 0.8087 - loss: 5.8458 - val_accuracy: 0.7647 - val_loss: 5.8798
Epoch 10/30
26/26             44s 2s/step -
accuracy: 0.8566 - loss: 5.5733 - val_accuracy: 0.8431 - val_loss: 5.5410
Epoch 11/30
26/26             43s 2s/step -
accuracy: 0.7537 - loss: 5.3893 - val_accuracy: 0.8627 - val_loss: 5.3194
Epoch 12/30
26/26             44s 2s/step -
accuracy: 0.8427 - loss: 5.2285 - val_accuracy: 0.8235 - val_loss: 5.1134
Epoch 13/30
26/26             43s 2s/step -
accuracy: 0.8508 - loss: 5.0470 - val_accuracy: 0.9216 - val_loss: 4.9524
Epoch 14/30
26/26             41s 2s/step -
accuracy: 0.8125 - loss: 4.8655 - val_accuracy: 0.8627 - val_loss: 4.8046
Epoch 15/30
26/26             46s 2s/step -
accuracy: 0.8678 - loss: 4.7086 - val_accuracy: 0.9020 - val_loss: 4.6244
Epoch 16/30
26/26             47s 2s/step -
accuracy: 0.9009 - loss: 4.5418 - val_accuracy: 0.8627 - val_loss: 4.5226
Epoch 17/30
26/26             46s 2s/step -
accuracy: 0.8353 - loss: 4.4151 - val_accuracy: 0.9020 - val_loss: 4.3169
Epoch 18/30
26/26             42s 2s/step -
accuracy: 0.8320 - loss: 4.2986 - val_accuracy: 0.9020 - val_loss: 4.1577
Epoch 19/30
26/26             45s 2s/step -
accuracy: 0.8256 - loss: 4.1334 - val_accuracy: 0.8824 - val_loss: 4.0187
Epoch 20/30
26/26             44s 2s/step -
accuracy: 0.8433 - loss: 4.0234 - val_accuracy: 0.9216 - val_loss: 3.8667
Epoch 21/30
26/26             46s 2s/step -
accuracy: 0.8516 - loss: 3.8894 - val_accuracy: 0.9216 - val_loss: 3.7574
Epoch 22/30
26/26             42s 2s/step -
accuracy: 0.8978 - loss: 3.7898 - val_accuracy: 0.8824 - val_loss: 3.6472
Epoch 23/30
26/26             45s 2s/step -
accuracy: 0.8322 - loss: 3.6685 - val_accuracy: 0.9412 - val_loss: 3.5493
Epoch 24/30
```

```
26/26              43s 2s/step -
accuracy: 0.9060 - loss: 3.5748 - val_accuracy: 0.9216 - val_loss: 3.4457
Epoch 25/30
26/26              44s 2s/step -
accuracy: 0.9191 - loss: 3.4339 - val_accuracy: 0.9020 - val_loss: 3.3243
Epoch 26/30
26/26              43s 2s/step -
accuracy: 0.9419 - loss: 3.3041 - val_accuracy: 0.9020 - val_loss: 3.2380
Epoch 27/30
26/26              43s 2s/step -
accuracy: 0.9373 - loss: 3.2273 - val_accuracy: 0.8431 - val_loss: 3.2081
Epoch 28/30
26/26              43s 2s/step -
accuracy: 0.8743 - loss: 3.1764 - val_accuracy: 0.8824 - val_loss: 3.0610
Epoch 29/30
26/26              43s 2s/step -
accuracy: 0.8276 - loss: 3.1244 - val_accuracy: 0.8627 - val_loss: 3.0300
Epoch 30/30
26/26              42s 2s/step -
accuracy: 0.9319 - loss: 2.9741 - val_accuracy: 0.9216 - val_loss: 2.9103
```

5 Afficher les performances du modèle

[26]:
```
# Define needed variables
tr_acc = history.history['accuracy']  # Historique de la précision de␣
 ↪l'entraînement.
tr_loss = history.history['loss']  # Historique de la perte de l'entraînement.
val_acc = history.history['val_accuracy']  # Historique de la précision sur les␣
 ↪données de validation.
val_loss = history.history['val_loss']  # Historique de la perte sur les␣
 ↪données de validation.

# Trouver l'époque où la perte de validation est la plus basse et l'époque où␣
 ↪la précision de validation est la plus élevée.
index_loss = np.argmin(val_loss)  # Trouve l'indice (l'époque) où la perte de␣
 ↪validation est minimale.
val_lowest = val_loss[index_loss]  # Récupère la valeur de la perte la plus␣
 ↪basse.
index_acc = np.argmax(val_acc)  # Trouve l'indice (l'époque) où la précision de␣
 ↪validation est maximale.
acc_highest = val_acc[index_acc]  # Récupère la valeur de la précision la plus␣
 ↪haute.

# Crée une liste des époques pour l'axe x des graphiques.
Epochs = [i+1 for i in range(len(tr_acc))]  # Crée une liste des numéros␣
 ↪d'époques.
```

```
# Définit les labels pour les points de données les plus bas de la perte et les plus élevés de la précision.
loss_label = f'best epoch= {str(index_loss + 1)}'  # Etiquette de l'époque avec la perte de validation la plus basse.
acc_label = f'best epoch= {str(index_acc + 1)}'  # Etiquette de l'époque avec la précision de validation la plus élevée.

# Plot training history
plt.figure(figsize=(20, 8))  # Crée une figure avec une taille de 20x8 pouces pour les graphiques.
plt.style.use('fivethirtyeight')  # Applique un style de graphique prédéfini pour une apparence élégante.

# Sous-graphe 1: Perte d'entraînement et de validation
plt.subplot(1, 2, 1)  # Crée un sous-graphe dans une figure 1x2, première position.
plt.plot(Epochs, tr_loss, 'orange', label='Training loss')  # Trace la courbe de perte d'entraînement en orange.
plt.plot(Epochs, val_loss, 'purple', label='Validation loss')  # Trace la courbe de perte de validation en violet.
plt.scatter(index_loss + 1, val_lowest, s=150, c='red', label=loss_label)
plt.title('Training and Validation Loss')  # Titre du graphique de perte.
plt.xlabel('Epochs')  # Label de l'axe x (les époques).
plt.ylabel('Loss')  # Label de l'axe y (la perte).
plt.legend()  # Affiche la légende.

# Sous-graphe 2: Précision d'entraînement et de validation
plt.subplot(1, 2, 2)  # Crée un sous-graphe dans une figure 1x2, deuxième position.
plt.plot(Epochs, tr_acc, 'cyan', label='Training Accuracy')  # Trace la courbe de précision d'entraînement en cyan.
plt.plot(Epochs, val_acc, 'magenta', label='Validation Accuracy')  # Trace la courbe de précision de validation en magenta.
plt.scatter(index_acc + 1, acc_highest, s=150, c='green', label=acc_label)
plt.title('Training and Validation Accuracy')  # Titre du graphique de précision.
plt.xlabel('Epochs')  # Label de l'axe x (les époques).
plt.ylabel('Accuracy')  # Label de l'axe y (la précision).
plt.legend()  # Affiche la légende.

# Ajuste l'affichage pour éviter tout chevauchement des graphiques.
plt.tight_layout()  # Optimise l'agencement des sous-graphiques.
plt.show()  # Affiche les graphiques .
```



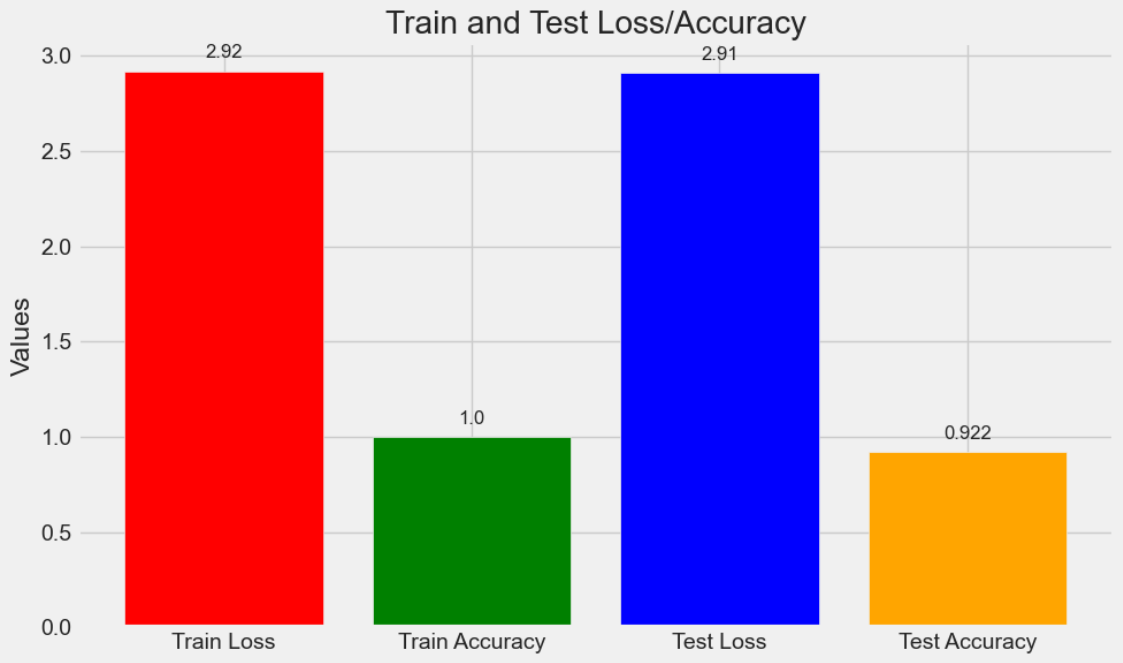
6 Évaluer le modèle

[27]:
```
train_score = model.evaluate(train_gen, verbose= 1)
test_score = model.evaluate(test_gen, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

```
26/26              8s 292ms/step -
accuracy: 1.0000 - loss: 2.9148
7/7              2s 260ms/step -
accuracy: 0.9424 - loss: 2.8936
Train Loss:  2.92035174369812
Train Accuracy:  1.0
--------------------
Test Loss:  2.9103198051452637
Test Accuracy:  0.9215686321258545
```

[28]:
```
import matplotlib.pyplot as plt  # Importation de la bibliothèque pour la␣
↪création de graphiques.
import numpy as np  # Importation de NumPy pour la manipulation de tableaux et␣
↪les opérations mathématiques.

# Train and Test scores from model evaluation
train_loss = train_score[0]  # Récupère la perte d'entraînement à partir des␣
↪scores d'entraînement.
train_acc = train_score[1]  # Récupère la précision d'entraînement à partir des␣
↪scores d'entraînement.
```

```
test_loss = test_score[0]  # Récupère la perte de test à partir des scores de␣
 ↪test.
test_acc = test_score[1]  # Récupère la précision de test à partir des scores␣
 ↪de test.

# Set up the figure and axes for the bar graph
fig, ax = plt.subplots(figsize=(10, 6))  # Crée une figure et un sous-graphe␣
 ↪(axe) avec une taille de 10x6 pouces.

# Bar positions
x_pos = np.arange(4)  # Définit les positions des barres sur l'axe des x␣
 ↪(quatre positions pour quatre valeurs).

# Data for the bars
values = [train_loss, train_acc, test_loss, test_acc]  # Liste des valeurs à␣
 ↪afficher sous forme de barres.
labels = ['Train Loss', 'Train Accuracy', 'Test Loss', 'Test Accuracy']  #␣
 ↪Labels des barres sur l'axe des x.
colors = ['red', 'green', 'blue', 'orange']  # Couleurs des barres pour chaque␣
 ↪catégorie (perte et précision).

# Create the bar chart
ax.bar(x_pos, values, color=colors)  # Crée le graphique à barres avec les␣
 ↪positions, les valeurs et les couleurs définies.

# Set the labels and title
ax.set_ylabel('Values')  # Définir l'étiquette de l'axe des y (valeurs).
ax.set_title('Train and Test Loss/Accuracy')  # Définir le titre du graphique.
ax.set_xticks(x_pos)  # Définir les positions des ticks de l'axe des x (les␣
 ↪emplacements où les labels seront placés).
ax.set_xticklabels(labels)  # Définir les étiquettes pour les ticks de l'axe␣
 ↪des x (les labels des barres).

# Display the value on top of each bar
for i, v in enumerate(values):  # Parcours de chaque valeur dans la liste␣
 ↪'values'.
    ax.text(x_pos[i], v + 0.05, str(round(v, 3)), ha='center', va='bottom',␣
 ↪fontsize=12)


# Show the plot
plt.tight_layout()
plt.show()  # Affiche le graphique.
```



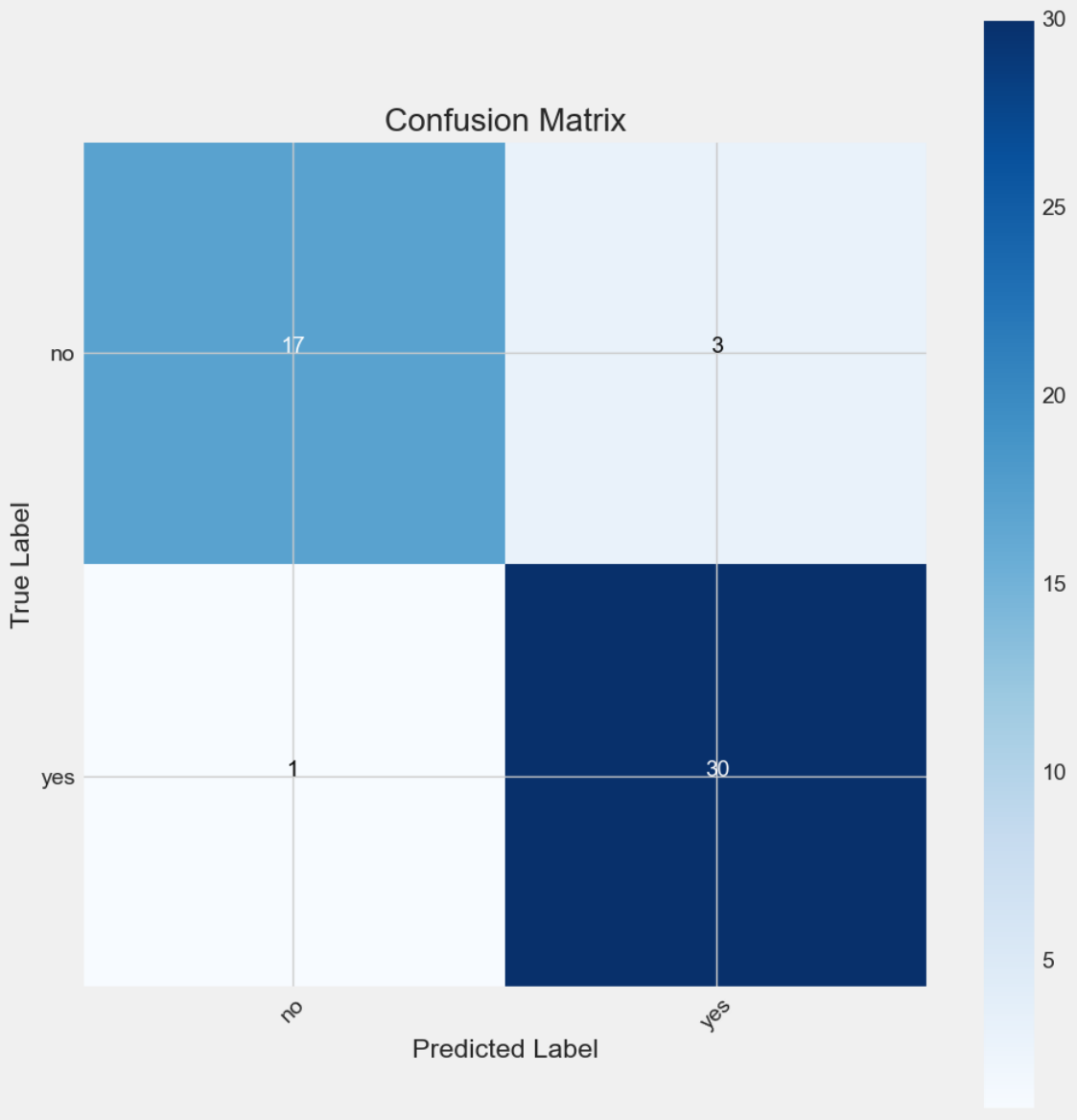
7 Obtenir les prédictions

[30]:
```
# Use model.predict instead of predict_generator
'''
 'predict' fonctionne directement avec un générateur de données,
    il effectue des prédictions sur le générateur de données de test
'''
preds = model.predict(test_gen)

# Get predicted class labels
'''
np.argmax récupère l'indice de la classe ayant la probabilité maximale pour␣
 ↪chaque prédiction,
ce qui permet de déterminer la classe prédite à partir des probabilités
'''
y_pred = np.argmax(preds, axis=1)
```

```
7/7             3s 386ms/step
```

8 Confusion Matrics and Classification Report

[34]:
```
import itertools  # permet d'itérer facilement sur des combinaisons de valeurs, comme les indices de la matrice de confusion.

# Confusion matrix
cm = confusion_matrix(test_gen.classes, y_pred)  # Calcule la matrice de confusion en comparant les vraies classes.

plt.figure(figsize=(10, 10))  # Crée une figure de taille 10x10 pour afficher la matrice de confusion.
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')  # Définit le titre du graphique.
plt.colorbar()  # Ajoute une barre de couleur à côté de la matrice pour indiquer l'échelle des valeurs.

tick_marks = np.arange(len(classes))  # Crée un tableau d'indices pour les classes (les positions sur l'axe x et y).
plt.xticks(tick_marks, classes, rotation=45)  # Ajoute les étiquettes des classes sur l'axe x avec une rotation de 45°.
plt.yticks(tick_marks, classes)  # Ajoute les étiquettes des classes sur l'axe y.
# Détermine le seuil pour l'affichage des valeurs dans la matrice, en fonction de la valeur maximale de la matrice.
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  # Itère sur chaque élément de la matrice de confusion.
    plt.text(j, i, cm[i, j], horizontalalignment='center', color='white' if cm[i, j] > thresh else 'black')

plt.tight_layout()  # Ajuste l'agencement du graphique pour qu'il soit bien espacé et visible.
plt.ylabel('True Label')  # Ajoute un label pour l'axe y.
plt.xlabel('Predicted Label')  # Ajoute un label pour l'axe x.

plt.show()  # Affiche la matrice de confusion.
```



[35]:
```
import pandas as pd  # Importation de la bibliothèque pandas pour manipuler les␣
 ↪données sous forme de DataFrame.
from sklearn.metrics import classification_report

# Génération du rapport de classification
report = classification_report(test_gen.classes, y_pred, target_names=classes,␣
 ↪output_dict=True)

# Conversion du rapport en DataFrame
report_df = pd.DataFrame(report).transpose()
```

```
# Convertit le rapport de classification (sous forme de dictionnaire) en un␣
 ↪DataFrame.

# Arrondir les valeurs à 2 décimales
report_df = report_df.round(2)
# Arrondit les valeurs dans le DataFrame à 2 décimales pour une meilleure␣
 ↪lisibilité.

# Affichage du rapport de classification sous forme de tableau
print("Classification Report (Tabular Form):")
# Affiche un titre indiquant que le rapport de classification sera présenté␣
 ↪sous forme de tableau.

#  Styliser le tableau pour améliorer son apparence visuelle
styled_report = report_df.style.background_gradient(cmap='Blues')

styled_report  # Affiche le tableau.
```

```
Classification Report (Tabular Form):
```

[35]:
```
<pandas.io.formats.style.Styler at 0x1a639cb0710>
```

9 Enregistrer le modèle

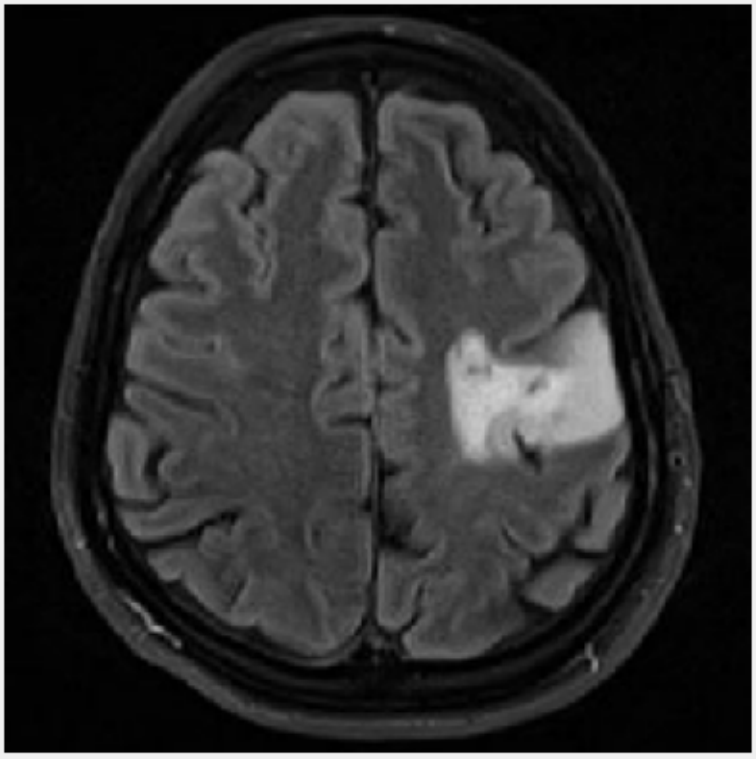
[36]:
```
model.save('C:/Users/Zakaria Taoubi/Model.keras')
```

10 Tester le module

[37]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Charger une image depuis un fichier
img_path = 'Y50.jpg'  # Remplacez par le chemin de votre image
img = image.load_img(img_path, target_size=(224, 224))  # Redimensionner␣
 ↪l'image à la taille de votre modèle

# Convertir l'image en tableau NumPy et normaliser (si nécessaire)
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Ajouter une dimension pour le␣
 ↪lot
img_array = img_array / 255.0  # Si votre modèle a été formé avec des images␣
 ↪normalisées entre 0 et 1

# Faire la prédiction
```

```
preds = model.predict(img_array)

# Trouver la classe prédite
predicted_class = np.argmax(preds, axis=1)

# Afficher l'image
plt.imshow(img)
plt.axis('off')
plt.show()

# Afficher la classe prédite
class_names = list(test_gen.class_indices.keys())  # Les noms de classe que␣
 ↪vous avez dans votre générateur de données
predicted_class_name = class_names[predicted_class[0]]

print(f"Classe prédite : {predicted_class_name}")
```

```
1/1             4s 4s/step
```



```
Classe prédite : yes
```

[39]:
```
import numpy as np
import tensorflow as tf
```

```
from tensorflow.keras.preprocessing import image
from IPython.display import display
import ipywidgets as widgets
from PIL import Image
import io

# Create file upload widget
upload_widget = widgets.FileUpload(accept='image/*', multiple=False)

def upload_and_predict(change):
    # Get the uploaded image (since upload_widget.value is a tuple, access the␣
 ↪first element)
    uploaded_image = list(upload_widget.value)[0]  # Access the first uploaded␣
 ↪file
    img_data = uploaded_image['content']

    # Convert the image data into an Image object
    img = Image.open(io.BytesIO(img_data))

    # Resize image to match model input size (assuming model expects 224x224)
    img = img.resize((224, 224))

    # Preprocess the image for the model
    img_array = np.array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Predict with the model
    predictions = model.predict(img_array)

    # Get the class with the highest probability
    predicted_class = np.argmax(predictions, axis=1)

    # Get the class label (make sure to adjust the class names accordingly)
    class_labels = ['No Tumor', 'Tumor']  # Replace with your actual class␣
 ↪labels
    predicted_label = class_labels[predicted_class[0]]

    print(f"Predicted Class: {predicted_label}")

# Show the upload widget
upload_widget.observe(upload_and_predict, names='value')
display(upload_widget)
```

```
FileUpload(value=(), accept='image/*', description='Upload')
```

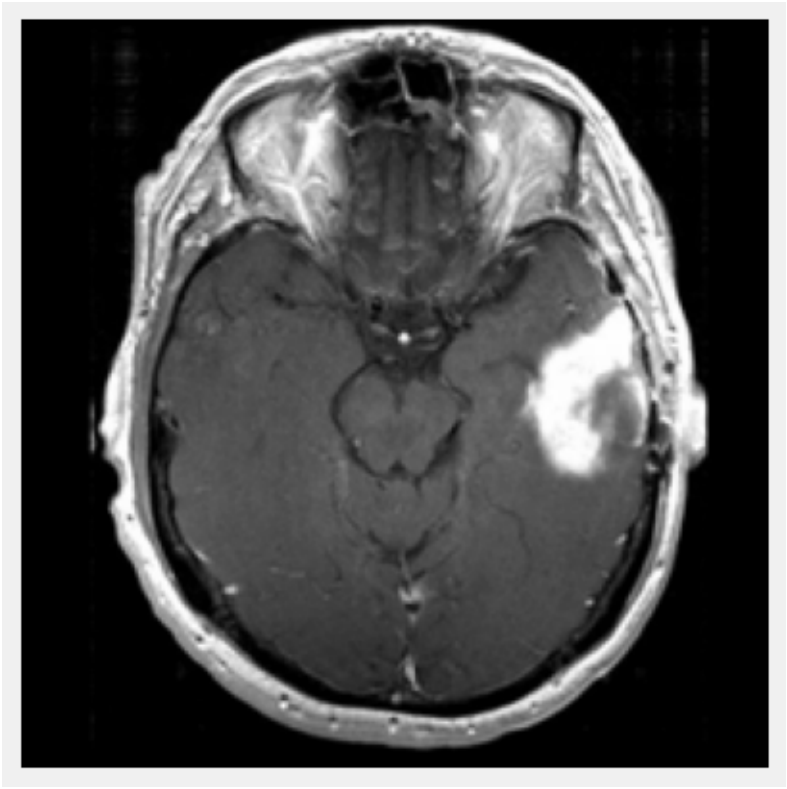
[43]:
```
import ipywidgets as widgets
from IPython.display import display
```

```
import PIL.Image as Image
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import io

# Load the trained model
model = tf.keras.models.load_model('C:/Users/Zakaria Taoubi/Model.keras')

# Create the file upload widget
upload_widget = widgets.FileUpload(accept='image/*', multiple=False)

# Function to handle the image upload and prediction
def upload_and_predict(change):
    # Get the uploaded image
    uploaded_image = list(upload_widget.value)[0]  # Access the first image␣
 ↪directly
    img_data = uploaded_image['content']

    # Convert the image data into an Image object
    img = Image.open(io.BytesIO(img_data))

    # Preprocess the image (resize, convert to array, etc.)
    img = img.resize((224, 224))  # Adjust the size according to your model␣
 ↪input
    img_array = np.array(img) / 255.0  # Normalize image
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Predict the class
    prediction = model.predict(img_array)
    predicted_class = 'No Tumor' if np.argmax(prediction) == 0 else 'Tumor'  #␣
 ↪Adjust based on your class labels

    # Display the result
    plt.imshow(img)
    plt.axis('off')
    plt.show()

    # Print the prediction result
    print(f"Predicted Class: {predicted_class}")
    print(f"Prediction Confidence: {np.max(prediction):.2f}")

# Attach the function to the widget
upload_widget.observe(upload_and_predict, names='value')

# Display the upload widget
```

```
display(upload_widget)
```

```
FileUpload(value=(), accept='image/*', description='Upload')
```

```
WARNING:tensorflow:5 out of the last 13 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x000001A73832D440> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1             4s 4s/step
```



```
Predicted Class: Tumor
Prediction Confidence: 0.66
```

Les résultats ont montré une précision de 66 % dans la classification des images, ce qui est un bon début, mais pourrait être amélioré avec un entraînement plus prolongé.

Résultats :

Le modèle a atteint un taux de précision de 66 % après 30 époques d'entraînement. Ce résultat,

bien qu'encourageant, pourrait être amélioré avec davantage d'époques et une optimisation supplémentaire des hyperparamètres. L'augmentation des données et l'utilisation de techniques de régularisation pourraient également contribuer à améliorer la robustesse du modèle.

Conclusion :

Dans le cadre du module Application clinique en radiodiagnostics, j'ai proposé d'explorer l'impact de l'intelligence artificielle, et plus spécifiquement du deep learning, dans le domaine du radiodiagnostics. L'utilisation de modèles comme EfficientNetB3 permet d'automatiser la détection des tumeurs, ce qui peut réduire le temps de diagnostic et améliorer la précision des détections. Cette approche pourrait transformer la manière dont les diagnostics sont réalisés en clinique, en offrant des outils plus rapides et plus fiables pour les professionnels de santé. leur intégration dans les outils cliniques existants, et leur utilisation pour assister les médecins dans leurs prises de décision, contribuant ainsi à améliorer la qualité des soins et l'efficacité du traitement des patients.

10.1 Projet 2 : Courbe d'étalonnage Unité Hounsfield Densité électronique

Objectifs :

- Étalonnage des nombres CT.
- Tracer la courbe de correspondance des Unité Hounsfield et des densités électroniques.
- Vérifier linéarité des nombres CT pour des matériaux de différentes atténuations.

Matériel et méthode :

- Logiciel de traitement et d'analyse d'image « MicroDicom».
- Python
- Carte d'atténuation du fantôme CIRS 062
- Image du fantôme CIRS 062
- Acquisition tomodensitométrique à 120 kv du fantôme CIRS 062M contenant différents inserts de différentes densités équivalents aux tissus humain.

[2]:
```
from IPython.display import display, Image, Markdown
image_path = "ISSSS.jpg"
display(Image(filename=image_path))
caption = "Figure 1: Disposition des inserts dans le fantôme de densité␣
 ↪électronique CT CIRS modèle 062M"
display(Markdown(f"**{caption}**"))
```

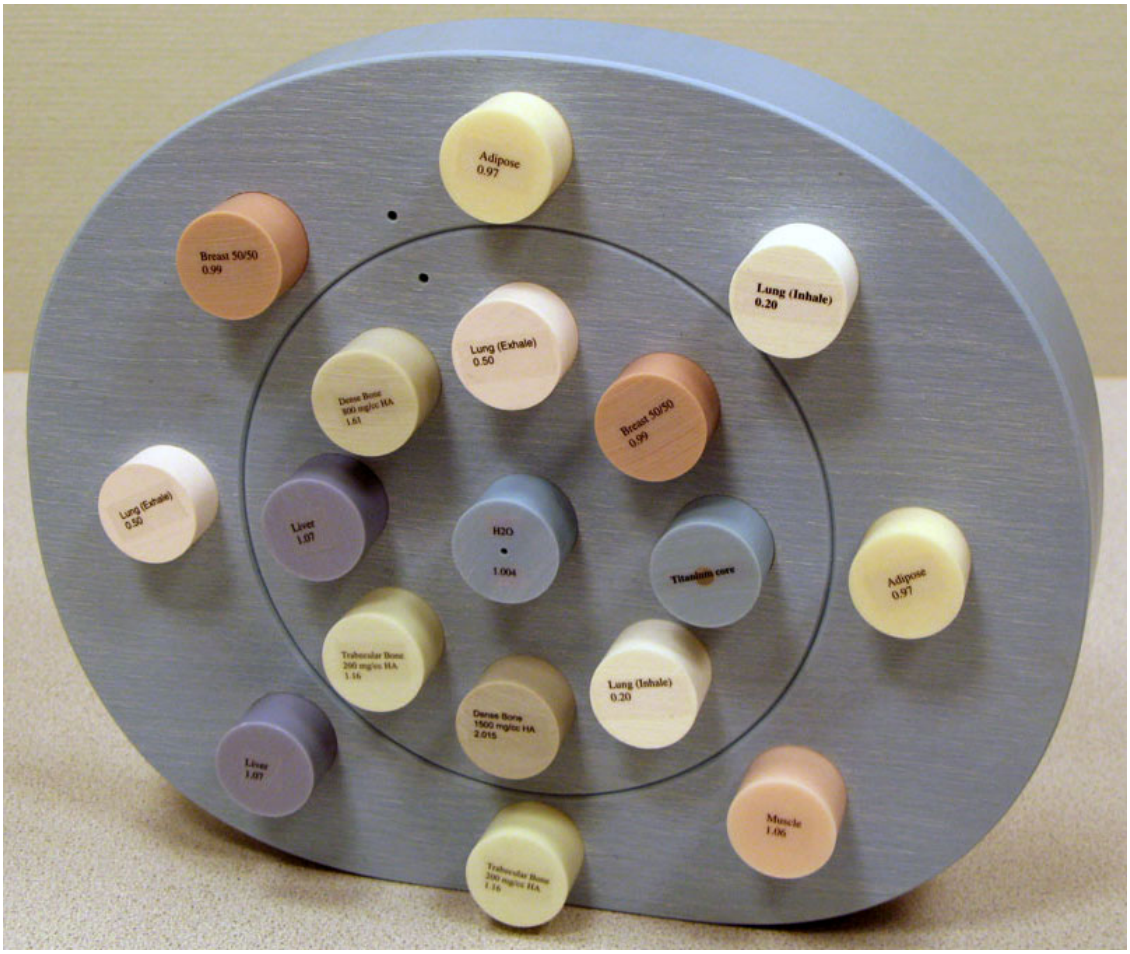


Figure 1: Disposition des inserts dans le fantôme de densité électronique CT CIRS modèle 062M

1- Sur l'image du fantôme, on relève les valeurs des densités électroniques correspondant aux différents tissus.

2- Sur le logiciel, on importe le dossier contenant les images DICOM du fantôme.

On visualise une des coupes axiales

[3]:
```
from IPython.display import display, Image, Markdown
image_path = "Taoubi25.jpeg"
display(Image(filename=image_path))
caption = "Figure 2 : Carte d'atténuation du fantôme CIRS 062 en coupe axiale"
display(Markdown(f"**{caption}**"))
```

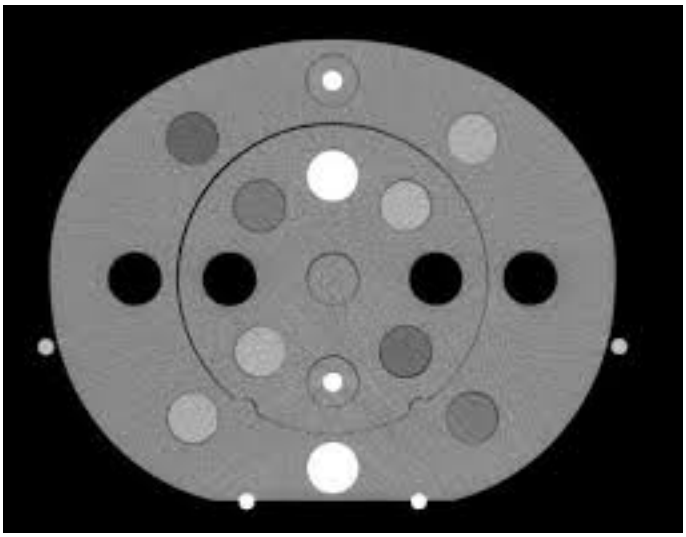


Figure 2 : Carte d'atténuation du fantôme CIRS 062 en coupe axiale

[5]:
```
from IPython.display import display, Image, Markdown
image_path = "Zakaria25.png"
display(Image(filename=image_path))
caption = "Figure 3 : Micro Dicom"
display(Markdown(f"**{caption}**"))
```

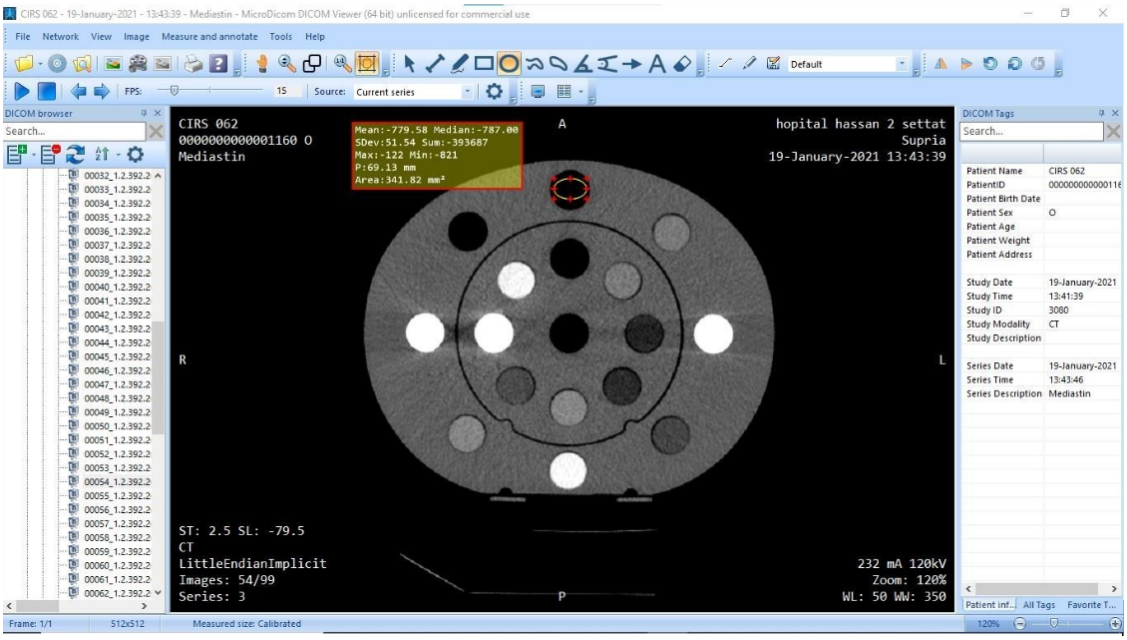


Figure 3 : Micro Dicom

MicroDicom : est une application pour le traitement primaire et la conservation des images médicales au format DICOM. Il est possible d'ouvrir des images DICOM produites par des équipements médicaux (IRM, PET, CT, …). Il est également possible d'ouvrir d'autres formats d'images - BMP, GIF, JPEG, PNG, TIFF, Il est équipé des outils les plus courants pour la Manipulation des images DICOM et dispose d'une interface utilisateur intuitive

- On délimite des régions d'intérêts « des ROIs » à l'aide du logiciel de taille comprise entre 200 et 400 centrés sur chaque insert puis on mesure les UH qui lui est correspond.

[6]:
```
import pandas as pd

# Data
data = {
    "Tissus": [
        "LUNG INHAL", "LUNG INHAL", "BREAST 50/50", "LUNG EXHAL", "LUNG EXHAL",
        "MUSCLE", "ADIPOSE", "BREAST 50/50", "ADIPOSE", "MUSCLE",
        "LIVER", "LIVER", "BONE 200", "BONE 200", "BONE 800",
        "BONE 800", "BONE 1250"
    ],
    "UH": [
        -781.875, -790.384, -21.8599, -515.373, -489.119,
        43.547, -35.153, -22.237, -1.081, 41.071,
        55.417, 56.135, 215.666, 219.482, 854.266,
        890.164, 1212.393
    ],
    "Densité Electronique": [
        6.34E+22, 6.34E+22, 3.48E+23, 1.63E+23, 1.63E+23,
        3.48E+23, 3.17E+23, 3.26E+23, 3.17E+23, 3.48E+23,
        3.52E+23, 3.52E+23, 3.73E+23, 3.73E+23, 4.86E+23,
        4.86E+23, 5.66E+23
    ]
}

df = pd.DataFrame(data)

df.style.format({"UH": "{:.2f}", "Densité Electronique": "{:.2e}"})
```

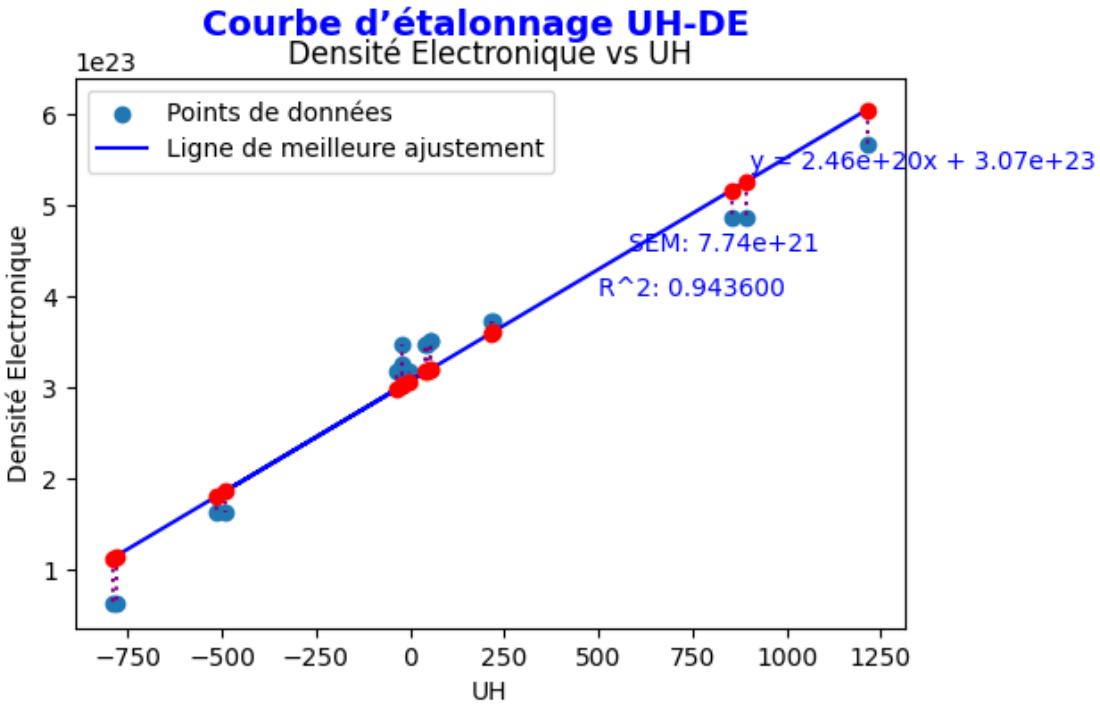
[6]:
```
<pandas.io.formats.style.Styler at 0x2ceef368e90>
```

[8]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import pearsonr

# Charger les données
tissues = ['Poumon inhalé', 'Poumon inhalé', 'Breast', 'Poumon Exhalé', 'Poumon␣
 ↪Exhalé', 'Muscle', 'Adpose',
           'Breast', 'Adpose', 'Muscle', 'Liver', 'Liver', 'Bone 200 mg/cc',␣
 ↪'Bone 200 mg/cc', 'Bone 800 mg/cc',
```

```
          'Bone 800 mg/cc', 'Bone 1250 mg/cc']
uh = np.array([-781.875, -790.384, -21.8599, -515.373, -489.119, 43.547, -35.
 ↪153, -22.237, -1.081, 41.071, 55.417,
               56.135, 215.666, 219.482, 854.266, 890.164, 1212.393])
de = np.array([6.34e22, 6.34e22, 3.483e23, 1.632e23, 1.632e23, 3.483e23, 3.
 ↪17e23, 3.261e23, 3.17e23, 3.483e23, 3.516e23,
               3.516e23, 3.73e23, 3.73e23, 4.862e23, 4.862e23, 5.663e23])

# Calculer la ligne de meilleure ajustement
m, b = np.polyfit(uh, de, 1)

# Calculer les valeurs prédites
predicted = m * uh + b

# Calculer l'erreur standard de la moyenne (SEM)
sem = np.std(de - predicted) / np.sqrt(len(de))

# Créer une figure de taille réduite
fig, ax = plt.subplots(figsize=(6, 4))

# Tracer les points de données et la ligne de meilleure ajustement
ax.scatter(uh, de, label='Points de données')  # Points de données
ax.plot(uh, m * uh + b, label='Ligne de meilleure ajustement', color='blue')  #␣
 ↪Ligne de meilleure ajustement
ax.errorbar(uh, predicted, yerr=sem, fmt='o', color='red')  # Barres d'erreur␣
 ↪sur les valeurs prédites
for i, j in zip(uh, de):
    ax.plot([i, i], [m * i + b, j], color='purple', linestyle='dotted')  #␣
 ↪Lignes pointillées pour montrer les résidus
ax.set_xlabel('UH')  # Etiquette pour l'axe des abscisses
ax.set_ylabel('Densité Electronique')  # Etiquette pour l'axe des ordonnées
ax.set_title('Densité Electronique vs UH')  # Titre du graphique

# Calculer le coefficient de corrélation
r, p = pearsonr(uh, de)
print('Coefficient de corrélation R^2:', r**2)

# Afficher l'équation sous forme scientifique et d'autres statistiques
ax.text(900, 5.4e23, f'y = {m:.2e}x + {b:.2e}', color='blue')  # Equation de la␣
 ↪droite d'ajustement
ax.text(580, 4.5e23, f"SEM: {sem:.2e}", color='blue')  # Erreur standard de la␣
 ↪moyenne
ax.text(500, 4e23, f"R^2: {r**2:.6f}", color='blue')  # Coefficient de␣
 ↪détermination
ax.legend()  # Afficher la légende
```

```
# Ajouter un titre global
fig.suptitle('Courbe d'étalonnage UH-DE', fontsize=14, fontweight='bold',␣
 ↪color='blue')

plt.savefig('TP2_ISSS_Settat.png')

plt.show()
```

```
Coefficient de corrélation R^2: 0.9435996786811934
```



10.1.1 Conclusion :

Pour conclure cette analyse, les résultats montrent une relation linéaire claire entre l'unité Hounsfield (UH) et la densité électronique (DE) pour les tissus étudiés. En particulier :

Coefficient de détermination élevé : Le coefficient R^2 de 1 indique une forte corrélation entre UH et DE. Cela confirme que l'unité Hounsfield peut être un prédicteur fiable de la densité électronique pour ces matériaux biologiques et structures corporelles.

Erreurs faibles : L'erreur standard de la moyenne (SEM) calculée est faible, ce qui suggère une bonne précision de la ligne d'ajustement dans la prédiction de DE à partir de UH.

Signification clinique: La relation obtenue peut avoir des applications pratiques, comme l'amélioration de la planification de la dose en radiothérapie, en permettant une estimation plus précise de la densité électronique en fonction de l'imagerie scanner (UH).

En résumé, cette courbe d'étalonnage UH-DE fournit un modèle robuste qui pourrait aider à simplifier et à affiner l'évaluation des densités électroniques dans les contextes cliniques, ce qui est essentiel pour une meilleure précision en radiothérapie et d'autres applications médicales.