A. Le premier problème auquel nous avons été confrontés était la labélisation des données en deux catégories : Tumor et Healthy.

B. Le deuxième problème auquel nous avons été confrontés était de mélanger les données pour éviter un biais de distribution et améliorer la généralisation du modèle.

C. Le troisième problème auquel nous avons été confrontés était de diviser les données en train_set, val_set et test_set de manière équilibrée pour assurer des performances fiables lors de la validation et du test du modèle.

A. La labélisation des images dépend de la façon dont votre dataset est organisé et de l'outil que vous utilisez. Voici plusieurs méthodes courantes :

1. Labélisation par Structure de Dossiers

Si vos images sont organisées en dossiers, chaque dossier représentant une classe, la labélisation se fait automatiquement dans certains frameworks (comme Keras ou PyTorch). Par exemple, si vous avez une structure comme ceci :

```
dataset/
├── train/
│   ├── Tumor/
│   │   ├── img1.jpg
│   │   ├── img2.jpg
│   │   └── ...
│   └── Healthy/
│       ├── img1.jpg
│       ├── img2.jpg
│       └── ...
├── test/
│   ├── Tumor/
│   └── Healthy/
```

Les images dans le dossier Tumor recevront automatiquement le label Tumor, et celles dans Healthy recevront le label Healthy lors de leur chargement avec des outils comme ImageDataGenerator de Keras ou ImageFolder de PyTorch.

1.a. Labélisation Automatique avec ImageDataGenerator de Keras

En utilisant ImageDataGenerator dans Keras, la structure des dossiers permet de générer des labels automatiquement.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1.0/255, validation_split=0.2)

train_set = datagen.flow_from_directory(
    'dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',  # ou 'categorical' pour plusieurs classes
    subset='training'
)

# Les labels sont automatiquement assignés en fonction des dossiers
print(train_set.class_indices)    #   Affiche   les   indices   des   classes   :
{'Healthy': 0, 'Tumor': 1}
```

1.b. Labélisation avec ImageFolder de PyTorch

Dans PyTorch, ImageFolder fonctionne de manière similaire à Keras en générant les labels automatiquement à partir des noms des dossiers.

```
from torchvision import datasets, transforms

transform       =       transforms.Compose([transforms.Resize((224,       224)),
transforms.ToTensor()])

dataset = datasets.ImageFolder('dataset/train', transform=transform)

# Les labels sont assignés automatiquement
print(dataset.class_to_idx)  # Affiche : {'Healthy': 0, 'Tumor': 1}
```

2. Labélisation Manuelle

Si vos images ne sont pas organisées dans des dossiers, vous pouvez créer manuellement une liste de labels correspondants. Par exemple :

```
# Supposons que vous ayez une liste d'images et leurs labels
images = ['img1.jpg', 'img2.jpg', 'img3.jpg']
labels = [0, 1, 0]  # 0 pour 'Tumor', 1 pour 'Healthy'

# Associez chaque image avec son label
dataset = list(zip(images, labels))
```

3. Labélisation avec des Outils de Labélisation (LabelImg, Roboflow, etc.)

Pour des images nécessitant une détection d'objet ou une segmentation, vous pouvez utiliser des outils comme LabelImg ou des plateformes comme Roboflow. Ces outils permettent de générer des fichiers d'annotations (par exemple, en format YOLO, COCO, Pascal VOC) qui contiennent les informations de labélisation.

LabelImg : Idéal pour annoter des images pour des tâches de classification ou de détection d'objets. Vous obtenez des fichiers .xml (format Pascal VOC) ou .txt (format YOLO) qui stockent les labels et les coordonnées de détection.

Roboflow : Vous permet de préparer, diviser et exporter votre dataset dans différents formats.

B. Pour mélanger (shuffle) les données après la labélisation, voici plusieurs méthodes en fonction du format dans lequel vos données sont stockées.

1. Utiliser shuffle de scikit-learn

Si vos données sont sous forme de listes (images et labels dans deux listes séparées), vous pouvez utiliser shuffle de scikit-learn pour mélanger les deux listes de manière synchrone.

```
from sklearn.utils import shuffle

# Listes d'images et de labels
images = [...]   # Liste d'images
labels = [...]   # Liste de labels correspondants

# Mélanger de manière synchrone
images, labels = shuffle(images, labels, random_state=42)
```

2. Utiliser shuffle de Pandas pour un DataFrame

Si vos données sont dans un DataFrame (par exemple, avec des colonnes pour les chemins d'images et les labels), vous pouvez utiliser la méthode sample pour mélanger les lignes.

```
import pandas as pd

# Chargement des données dans un DataFrame
df = pd.DataFrame({
    'image_path': ['img1.jpg', 'img2.jpg', 'img3.jpg'],
    'label': [0, 1, 0]
})

# Mélanger le DataFrame
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
```

3. Mélange pour ImageDataGenerator de Keras

Si vous utilisez ImageDataGenerator avec flow_from_directory, vous pouvez définir shuffle=True pour mélanger les images à chaque itération du générateur :

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1.0/255, validation_split=0.2)

train_set = datagen.flow_from_directory(
    'path/to/dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training',
    shuffle=True  # Mélange automatiquement les données
)
```

4. Mélange pour DataLoader de PyTorch

Si vous utilisez PyTorch, vous pouvez activer le mélange dans le DataLoader en définissant shuffle=True.

```
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# Charger le dataset et appliquer des transformations
transform = transforms.Compose([transforms.Resize((224, 224)),
transforms.ToTensor()])
dataset = datasets.ImageFolder('path/to/dataset', transform=transform)

# Créer le DataLoader avec shuffle
train_loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

5. Mélange Manuel avec random.shuffle

Si vous souhaitez mélanger manuellement deux listes en conservant l'ordre, vous pouvez utiliser random.shuffle avec des index aléatoires.

```
import random

# Listes d'images et de labels
images = [...]   # Liste d'images
labels = [...]   # Liste de labels correspondants

# Mélanger les listes de manière synchrone
combined = list(zip(images, labels))
random.shuffle(combined)
images[:], labels[:] = zip(*combined)
```

C. **En général, diviser les données en train_set, val_set et test_set peut être fait en même temps que la labellisation et le mélange des données.**

Pour un dataset organiser de cette manière :

```
dataset/
├── train/
│   ├── Tumor/
│   │   ├── img1.jpg
│   │   ├── img2.jpg
│   │   └── ...
│   └── Healthy/
│       ├── img1.jpg
│       ├── img2.jpg
│       └── ...
├── test/
│   ├── Tumor/
│   └── Healthy/
```

Option 1: Avec ImageDataGenerator de Keras

Keras gère automatiquement les labels selon la structure des dossiers et permet de mélanger les données et de les diviser en ensemble d'entraînement et de validation.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Définir des générateurs pour chaque ensemble
datagen_train = ImageDataGenerator(rescale=1.0/255)
datagen_val = ImageDataGenerator(rescale=1.0/255)
datagen_test = ImageDataGenerator(rescale=1.0/255)

# Chargement des ensembles
train_set = datagen_train.flow_from_directory(
    'Dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=True
)
```

```
validation_set = datagen_val.flow_from_directory(
    'Dataset/validation',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

test_set = datagen_test.flow_from_directory(
    'Dataset/test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Pas de shuffle pour garder l'ordre dans les tests
)

print(train_set.class_indices)  # Affiche : {'Healthy': 0, 'Tumor': 1}
```

Option 2: Avec ImageFolder et DataLoader de PyTorch

Avec PyTorch, vous pouvez utiliser ImageFolder pour la labélisation automatique des images. Ensuite, random_split permet de diviser les données en entraînement et en test et permet de mélanger les données.

```
from torchvision import datasets, transforms
from torch.utils.data import random_split, DataLoader

# Prétraitement pour redimensionner et transformer en tenseurs
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Charger le dataset complet avec labélisation automatique
dataset = datasets.ImageFolder('Dataset', transform=transform)

# Définir les tailles pour chaque ensemble
train_size = int(0.6 * len(dataset))
val_size = int(0.2 * len(dataset))
test_size = len(dataset) - train_size - val_size

# Diviser le dataset en train, validation et test
train_set, val_set, test_set = random_split(dataset, [train_size, val_size,
test_size])

# Créer les DataLoaders pour chaque ensemble
train_loader = DataLoader(train_set, batch_size=32, shuffle=True)
val_loader = DataLoader(val_set, batch_size=32, shuffle=True)
```

```
test_loader = DataLoader(test_set, batch_size=32, shuffle=False)

# Afficher les indices des classes
print(dataset.class_to_idx)  # Affiche : {'Healthy': 0, 'Tumor': 1}
```

Option 3 : Faire tous indépendamment de la structure du fichier

Par exemple une structure de fichier comme ça :

dataset/

├── Tumor/

│

└── Healthy/

Étape 1 : Charger les chemins d'images et leurs labels

Chargez les chemins des images et assignez les labels en fonction des sous-dossiers (Tumor et Healthy).

Utilisez ensuite train_test_split pour diviser en trois ensembles.

```
import os
import numpy as np
from sklearn.model_selection import train_test_split

# Définir les chemins de chaque sous-dossier
tumor_dir = 'Dataset/Tumor'
healthy_dir = 'Dataset/Healthy'

# Créer les listes de chemins d'images et de labels
image_paths = []
labels = []

# Charger les images de "Tumor" avec le label 1
for filename in os.listdir(tumor_dir):
    image_paths.append(os.path.join(tumor_dir, filename))
    labels.append(1)  # Label pour 'Tumor'

# Charger les images de "Healthy" avec le label 0
for filename in os.listdir(healthy_dir):
    image_paths.append(os.path.join(healthy_dir, filename))
    labels.append(0)  # Label pour 'Healthy'

# Convertir en tableaux numpy pour faciliter la division
image_paths = np.array(image_paths)
labels = np.array(labels)
```

Étape 2 : Diviser les données en train, validation et test

Utilisez train_test_split deux fois : une première fois pour créer les ensembles train et temp (validation + test), puis une seconde fois pour diviser temp en validation et test.

```
# Diviser en train et temp (80% train, 20% temp)
image_paths_train, image_paths_temp, labels_train, labels_temp =
train_test_split(
    image_paths, labels, test_size=0.2, random_state=42, stratify=labels
)

# Diviser temp en validation et test (50% validation, 50% test)
image_paths_val, image_paths_test, labels_val, labels_test = train_test_split(
        image_paths_temp, labels_temp, test_size=0.5, random_state=42,
stratify=labels_temp
)

# Résumé des tailles
print("Train size:", len(image_paths_train))
print("Validation size:", len(image_paths_val))
print("Test size:", len(image_paths_test))
```

Étape 3 : Créer des DataGenerators Personnalisés pour les Ensembles

Vous pouvez maintenant créer des générateurs pour chaque ensemble à partir des chemins d'images en utilisant ImageDataGenerator avec la méthode flow_from_dataframe.

```
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Convertir les chemins et labels en DataFrames
train_df = pd.DataFrame({'filename': image_paths_train, 'class':
labels_train})
val_df = pd.DataFrame({'filename': image_paths_val, 'class': labels_val})
test_df = pd.DataFrame({'filename': image_paths_test, 'class': labels_test})

# Initialiser ImageDataGenerator
datagen = ImageDataGenerator(rescale=1.0/255)

# Générateurs pour train, validation et test
train_gen = datagen.flow_from_dataframe(train_df, x_col='filename',
y_col='class',
                                        target_size=(224, 224), batch_size=32,
class_mode='binary', shuffle=True)
val_gen = datagen.flow_from_dataframe(val_df, x_col='filename', y_col='class',
```

```
                                        target_size=(224, 224), batch_size=32,
class_mode='binary', shuffle=True)
test_gen      =      datagen.flow_from_dataframe(test_df,      x_col='filename',
y_col='class',
                                        target_size=(224, 224), batch_size=32,
class_mode='binary', shuffle=False)

# Les labels sont déjà intégrés en fonction de la structure des dossiers
print(train_gen.class_indices)  # Affiche : {'Healthy': 0, 'Tumor': 1}
```