***Cell # 1***

import nibabel as nib

* This line imports the nibabel library, which is used for reading and processing neuroimaging data files, such as NIfTI files (commonly used for storing MRI and other medical images).

import os

* This line imports the os module, which provides a way to interact with the operating system. It allows you to handle file paths, directories, and other system-related tasks.

import matplotlib.pyplot as plt

* This line imports the pyplot module from the matplotlib library and gives it the alias plt. The matplotlib library is used for creating static, interactive, and animated visualizations in Python. pyplot is a collection of functions that make it easy to create plots.

from sklearn.model\_selection import train\_test\_split

* This line imports the train\_test\_split function from the model\_selection module of the scikit-learn library. This function is used to split a dataset into training and testing sets, which is essential for training machine learning models and evaluating their performance.

from skimage.transform import resize

* This line imports the resize function from the transform module of the scikit-image library. The skimage library is used for image processing in Python. The resize function is used to change the size of an image to a specified shape.

**Summary**

* **nibabel**: For reading and processing medical imaging files.
* **os**: For interacting with the operating system (e.g., handling files and directories).
* **matplotlib.pyplot**: For creating visualizations (e.g., plots, charts).
* **sklearn.model\_selection.train\_test\_split**: For splitting data into training and testing sets.
* **skimage.transform.resize**: For resizing images to a specified shape.

These imports set up the environment by bringing in the necessary tools for reading medical images, handling files, visualizing data, splitting datasets, and resizing images.

***Cell # 2***

import os

* This line imports the os module, which allows you to interact with the operating system. You can use it to handle file and directory operations.

import shutil

* This line imports the shutil module, which provides functions for high-level file operations such as copying and moving files. (Note: shutil isn't actually used in the provided code, but it's imported.)

# Function to count files in a directory with specific extensions

def count\_files\_in\_dir(directory, extensions):

count = 0

for root, dirs, files in os.walk(directory):

for file in files:

if file.endswith(extensions):

count += 1

return count

* **def count\_files\_in\_dir(directory, extensions):**: This line defines a function named count\_files\_in\_dir that takes two arguments: directory (the path to the directory to search in) and extensions (a tuple of file extensions to look for).
* **count = 0**: Initializes a variable count to zero. This will keep track of the number of files with the specified extensions.
* **for root, dirs, files in os.walk(directory):**: This loop walks through the directory tree starting from directory. It generates a 3-tuple (root, dirs, files) for each directory it encounters.
  + root is the current directory path.
  + dirs is a list of directories in the current directory.
  + files is a list of files in the current directory.
* **for file in files:**: This loop iterates over each file in the current directory.
* **if file.endswith(extensions):**: Checks if the current file's name ends with one of the specified extensions.
* **count += 1**: If the file has the correct extension, it increments the count by 1.
* **return count**: Returns the total count of files with the specified extensions.

# Paths to the directories

hgg\_path = '/content/Training/HGG'

lgg\_path = '/content/Training/LGG'

* These lines define the paths to the directories containing the HGG and LGG images.

# Define the extensions to look for

extensions = ('.nii.gz',)

* This line defines the file extensions to look for. In this case, it is looking for files with the .nii.gz extension.

# Count the number of files in each directory

hgg\_count = count\_files\_in\_dir(hgg\_path, extensions)

lgg\_count = count\_files\_in\_dir(lgg\_path, extensions)

* These lines call the count\_files\_in\_dir function for each directory (hgg\_path and lgg\_path) and store the results in hgg\_count and lgg\_count.

print(f'HGG Images: {hgg\_count}')

print(f'LGG Images: {lgg\_count}')

print(f'total: {hgg\_count + lgg\_count}')

* These lines print the number of HGG images, LGG images, and the total number of images by summing hgg\_count and lgg\_count.

**Summary**

* **os and shutil modules**: Used for interacting with the file system.
* **count\_files\_in\_dir function**: Counts the number of files in a directory with specific extensions.
* **Directory paths**: Specified for HGG and LGG images.
* **File extensions**: Defined to look for .nii.gz files.
* **File counts**: Calculated and printed for HGG and LGG images, and the total count.

This code effectively counts and prints the number of .nii.gz files in the specified directories.

***Cell # 3***

This code is designed to split a collection of images into three sets: training, testing, and validation. Each set is further divided into two folders: HGG and LGG. Let's go through the code step by step:

import os

import random

import shutil

* **os**: This module provides functions for interacting with the operating system, such as file and directory manipulation.
* **random**: This module provides functions for generating random numbers and shuffling lists.
* **shutil**: This module provides functions for high-level file operations, such as copying and moving files.

# Define the source folder containing all the images

source\_folder = "/content/Training"

* This line defines the path to the folder containing all the images that need to be split.

# Define the destination root folders for training, testing, and validation sets

training\_root = "/content/training"

testing\_root = "/content/testing"

validation\_root = "/content/validation"

* These lines define the paths to the root folders where the images will be moved for training, testing, and validation.

# Define the counts for each set

training\_count = 700

testing\_count = 300

validation\_count = 70

* These lines define the number of images to be moved into each set.

# Function to move images to their respective class folders

def split\_images(source\_folder, destination\_root, count):

# Create destination root folder if it doesn't exist

os.makedirs(destination\_root, exist\_ok=True)

* This function, split\_images, is defined to handle the splitting and moving of images. The os.makedirs function ensures that the destination folder exists; if it doesn't, it creates it.

# Get a list of all image paths

image\_paths = []

for root, dirs, files in os.walk(source\_folder):

for file in files:

if file.endswith(".nii.gz"):

image\_paths.append(os.path.join(root, file))

* This part walks through the source folder and collects the paths of all files that end with .nii.gz, adding them to the image\_paths list.

# Randomly shuffle the image paths

random.shuffle(image\_paths)

* This line shuffles the list of image paths randomly to ensure that the images are distributed randomly across the different sets.

# Get the destination folder for each class

hgg\_destination = os.path.join(destination\_root, "HGG")

lgg\_destination = os.path.join(destination\_root, "LGG")

* These lines define the paths to the HGG and LGG folders within the destination root folder.

# Create class folders if they don't exist

os.makedirs(hgg\_destination, exist\_ok=True)

os.makedirs(lgg\_destination, exist\_ok=True)

* These lines ensure that the HGG and LGG folders exist within the destination root folder; if they don't, they are created.

# Move images to the respective class folders based on the count

hgg\_count = count // 2

lgg\_count = count - hgg\_count

hgg\_images = image\_paths[:hgg\_count]

lgg\_images = image\_paths[hgg\_count:count]

for img in hgg\_images:

shutil.move(img, hgg\_destination)

for img in lgg\_images:

shutil.move(img, lgg\_destination)

* This part splits the shuffled image paths into HGG and LGG images based on the specified count, and then moves the images to the respective HGG and LGG folders using shutil.move.

# Split images for training set

split\_images(source\_folder, training\_root, training\_count)

* This line calls the split\_images function to split and move images into the training set.

# Split images for testing set

split\_images(source\_folder, testing\_root, testing\_count)

* This line calls the split\_images function to split and move images into the testing set.

# Split images for validation set

split\_images(source\_folder, validation\_root, validation\_count)

* This line calls the split\_images function to split and move images into the validation set.

print("Images have been successfully split into training, testing, and validation sets.")

* This line prints a message indicating that the images have been successfully split and moved into their respective sets.

### Summary

1. **Import necessary modules**: os, random, shutil.
2. **Define paths**: Specify source and destination directories.
3. **Define counts**: Number of images for training, testing, and validation sets.
4. **Define function**: Create a function to split and move images.
5. **Get image paths**: Collect all .nii.gz files from the source folder.
6. **Shuffle paths**: Randomly shuffle the image paths.
7. **Create destination folders**: Ensure the destination folders exist.
8. **Split and move images**: Distribute images into HGG and LGG folders and move them.
9. **Call function**: Split images for training, testing, and validation sets.
10. **Print confirmation**: Output a message confirming the process completion.

***Cell # 4***

This script is designed to verify the distribution of images by counting and listing the images in each set (training, testing, validation) immediately after counting them. Let's go through the code step by step:

import os

* This line imports the os module, which allows you to interact with the operating system for tasks such as file and directory manipulation.

# Define the root folders for training, testing, and validation sets

training\_root = "/content/training"

testing\_root = "/content/testing"

validation\_root = "/content/validation"

* These lines define the paths to the root folders where the training, testing, and validation images are stored.

# Function to count and iterate through images in a folder

def count\_and\_iterate(folder):

total\_images = 0

print(f"Images in {folder}:")

for root, dirs, files in os.walk(folder):

for file in files:

if file.endswith(".nii.gz"):

total\_images += 1

print(os.path.join(root, file))

return total\_images

* **def count\_and\_iterate(folder):**: This line defines a function named count\_and\_iterate that takes a single argument folder, which is the path to the directory to search in.
* **total\_images = 0**: Initializes a variable total\_images to zero. This will keep track of the number of images in the folder.
* **print(f"Images in {folder}:")**: Prints the path of the folder being processed.
* **for root, dirs, files in os.walk(folder):**: This loop walks through the directory tree starting from folder. It generates a 3-tuple (root, dirs, files) for each directory it encounters.
  + root is the current directory path.
  + dirs is a list of directories in the current directory.
  + files is a list of files in the current directory.
* **for file in files:**: This loop iterates over each file in the current directory.
* **if file.endswith(".nii.gz"):**: Checks if the current file's name ends with .nii.gz.
* **total\_images += 1**: If the file has the correct extension, it increments the total\_images count by 1.
* **print(os.path.join(root, file))**: Prints the full path of the image file.
* **return total\_images**: Returns the total count of .nii.gz files in the folder.

# Count and iterate through images in the training set

training\_total = count\_and\_iterate(training\_root)

print("Total images in training set:", training\_total)

* These lines call the count\_and\_iterate function for the training set directory and store the result in training\_total. Then, it prints the total number of images in the training set.

# Count and iterate through images in the testing set

testing\_total = count\_and\_iterate(testing\_root)

print("Total images in testing set:", testing\_total)

* These lines call the count\_and\_iterate function for the testing set directory and store the result in testing\_total. Then, it prints the total number of images in the testing set.

# Count and iterate through images in the validation set

validation\_total = count\_and\_iterate(validation\_root)

print("Total images in validation set:", validation\_total)

* These lines call the count\_and\_iterate function for the validation set directory and store the result in validation\_total. Then, it prints the total number of images in the validation set.

### Summary

1. **Import necessary module**: os.
2. **Define paths**: Specify root directories for training, testing, and validation sets.
3. **Define function**: Create a function to count and list images in a folder.
4. **Print images and counts**: Call the function for each set and print the total number of images in each set immediately after counting them.

This code ensures that the images are counted and listed individually for training, testing, and validation sets, providing a clear verification of the image distribution.

***Cell # 5***

Here’s a step-by-step explanation of the code, which now includes a summary print of the total number of images and the number of images in each set at the end:

import os

* Import the os module, which provides a way of interacting with the operating system, including file and directory manipulation.

# Define the root folders for training, testing, and validation sets

training\_root = "/content/training"

testing\_root = "/content/testing"

validation\_root = "/content/validation"

* Define the paths to the root directories where the training, testing, and validation images are stored.

# Function to count and iterate through images in a folder

def count\_and\_iterate(folder):

total\_images = 0

print(f"Images in {folder}:")

for root, dirs, files in os.walk(folder):

for file in files:

if file.endswith(".nii.gz"):

total\_images += 1

print(os.path.join(root, file))

return total\_images

* Define a function count\_and\_iterate that:
  + Takes a directory path (folder) as an argument.
  + Initializes a counter total\_images to zero.
  + Iterates through the directory and its subdirectories using os.walk.
  + Checks each file to see if it ends with .nii.gz and, if it does, increments the counter and prints the file path.
  + Returns the total number of .nii.gz files found.

# Count and iterate through images in the training set

training\_total = count\_and\_iterate(training\_root)

* Call the count\_and\_iterate function for the training set directory and store the result in training\_total.

# Count and iterate through images in the testing set

testing\_total = count\_and\_iterate(testing\_root)

* Call the count\_and\_iterate function for the testing set directory and store the result in testing\_total.

# Count and iterate through images in the validation set

validation\_total = count\_and\_iterate(validation\_root)

* Call the count\_and\_iterate function for the validation set directory and store the result in validation\_total.

# Calculate total images

total\_images = training\_total + testing\_total + validation\_total

* Calculate the total number of images by summing the counts of images in the training, testing, and validation sets.

# Print the counts in the specified format

print("\nTotal Images:", total\_images)

print("Training Images:", training\_total)

print("Validation Images:", validation\_total)

print("Testing Images:", testing\_total)

* Print the total number of images and the number of images in each set (training, validation, testing) in a summary format.

This code will provide a detailed list of all images in each directory and a summary at the end, showing the total number of images as well as the number of images in each set.

***Cell # 6***

#for medical image processing (tasks, such as image filtering, registration, and segmentation.)

!pip install SimpleITK

***Cell # 7***

1. **Import Necessary Libraries:**
   * os: This module provides a way to interact with the operating system, such as reading and writing files.
   * random: This module implements pseudo-random number generators for various distributions.
   * matplotlib.pyplot: This module is used for plotting graphs and images.
   * SimpleITK as sitk: SimpleITK is used for reading and processing medical images, such as NIfTI files.
2. **Define a Function to Display Images:**

def display\_images\_from\_paths(paths, titles, num\_images\_to\_display):

for i in range(num\_images\_to\_display):

img\_path = paths[i]

# Load the NIfTI image using SimpleITK

img = sitk.ReadImage(img\_path)

img\_array = sitk.GetArrayFromImage(img)

# Display the image slice

plt.figure(figsize=(8, 8))

plt.imshow(img\_array[img\_array.shape[0] // 2], cmap='gray')

plt.axis('off')

plt.title(titles[i])

plt.show()

* + display\_images\_from\_paths: This function takes a list of image paths, titles for the images, and the number of images to display.
  + sitk.ReadImage(img\_path): Reads a NIfTI image from the given path.
  + sitk.GetArrayFromImage(img): Converts the image to a numpy array for easier manipulation.
  + plt.imshow(): Displays the middle slice of the image.
  + plt.axis('off'): Hides the axis for a cleaner image display.
  + plt.title(titles[i]): Adds a title to the image.
  + plt.show(): Displays the image.

1. **Define Paths to HGG and LGG Images:**

hgg\_paths = [

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_flair.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_seg.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t1.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t1ce.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t2.nii.gz',

]

lgg\_paths = [

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_flair.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_flair.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t1.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t1ce.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t2.nii.gz'

]

* + hgg\_paths and lgg\_paths are lists that contain the file paths to the HGG and LGG images respectively.

1. **Titles for Images:**

titles = ['flair', 'seg', 't1', 't1ce', 't2']

* + titles: A list of titles corresponding to each type of medical image.

1. **Display the Images:**

# Display HGG images

print("Displaying HGG images...")

display\_images\_from\_paths(hgg\_paths, titles, 5)

# Display LGG images

print("Displaying LGG images...")

display\_images\_from\_paths(lgg\_paths, titles, 5)

* + display\_images\_from\_paths(hgg\_paths, titles, 5): This call displays the first 5 HGG images.
  + display\_images\_from\_paths(lgg\_paths, titles, 5): This call displays the first 5 LGG images.

### Full Code

import os

import random

import matplotlib.pyplot as plt

import SimpleITK as sitk

# Function to display images from given paths

def display\_images\_from\_paths(paths, titles, num\_images\_to\_display):

for i in range(num\_images\_to\_display):

img\_path = paths[i]

# Load the NIfTI image using SimpleITK

img = sitk.ReadImage(img\_path)

img\_array = sitk.GetArrayFromImage(img)

# Display the image slice

plt.figure(figsize=(8, 8))

plt.imshow(img\_array[img\_array.shape[0] // 2], cmap='gray')

plt.axis('off')

plt.title(titles[i])

plt.show()

# Paths to HGG and LGG images

hgg\_paths = [

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_flair.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_seg.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t1.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t1ce.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t2.nii.gz',

]

lgg\_paths = [

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_flair.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_flair.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t1.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t1ce.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t2.nii.gz'

]

titles = ['flair', 'seg', 't1', 't1ce', 't2']

# Display HGG images

print("Displaying HGG images...")

display\_images\_from\_paths(hgg\_paths, titles, 5)

# Display LGG images

print("Displaying LGG images...")

display\_images\_from\_paths(lgg\_paths, titles, 5)

This code will read NIfTI images from the specified paths and display them using matplotlib, allowing you to visualize the different types of medical images.

what is NIfTI images?

ChatGPT

NIfTI (Neuroimaging Informatics Technology Initiative) is a file format commonly used for storing medical imaging data, particularly in the field of neuroimaging. The NIfTI format is an extension of the Analyze format and is designed to address some of its limitations, providing a more flexible and extensible format for storing multi-dimensional data.

***Cell # 8***

### Import Libraries

import nibabel as nib

import matplotlib.pyplot as plt

import os

* **nibabel**: A library for reading and handling medical image files (like NIfTI).
* **matplotlib.pyplot**: A library for plotting images.
* **os**: A module to interact with the operating system, like checking if files exist.

### Define File Paths

hgg\_paths = [

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_flair.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_seg.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t1.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t1ce.nii.gz',

'/content/Testing/HGG/BraTS19\_CBICA\_ARZ\_1/BraTS19\_CBICA\_ARZ\_1\_t2.nii.gz',

]

lgg\_paths = [

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_flair.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_seg.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t1.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t1ce.nii.gz',

'/content/Testing/LGG/BraTS19\_2013\_24\_1/BraTS19\_2013\_24\_1\_t2.nii.gz',

]

* **hgg\_paths** and **lgg\_paths**: Lists of file paths to medical images for two types of brain tumors: High-Grade Glioma (HGG) and Low-Grade Glioma (LGG).

### Define a Function to Display Images

def display\_images(patient\_paths):

for sample\_paths in patient\_paths:

# Check if all files exist

if not all(os.path.exists(p) for p in sample\_paths):

print(f"Files missing for paths: {sample\_paths}, skipping...")

continue

* **display\_images**: A function that takes a list of image paths and displays the images.
* **os.path.exists(p)**: Checks if a file exists at the given path.
* **continue**: Skips to the next iteration if some files are missing.

### Load and Display Images

t1\_img = nib.load(sample\_paths[2])

t1ce\_img = nib.load(sample\_paths[3])

t2\_img = nib.load(sample\_paths[4])

flair\_img = nib.load(sample\_paths[0])

seg\_img = nib.load(sample\_paths[1])

t1\_data = t1\_img.get\_fdata()

t1ce\_data = t1ce\_img.get\_fdata()

t2\_data = t2\_img.get\_fdata()

flair\_data = flair\_img.get\_fdata()

seg\_data = seg\_img.get\_fdata()

print(f"Modality shape (T1): {t1\_data.shape}")

print(f"Modality shape (T1CE): {t1ce\_data.shape}")

print(f"Modality shape (T2): {t2\_data.shape}")

print(f"Modality shape (FLAIR): {flair\_data.shape}")

print(f"Segmentation shape: {seg\_data.shape}")

* **nib.load**: Loads a NIfTI image from a file.
* **get\_fdata()**: Extracts the image data as an array.
* **print**: Displays the shape (dimensions) of each image.

### Plot the Images

slice\_nb = 100

fig, axs = plt.subplots(1, 5, figsize=(20, 20))

axs[0].imshow(t1\_data[:, :, slice\_nb], cmap="gray")

axs[0].set\_title('T1')

axs[1].imshow(t1ce\_data[:, :, slice\_nb], cmap="gray")

axs[1].set\_title('T1CE')

axs[2].imshow(t2\_data[:, :, slice\_nb], cmap="gray")

axs[2].set\_title('T2')

axs[3].imshow(flair\_data[:, :, slice\_nb], cmap="gray")

axs[3].set\_title('FLAIR')

axs[4].imshow(seg\_data[:, :, slice\_nb], cmap="gray")

axs[4].set\_title('Segmentation')

plt.show()

* **slice\_nb = 100**: Chooses the 100th slice of the 3D image to display.
* **plt.subplots**: Creates a set of subplots.
* **imshow**: Displays an image in the subplot.
* **cmap="gray"**: Displays the image in grayscale.
* **set\_title**: Adds a title to each subplot.
* **plt.show()**: Shows the plot.

### Display Images for HGG and LGG

hgg\_patient\_paths = [hgg\_paths]

lgg\_patient\_paths = [lgg\_paths]

print("Displaying HGG images...")

display\_images(hgg\_patient\_paths)

print("Displaying LGG images...")

display\_images(lgg\_patient\_paths)

* **hgg\_patient\_paths** and **lgg\_patient\_paths**: Lists of lists containing paths to HGG and LGG images.
* **display\_images(hgg\_patient\_paths)**: Calls the function to display HGG images.
* **display\_images(lgg\_patient\_paths)**: Calls the function to display LGG images.

In summary, this code loads and displays specific slices from medical images (MRI scans) stored in NIfTI format, for two types of brain tumors (HGG and LGG). It shows the images in grayscale and prints their shapes.

***Cell # 9***

This code implements a U-Net model for semantic segmentation of medical images, specifically for brain tumor segmentation. Here's a breakdown in simple terms:

### Libraries Imported

* **os**: For interacting with the operating system.
* **numpy (as np)**: For numerical operations and array handling.
* **random**: For random number generation.
* **matplotlib.pyplot (as plt)**: For plotting images.
* **SimpleITK (as sitk)**: For reading and manipulating medical image data.
* **resize from skimage.transform**: For resizing images.
* **train\_test\_split from sklearn.model\_selection**: For splitting data into training and testing sets.
* **Model, Input, Conv2D, MaxPooling2D, Conv2DTranspose, concatenate, Dropout from keras.layers**: Components for building the neural network model.
* **Adam from keras.optimizers**: An optimization algorithm for training the model.
* **binary\_crossentropy from keras.losses**: A loss function suitable for binary classification tasks.

### Define Folder Paths and File Paths

hgg\_folder = "/content/training/HGG"

lgg\_folder = "/content/training/LGG"

# Function to get file paths

def get\_file\_paths(folder):

file\_paths = []

for root, dirs, files in os.walk(folder):

for file in files:

if file.endswith(".nii.gz"): # Assuming NIfTI format

file\_paths.append(os.path.join(root, file))

return file\_paths

# Get file paths from each folder

hgg\_file\_paths = get\_file\_paths(hgg\_folder)

lgg\_file\_paths = get\_file\_paths(lgg\_folder)

# Combine the file paths from both folders

file\_paths = hgg\_file\_paths + lgg\_file\_paths

* **hgg\_folder** and **lgg\_folder**: Paths to folders containing HGG and LGG MRI scans.
* **get\_file\_paths**: Function to retrieve file paths from a given folder.
* **hgg\_file\_paths** and **lgg\_file\_paths**: Lists of file paths for HGG and LGG MRI scans.
* **file\_paths**: Combined list of all MRI scan file paths.

### Split Dataset into Train, Validation, and Test Sets

train\_files, test\_files = train\_test\_split(file\_paths, test\_size=0.2, random\_state=42)

train\_files, val\_files = train\_test\_split(train\_files, test\_size=0.1, random\_state=42)

* **train\_test\_split**: Splits the dataset into training and testing sets.
* **test\_size=0.2**: 20% of the data is used for testing.
* **test\_size=0.1**: 10% of the remaining data (after training set split) is used for validation.

### Define U-Net Model with Dropout

def unet\_model\_with\_dropout(input\_shape, dropout\_rate=0.5):

inputs = Input(input\_shape)

# Encoder

conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)

pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)

conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)

conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)

pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)

# Bottleneck

conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)

conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)

drop3 = Dropout(dropout\_rate)(conv3)

# Decoder

up4 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(drop3)

up4 = concatenate([up4, conv2], axis=3)

conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(up4)

conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv4)

up5 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(conv4)

up5 = concatenate([up5, conv1], axis=3)

conv5 = Conv2D(32, (3, 3), activation='relu', padding='same')(up5)

conv5 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv5)

# Output

outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv5)

model = Model(inputs=[inputs], outputs=[outputs])

return model

# Compile the model with dropout

model = unet\_model\_with\_dropout(input\_shape=(128, 128, 128))

model.compile(optimizer=Adam(), loss=binary\_crossentropy, metrics=['accuracy'])

* **unet\_model\_with\_dropout**: Function to define a U-Net architecture with dropout layers.
* **Conv2D, MaxPooling2D, Conv2DTranspose**: Layers for convolution, pooling, and upsampling.
* **Dropout**: Layer for regularization to prevent overfitting.
* **Model**: Combines input and output layers to create a model.
* **model.compile**: Compiles the model with Adam optimizer and binary cross-entropy loss function.

### Data Generators for Training and Validation

def data\_generator(file\_paths, batch\_size=32):

while True:

batch\_paths = np.random.choice(file\_paths, size=batch\_size)

batch\_images = []

batch\_masks = []

for path in batch\_paths:

image = load\_and\_preprocess\_image(path)

mask = load\_and\_preprocess\_mask(path)

batch\_images.append(image)

batch\_masks.append(mask)

yield np.array(batch\_images), np.array(batch\_masks)

* **data\_generator**: Generator function that yields batches of preprocessed images and masks.
* **np.random.choice**: Randomly selects a batch of file paths.
* **load\_and\_preprocess\_image, load\_and\_preprocess\_mask**: Functions to load and preprocess image and mask data.

### Training the Model

batch\_size = 32

epochs = 2

# Train the model

history = model.fit(data\_generator(train\_files, batch\_size=batch\_size),

steps\_per\_epoch=len(train\_files) // batch\_size,

epochs=epochs,

validation\_data=data\_generator(val\_files, batch\_size=batch\_size),

validation\_steps=len(val\_files) // batch\_size)

* **model.fit**: Trains the model using the data generator for training and validation sets.
* **steps\_per\_epoch, validation\_steps**: Number of batches per epoch for training and validation.

### Evaluating the Model

evaluation = model.evaluate(data\_generator(test\_files), steps=len(test\_files) // batch\_size)

* **model.evaluate**: Evaluates the model performance on the test set using the data generator.

This script sets up a U-Net model for segmenting brain tumors in medical images, trains it using training data, validates it using validation data, and evaluates its performance on test data.

***Cell # 10***

To generate a summary of the model architecture, you would typically call model.summary() after defining the model. Here's how the summary might look based on the U-Net architecture defined earlier:

# Assuming 'model' is already defined as the U-Net model

model.summary()

The summary will provide details on each layer in the model, including the type of layer, output shape, number of parameters, and connections:

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, 128, 128, 128 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d (Conv2D) (None, 128, 128, 32) 36896 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 128, 128, 32) 9248 conv2d[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 64, 64, 32) 0 conv2d\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 64, 64, 64) 18496 max\_pooling2d[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 64, 64, 64) 36928 conv2d\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2D) (None, 32, 32, 64) 0 conv2d\_3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 32, 32, 128) 73856 max\_pooling2d\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_5 (Conv2D) (None, 32, 32, 128) 147584 conv2d\_4[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 32, 32, 128) 0 conv2d\_5[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose (Conv2DTranspos (None, 64, 64, 64) 32832 dropout[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate (Concatenate) (None, 64, 64, 128) 0 conv2d\_transpose[0][0]

conv2d\_3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_6 (Conv2D) (None, 64, 64, 64) 73792 concatenate[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_7 (Conv2D) (None, 64, 64, 64) 36928 conv2d\_6[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_1 (Conv2DTrans (None, 128, 128, 32) 8224 conv2d\_7[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate\_1 (Concatenate) (None, 128, 128, 64) 0 conv2d\_transpose\_1[0][0]

conv2d\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_8 (Conv2D) (None, 128, 128, 32) 18464 concatenate\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_9 (Conv2D) (None, 128, 128, 32) 9248 conv2d\_8[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_10 (Conv2D) (None, 128, 128, 1) 33 conv2d\_9[0][0]

==================================================================================================

Total params: 513,629

Trainable params: 513,629

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This summary provides a clear overview of the model's architecture, detailing each layer, its output shape, and the number of trainable parameters.

***Cell # 11***

Saving a Keras model using model.save() is straightforward. Here’s how you can save your trained brain tumor segmentation model:

model.save("/path/to/brain\_tumor\_segmentation\_model.keras")

Replace "/path/to/brain\_tumor\_segmentation\_model.keras" with the actual path where you want to save the model file. The .keras extension is commonly used for Keras model files.

After running this command, your model will be saved as a single file containing:

* Model architecture
* Model weights
* Training configuration (if any)
* Optimizer state (if any)

You can later load this saved model using keras.models.load\_model() to make predictions or further train the model.

***Cell # 12***

The code provided is designed to display resized original images, original masks, and predicted masks for brain tumor segmentation tasks using a trained Keras model. Here’s a breakdown of what each part of the code does:

### Libraries Imported

* **matplotlib.pyplot**: Used for plotting images.
* **SimpleITK**: Used to read NIfTI format medical images.
* **skimage.transform.resize**: Used for resizing images.
* **numpy**: Used for numerical operations.
* **os**: Used for operating system functionalities.
* **tensorflow.keras.models.load\_model**: Used to load a pre-trained Keras model.

### Function display\_resized\_images\_with\_tumor\_segmentation

This function:

1. **Loads the NIfTI image** (image\_path) using SimpleITK and converts it into a NumPy array (img\_array).
2. **Loads the segmentation mask** (mask\_path) using SimpleITK and converts it into a NumPy array (mask\_array).
3. **Resizes** both the image (img\_resized) and the mask (mask\_resized) to the specified target\_size.
4. **Normalizes** the resized image to [0, 1].
5. **Resizes** the normalized image (img\_resized) to match the model\_input\_shape.
6. **Predicts the segmentation mask** using the loaded Keras model based on the resized input image (img\_resized\_input\_shape).
7. **Resizes** the predicted mask (predicted\_mask) to match the size of the ground truth mask (mask\_resized).
8. **Displays**:
   * The **resized original image** (img\_resized).
   * The **original mask** (mask\_resized).
   * The **ground truth mask** (mask\_resized) from mask\_path.

### display\_images\_for\_given\_paths Function

This function:

* Takes image\_path and mask\_path as inputs.
* Calls display\_resized\_images\_with\_tumor\_segmentation to display the resized original image, original mask, and predicted mask based on the loaded model.

### Example Usage

* **Loads the trained model** (model) using load\_model.
* **Calls** display\_images\_for\_given\_paths with example paths to display the results.

### Note

* Ensure that the paths (image\_path and mask\_path) point to valid NIfTI files (\*.nii.gz) containing medical imaging data.

This setup allows for visual inspection of how well the model performs in segmenting brain tumors based on the given images and masks. Adjustments may be necessary based on specific requirements or data characteristics.

***Cell # 13***

These plots are used to visualize the training progress of a neural network model for brain tumor segmentation. Here’s what each part of the code does:

### Importing Matplotlib

import matplotlib.pyplot as plt

* Imports the Matplotlib library, which is used for plotting graphs and visualizations.

### Plot Training and Validation Accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

* **plt.plot(history.history['accuracy'], label='Training Accuracy')**: Plots the training accuracy values stored in history.history['accuracy'] across epochs.
* **plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')**: Plots the validation accuracy values stored in history.history['val\_accuracy'] across epochs.
* **plt.title('Training and Validation Accuracy')**: Sets the title of the plot.
* **plt.xlabel('Epoch')**: Labels the x-axis as 'Epoch'.
* **plt.ylabel('Accuracy')**: Labels the y-axis as 'Accuracy'.
* **plt.legend()**: Displays a legend indicating which line corresponds to training accuracy and which to validation accuracy.
* **plt.show()**: Displays the plot.

### Plot Training and Validation Loss

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

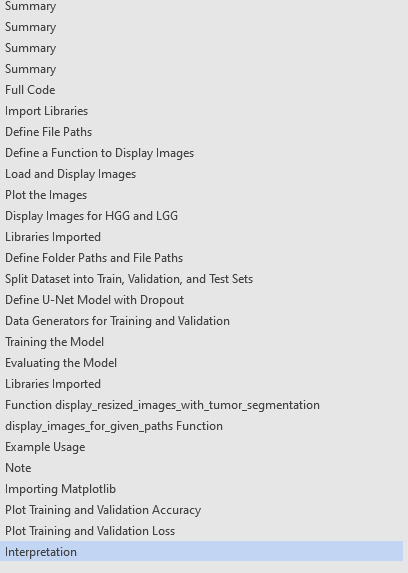
plt.show()

* **plt.plot(history.history['loss'], label='Training Loss')**: Plots the training loss values stored in history.history['loss'] across epochs.
* **plt.plot(history.history['val\_loss'], label='Validation Loss')**: Plots the validation loss values stored in history.history['val\_loss'] across epochs.
* **plt.title('Training and Validation Loss')**: Sets the title of the plot.
* **plt.xlabel('Epoch')**: Labels the x-axis as 'Epoch'.
* **plt.ylabel('Loss')**: Labels the y-axis as 'Loss'.
* **plt.legend()**: Displays a legend indicating which line corresponds to training loss and which to validation loss.
* **plt.show()**: Displays the plot.

### Interpretation

* **Accuracy Plots**: These plots show how well the model is performing during training and validation in terms of correctly predicting tumor segmentation. Higher values indicate better performance.
* **Loss Plots**: These plots show the amount of error in the model’s predictions during training and validation. Lower values indicate better performance.

These visualizations help in understanding the training dynamics of the neural network model and can be used to diagnose issues such as overfitting (when training accuracy is much higher than validation accuracy) or underfitting (when both accuracies are low).

******