Brain Tumor MRI Classification

This project demonstrates the classification of brain tumors using MRI images with deep learning techniques, based on the **"Brain Tumor MRI Dataset."**

Kaggle Dataset: Brain Tumor MRI Dataset

GitHub Repository: Brain Tumor MRI Classification Project

Table of Contents

Edwin P. Bayog Jr. BSCpE 3-A

CpETE1 Embedded System 1 - Realtime Systems

Technological University of the Philippines - Visayas

Note: You can run the project on both a local device and Google Colab, or jump directly to the `streamlit run` command to get started with the application right away.

1. Kaggle Installation

```
# Install the Kaggle library
!pip install kaggle --quiet

from google.colab import files

print("Please upload your kaggle.json file:")
uploaded = files.upload()

with open('kaggle.json', 'wb') as f:
    f.write(uploaded['kaggle.json'])

!mkdir -p ~/.kaggle
```

```
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
Please upload your kaggle.json file:
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

```
# Download the dataset in colab
!kaggle datasets download masoudnickparvar/brain-tumor-mri-dataset -p
/content/brain-tumor-mri-dataset --unzip
```

```
Dataset URL: https://www.kaggle.com/datasets/masoudnickparvar/brain-
tumor-mri-dataset
License(s): CC0-1.0
Downloading brain-tumor-mri-dataset.zip to /content/brain-tumor-mri-
dataset
 83% 124M/149M [00:00<00:00, 1.28GB/s]
100% 149M/149M [00:00<00:00, 1.20GB/s]
```

```
# Download the dataset locally
!kaggle datasets download masoudnickparvar/brain-tumor-mri-dataset -p
/brain-tumor-mri-dataset --unzip
```

Kaggle Installation, this section installs the Kaggle API to enable dataset downloads, handles authentication by securely uploading and configuring the `kaggle.json` token file, and uses the Kaggle CLI to download and extract the Brain Tumor MRI Dataset both in Google Colab (`/content/brain-tumor-mri-dataset`) and locally (`/brain-tumor-mri-dataset`).

2. Importing and Setup of Dependencies

```
# General Imports
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
import shutil
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Neural Network imports
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
```

```python
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input
from tensorflow.keras.optimizers import Adam

# Image augmentation importrs
from tensorflow.keras.utils import load_img
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import RandomRotation
from tensorflow.keras.layers import RandomContrast
from tensorflow.keras.layers import RandomZoom
from tensorflow.keras.layers import RandomFlip
from tensorflow.keras.layers import RandomTranslation

# Training Model callbacks
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import ModelCheckpoint

print(f'Tensorflow Version: {tf.__version__}')

SEED = 111

# Data Visualization updates
%config InlineBackend.figure_format = 'retina'
plt.rcParams["figure.figsize"] = (16, 10)
plt.rcParams.update({'font.size': 14})
```

```
Tensorflow Version: 2.18.0
```

```python
def get_data_labels(directory, shuffle=True, random_state=0):
    from sklearn.utils import shuffle
    data_path = []
    data_index = []
    label_dict = {label: index for index, label in
enumerate(sorted(os.listdir(directory)))}

    for label, index in label_dict.items():
        label_dir = os.path.join(directory, label)
        for image in os.listdir(label_dir):
            image_path = os.path.join(label_dir, image)
            data_path.append(image_path)
            data_index.append(index)

    if shuffle:
        data_path, data_index = shuffle(data_path, data_index,
random_state=random_state)

    return data_path, data_index

def parse_function(filename, label, image_size, n_channels):
    image_string = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image_string, n_channels)
    image = tf.image.resize(image, image_size)
    return image, label

def get_dataset(paths, labels, image_size, n_channels=1,
num_classes=4, batch_size=32):
    path_ds = tf.data.Dataset.from_tensor_slices((paths, labels))
    image_label_ds = path_ds.map(lambda path, label:
parse_function(path, label, image_size, n_channels),
                                 num_parallel_calls=tf.data.AUTOTUNE)
    return
image_label_ds.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE
)
```

Importing and Setup of Dependencies, this section imports essential libraries for data manipulation, visualization, and deep learning model development using TensorFlow and Keras. It also sets up global configurations like random seeds, image processing utilities, and custom functions for loading and preprocessing the dataset efficiently using `tf.data.Dataset`.

3. Importing Datasets with Validation Split

```
import os
import random
import shutil

USER_PATH = "/content/brain-tumor-mri-dataset"
TRAIN_DIR = os.path.join(USER_PATH, 'Training')
TEST_DIR = os.path.join(USER_PATH, 'Testing')
VAL_DIR = os.path.join(USER_PATH, 'Validation')

# Define the percentage to retain in Testing
TESTING_SPLIT_PERCENTAGE = 0.03  # Keep only 3% in Testing

# Create Validation directory and its subdirectories if they don't
exist
if not os.path.exists(VAL_DIR):
    os.makedirs(VAL_DIR)

for category in os.listdir(TEST_DIR):
    category_path_testing = os.path.join(TEST_DIR, category)
    category_path_validation = os.path.join(VAL_DIR, category)

    if not os.path.exists(category_path_validation):
        os.makedirs(category_path_validation)

    if os.path.isdir(category_path_testing):  # Ensure it's a
directory
        images = [img for img in os.listdir(category_path_testing) if
os.path.isfile(os.path.join(category_path_testing, img))]

        # Calculate number of images to keep in Testing
        num_test_images = max(1, int(len(images) *
TESTING_SPLIT_PERCENTAGE))  # Ensure at least 1 image remains
        num_val_images = len(images) - num_test_images

        # Randomly select images to move to Validation
        val_images_to_move = random.sample(images, num_val_images)

        # Move selected images to Validation
        for img_name in val_images_to_move:
            src_path = os.path.join(category_path_testing, img_name)
            dest_path = os.path.join(category_path_validation,
img_name)
            shutil.move(src_path, dest_path)

        print(f"Moved {len(val_images_to_move)} images from
Testing/{category} to Validation/{category}")

# Continue with the rest of your pipeline
train_paths, train_index = get_data_labels(TRAIN_DIR,
random_state=SEED)
test_paths, test_index = get_data_labels(TEST_DIR, random_state=SEED)
val_paths, val_index = get_data_labels(VAL_DIR, random_state=SEED)

# Output counts
print('\nTraining')
print(f'Number of Paths: {len(train_paths)}')
print(f'Number of Labels: {len(train_index)}')

print('\nTesting')
print(f'Number of Paths: {len(test_paths)}')
print(f'Number of Labels: {len(test_index)}')

print('\nValidation')
print(f'Number of Paths: {len(val_paths)}')
print(f'Number of Labels: {len(val_index)}')

# Dataset preparation
batch_size = 32
image_dim = (168, 168)  # height, width
train_ds = get_dataset(train_paths, train_index, image_dim,
n_channels=1, num_classes=4, batch_size=batch_size)
test_ds = get_dataset(test_paths, test_index, image_dim, n_channels=1,
num_classes=4, batch_size=batch_size)
val_ds = get_dataset(val_paths, val_index, image_dim, n_channels=1,
num_classes=4, batch_size=batch_size)

# Print dataset objects
print(f"\nTraining dataset: {train_ds}")
print(f"\nTesting dataset: {test_ds}")
print(f"\nValidation dataset: {val_ds}")
```

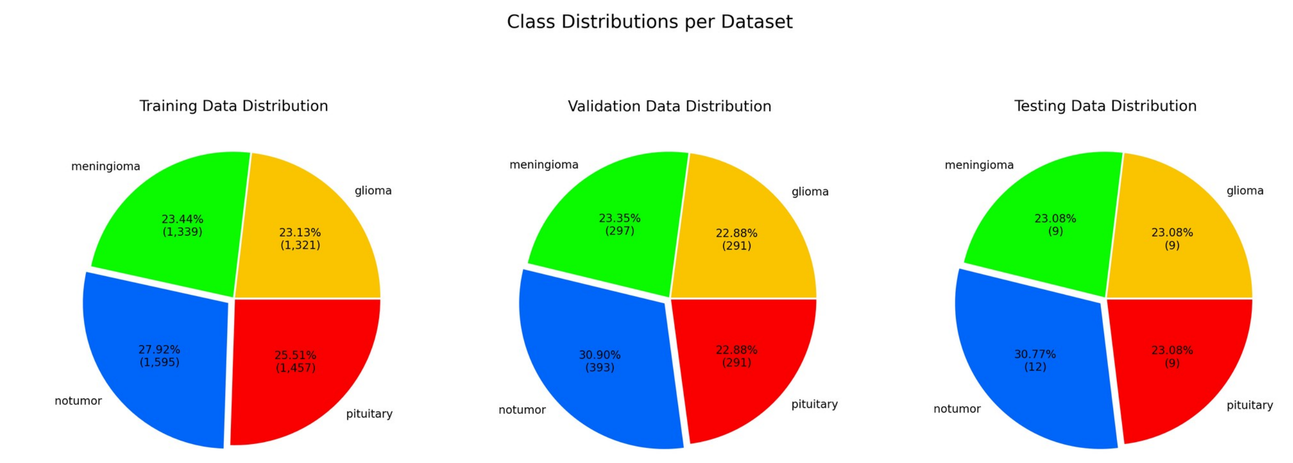
```
# Class mappings
class_mappings = {label: index for index, label in
enumerate(sorted(os.listdir(TRAIN_DIR)))}
inv_class_mappings = {v: k for k, v in class_mappings.items()}
class_names = list(class_mappings.keys())

print(f"\nClass Mappings used by get_data_labels: {class_mappings}")
print(f"Class Names for plotting (derived from mappings):
{class_names}")
```

```
Moved 393 images from Testing/notumor to Validation/notumor
Moved 291 images from Testing/pituitary to Validation/pituitary
Moved 297 images from Testing/meningioma to Validation/meningioma
Moved 291 images from Testing/glioma to Validation/glioma

Training
Number of Paths: 5712
Number of Labels: 5712

Testing
Number of Paths: 39
Number of Labels: 39

Validation
Number of Paths: 1272
Number of Labels: 1272

Training dataset: <_PrefetchDataset
element_spec=(TensorSpec(shape=(None, 168, 168, 1), dtype=tf.float32,
name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>

Testing dataset: <_PrefetchDataset
element_spec=(TensorSpec(shape=(None, 168, 168, 1), dtype=tf.float32,
name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>

Validation dataset: <_PrefetchDataset
element_spec=(TensorSpec(shape=(None, 168, 168, 1), dtype=tf.float32,
name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>

Class Mappings used by get_data_labels: {'glioma': 0, 'meningioma': 1,
'notumor': 2, 'pituitary': 3}
Class Names for plotting (derived from mappings): ['glioma',
'meningioma', 'notumor', 'pituitary']
```

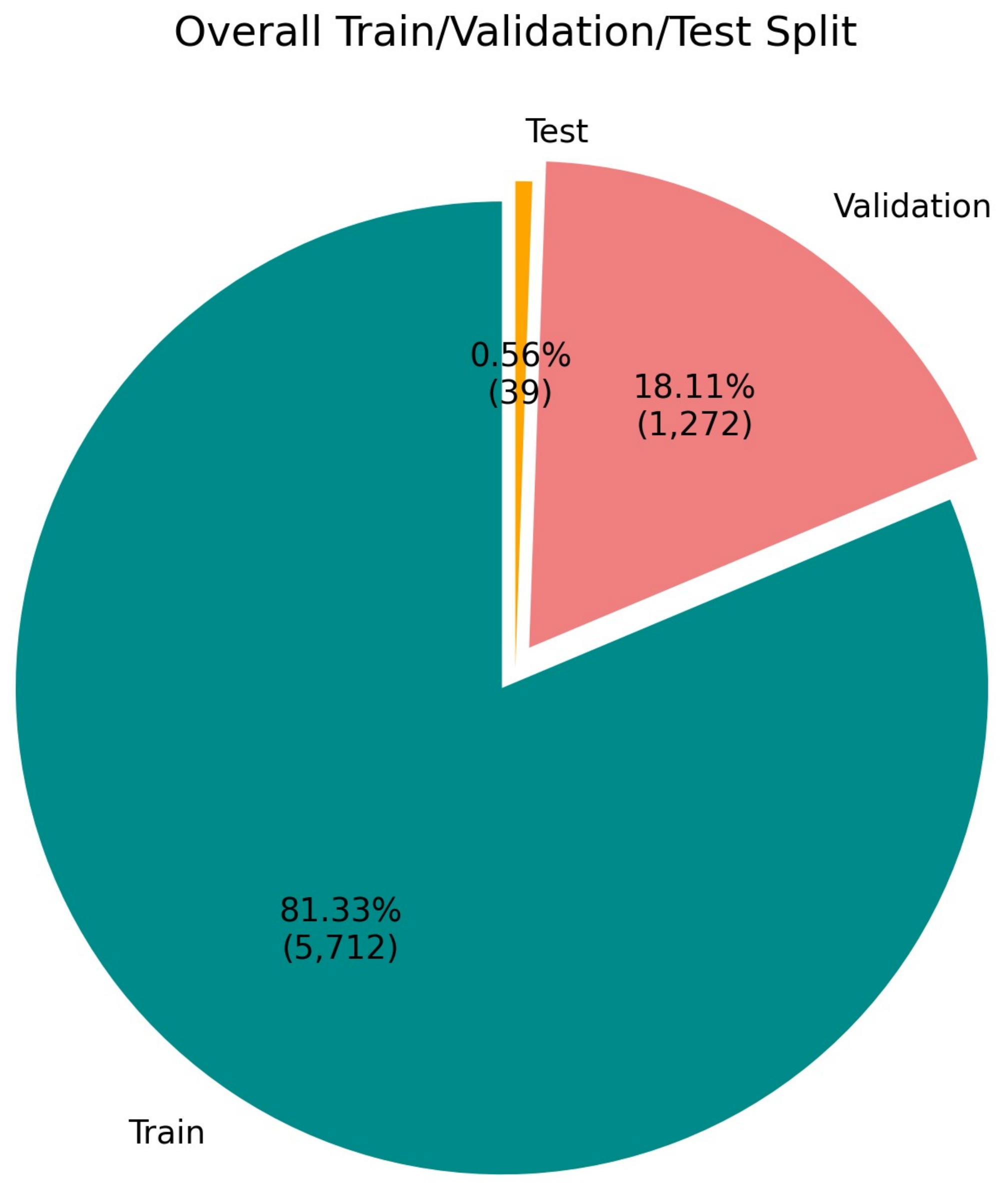
Importing Datasets with Validation Split, this section defines directory paths for training, testing, and validation datasets, and programmatically splits off a portion of the test data into a validation set based on a percentage split (minimum 3%), ensuring at least one image is moved if necessary. It then uses the `get_data_labels` function to extract file paths and labels from each dataset directory, prepares batched and prefetched TensorFlow datasets for training, validation, and testing using grayscale images resized to 168x168 pixels, and creates class mappings for label interpretation during model evaluation and visualization.

4. Data Visualization

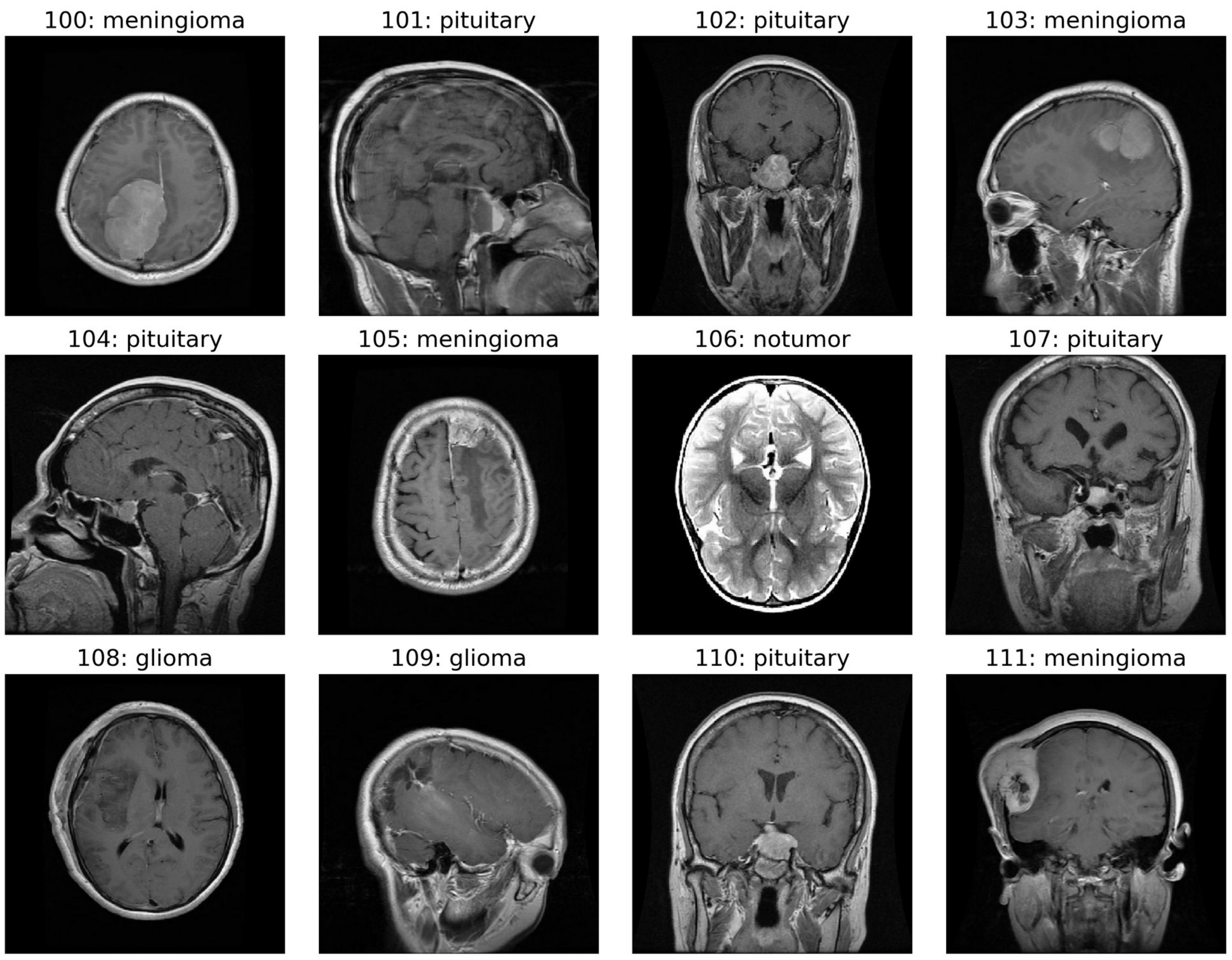
```
# Figure 1: Class Distributions (Training, Validation, Testing)
fig1, ax1 = plt.subplots(nrows=1, ncols=3, figsize=(24, 8))
plt.subplots_adjust(wspace=0.2)

train_class_counts = [len([x for x in train_index if x ==
class_mappings[name]]) for name in class_names]
ax1[0].set_title('Training Data Distribution', fontsize=16)
ax1[0].pie(
    train_class_counts,
    labels=class_names,
    colors=['#FAC500','#0BFA00', '#0066FA','#FA0000'],
    autopct=lambda p: '{:.2f}%\n({:,.0f})'.format(p, p *
sum(train_class_counts) / 100),
    explode=(0.01, 0.01, 0.05, 0.01),
    textprops={'fontsize': 12}
)

val_class_counts = [len([x for x in val_index if x ==
class_mappings[name]]) for name in class_names]
ax1[1].set_title('Validation Data Distribution', fontsize=16)
ax1[1].pie(
    val_class_counts,
    labels=class_names,
    colors=['#FAC500','#0BFA00', '#0066FA','#FA0000'],
    autopct=lambda p: '{:.2f}%\n({:,.0f})'.format(p, p *
sum(val_class_counts) / 100),
    explode=(0.01, 0.01, 0.05, 0.01),
    textprops={'fontsize': 12}
)

test_class_counts = [len([x for x in test_index if x ==
class_mappings[name]]) for name in class_names]
ax1[2].set_title('Testing Data Distribution', fontsize=16)
ax1[2].pie(
    test_class_counts,
    labels=class_names,
    colors=['#FAC500', '#0BFA00', '#0066FA', '#FA0000'],
    autopct=lambda p: '{:.2f}%\n({:,.0f})'.format(p, p *
sum(test_class_counts) / 100),
    explode=(0.01, 0.01, 0.05, 0.01),
    textprops={'fontsize': 12}
)

fig1.suptitle('Class Distributions per Dataset', fontsize=20, y=1.03)
plt.show()
```



```
# Figure 2: Train/Validation/Test Split
fig2, ax2 = plt.subplots(figsize=(10, 8))

split_counts = [len(train_index), len(val_index), len(test_index)]
split_labels = ['Train', 'Validation', 'Test']
split_colors = ['darkcyan', 'lightcoral', 'orange']
split_explode = (0.05, 0.05, 0)

ax2.set_title('Overall Train/Validation/Test Split', fontsize=18)
ax2.pie(
    split_counts,
    labels=split_labels,
    colors=split_colors,
    autopct=lambda p: '{:.2f}%\n({:,.0f})'.format(p, p *
sum(split_counts) / 100),
    explode=split_explode,
    startangle=90,
    textprops={'fontsize': 14}
)
fig2.tight_layout()
plt.show()
```



```
# Function to display a list of images based on the given index
def show_images(paths, label_paths, class_mappings,
index_list=range(10), im_size=250, figsize=(12, 8)):

    num_images = len(index_list)
    num_rows = (num_images + 3) // 4
    index_to_class = {v: k for k, v in class_mappings.items()}
    _, ax = plt.subplots(nrows=num_rows, ncols=4, figsize=figsize)
    ax = ax.flatten()

    for i, index in enumerate(index_list):
        if i >= num_images:
            break
        image = load_img(paths[index], target_size=(im_size, im_size),
color_mode='grayscale')
        ax[i].imshow(image, cmap='Greys_r')
        class_name = index_to_class[label_paths[index]]
        ax[i].set_title(f'{index}: {class_name}')
        ax[i].axis('off')

    plt.tight_layout()
    plt.show()

# Four images from different angles
show_images(train_paths, train_index, class_mappings, im_size=350,
figsize=(13,10),
            index_list=range(100, 112))
```

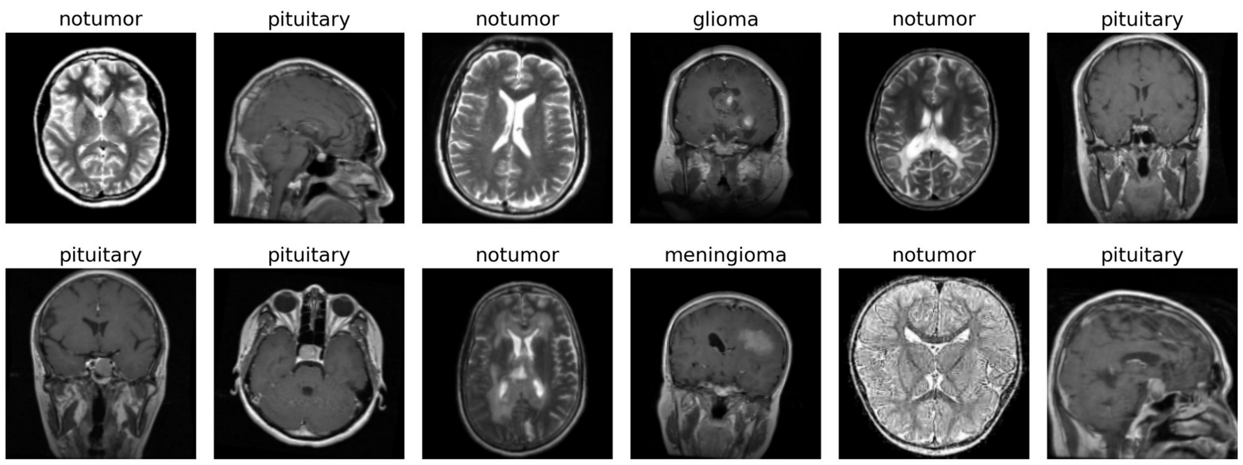


Data Visualization, this section visualizes the class distribution across training, validation, and testing datasets using pie charts to show the proportion of each tumor type (`glioma`, `meningioma`, `notumor`, `pituitary`) within them, and also displays an overall pie chart of the train/validation/test split ratio, helping to confirm balanced data distribution and proper dataset partitioning for model training and evaluation. It also defines a function to display grayscale MRI images with their corresponding labels for qualitative inspection, helping to verify correct data loading and class assignments during preprocessing.

5. Data Preprocessing & Training Setup Values

```
# Data augmentation sequential model
data_augmentation = Sequential([
    # RandomFlip("horizontal_and_vertical"),
    RandomFlip("horizontal"),
    RandomRotation(0.02, fill_mode='constant'),
    RandomContrast(0.1),
    RandomZoom(height_factor=0.01, width_factor=0.05),
    RandomTranslation(height_factor=0.0015, width_factor=0.0015,
fill_mode='constant'),
])

# Training augmentation and nornalization
def preprocess_train(image, label):
    # Apply data augmentation and Normalize
    image = data_augmentation(image) / 255.0
    return image, label

# For validation dataset only appying normalization
def preprocess_val(image, label):
    return image / 255.0, label

# Apply transformation to training and validation datasets
train_ds_preprocessed = train_ds.map(preprocess_train,
num_parallel_calls=tf.data.AUTOTUNE)
val_ds_preprocessed = val_ds.map(preprocess_val,
num_parallel_calls=tf.data.AUTOTUNE)

# Function to display augmented images
def plot_augmented_images(dataset, shape, class_mappings, figsize=(15,
6)):
    plt.figure(figsize=figsize)
    index_to_class = {v: k for k, v in class_mappings.items()}
    for images, label in dataset.take(1):
        i = 0
        for i in range(shape[0]*shape[1]):
            ax = plt.subplot(shape[0], shape[1], i + 1)
            plt.imshow(images[i].numpy().squeeze(), cmap='gray')
            plt.title(index_to_class[label.numpy()[i]])
            plt.axis("off")
            i += 1

    plt.tight_layout()
    plt.show()
```

```
# Displaying augmented images
plot_augmented_images(train_ds_preprocessed, shape=(2, 6),
class_mappings=class_mappings)
```



```
# Classes and Image shape: height, width, grayscale
num_classes = len(class_mappings.keys())
image_shape = (image_dim[0], image_dim[1], 1)

# Training epochs and batch size
epochs = 50
print(f'Number of Classes: {num_classes}')
print(f'Image shape: {image_shape}')
print(f'Epochs: {epochs}')
print(f'Batch size: {batch_size}')

def encode_labels(image, label):
    return image, tf.one_hot(label, depth=num_classes)

train_ds_preprocessed = train_ds_preprocessed.map(encode_labels,
num_parallel_calls=tf.data.AUTOTUNE)
val_ds_preprocessed = val_ds_preprocessed.map(encode_labels,
num_parallel_calls=tf.data.AUTOTUNE)
```

```
Number of Classes: 4
Image shape: (168, 168, 1)
Epochs: 50
Batch size: 32
```

Data Preprocessing & Training Setup Values, this section defines a `data_augmentation` pipeline using Keras preprocessing layers to artificially diversify the training data through random flips, rotations, contrast changes, zoom, and translation. It sets up preprocessing functions to normalize pixel values and apply augmentations, applies these transformations to the training and test datasets using `tf.data.Dataset.map`, displays example augmented images for visual verification, and concludes by one-hot encoding the labels for multi-class classification compatibility, while also defining key training parameters such as number of classes, image shape, epochs, and batch size.

6. Model Training and Analysis

```
# Building model
model = Sequential([
    # Input tensor shape
    Input(shape=image_shape),

    # Convolutional layer 1
    Conv2D(64, (5, 5), activation="relu"),
    MaxPooling2D(pool_size=(3, 3)),

    # Convolutional layer 2
    Conv2D(64, (5, 5), activation="relu"),
    MaxPooling2D(pool_size=(3, 3)),

    # Convolutional layer 3
    Conv2D(128, (4, 4), activation="relu"),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolutional layer 4
    Conv2D(128, (4, 4), activation="relu"),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),

    # Dense layers
    Dense(512, activation="relu"),
    Dense(num_classes, activation="softmax")
])

# Model summary
model.summary()

# COompilng model with Adam optimizer
optimizer = Adam(learning_rate=0.001, beta_1=0.85, beta_2=0.9925)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics= ['accuracy'])
```

```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d (Conv2D)                  │ (None, 164, 164, 64)    │
1,664 │
├──────────────────────────────────┼─────────────────────────┼──────────
```

```
──────┤
│ max_pooling2d (MaxPooling2D)      │ (None, 54, 54, 64)        │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ conv2d_1 (Conv2D)                 │ (None, 50, 50, 64)        │
102,464 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ max_pooling2d_1 (MaxPooling2D)    │ (None, 16, 16, 64)        │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ conv2d_2 (Conv2D)                 │ (None, 13, 13, 128)       │
131,200 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ max_pooling2d_2 (MaxPooling2D)    │ (None, 6, 6, 128)         │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ conv2d_3 (Conv2D)                 │ (None, 3, 3, 128)         │
262,272 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ max_pooling2d_3 (MaxPooling2D)    │ (None, 1, 1, 128)         │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ flatten (Flatten)                 │ (None, 128)               │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ dense (Dense)                     │ (None, 512)               │
66,048 │
├───────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                   │ (None, 4)                 │
2,052 │
└───────────────────────────────────┴───────────────────────────┴──────────
──────┘

 Total params: 565,700 (2.16 MB)

 Trainable params: 565,700 (2.16 MB)

 Non-trainable params: 0 (0.00 B)

# Custom callback for reducing learning rate at accuracy values
class ReduceLROnMultipleAccuracies(tf.keras.callbacks.Callback):
    def __init__(self, thresholds, factor, monitor='val_accuracy',
verbose=1):
        super(ReduceLROnMultipleAccuracies, self).__init__()
        self.thresholds = thresholds  # List of accuracy thresholds
        self.factor = factor  # Factor to reduce the learning rate
        self.monitor = monitor
        self.verbose = verbose
        self.thresholds_reached = [False] * len(thresholds)  # Track
each threshold

    def on_epoch_end(self, epoch, logs=None):
        current_accuracy = logs.get(self.monitor)
        for i, threshold in enumerate(self.thresholds):
            if current_accuracy >= threshold and not
self.thresholds_reached[i]:
                optimizer = self.model.optimizer
                old_lr = optimizer.learning_rate.numpy()
                new_lr = old_lr * self.factor
                optimizer.learning_rate.assign(new_lr)
                self.thresholds_reached[i] = True  # Mark this
threshold as reached
                if self.verbose > 0:
                    print(f"\nEpoch {epoch+1}: {self.monitor} reached
{threshold}. Reducing learning rate from {old_lr} to {new_lr}.")

# Try a custom callback
thresholds = [0.96, 0.99, 0.9935]
lr_callback = ReduceLROnMultipleAccuracies(thresholds=thresholds,
factor=0.75, monitor='val_accuracy', verbose=False)

# Callbacks for improved covergence of gradient and best test accuracy
model_rlr = ReduceLROnPlateau(monitor='val_loss', factor=0.8,
min_lr=1e-4, patience=4, verbose=False)
model_mc = ModelCheckpoint('model.keras', monitor='val_accuracy',
mode='max', save_best_only=True, verbose=False)

# Training the model
history = model.fit(
    train_ds_preprocessed,
    epochs=epochs,
    validation_data=val_ds_preprocessed,
    callbacks=[model_rlr, model_mc],
    verbose=True
)

Epoch 1/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 28s 120ms/step - accuracy: 0.4664 - loss:
1.1309 - val_accuracy: 0.7209 - val_loss: 0.6790 - learning_rate:
0.0010
Epoch 2/50
```

```
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 108ms/step - accuracy: 0.7907 - loss:
0.5633 - val_accuracy: 0.8278 - val_loss: 0.4331 - learning_rate:
0.0010
Epoch 3/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 98ms/step - accuracy: 0.8676 - loss:
0.3577 - val_accuracy: 0.8766 - val_loss: 0.3318 - learning_rate:
0.0010
Epoch 4/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 23s 111ms/step - accuracy: 0.9089 - loss:
0.2653 - val_accuracy: 0.9214 - val_loss: 0.2181 - learning_rate:
0.0010
Epoch 5/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 102ms/step - accuracy: 0.9321 - loss:
0.1988 - val_accuracy: 0.9363 - val_loss: 0.2037 - learning_rate:
0.0010
Epoch 6/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 98ms/step - accuracy: 0.9435 - loss:
0.1642 - val_accuracy: 0.9395 - val_loss: 0.2084 - learning_rate:
0.0010
Epoch 7/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 22s 104ms/step - accuracy: 0.9452 - loss:
0.1494 - val_accuracy: 0.9363 - val_loss: 0.1879 - learning_rate:
0.0010
Epoch 8/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 98ms/step - accuracy: 0.9606 - loss:
0.1059 - val_accuracy: 0.9575 - val_loss: 0.1297 - learning_rate:
0.0010
Epoch 9/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 22s 105ms/step - accuracy: 0.9679 - loss:
0.0979 - val_accuracy: 0.9568 - val_loss: 0.1177 - learning_rate:
0.0010
Epoch 10/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 98ms/step - accuracy: 0.9711 - loss:
0.0830 - val_accuracy: 0.9575 - val_loss: 0.1470 - learning_rate:
0.0010
Epoch 11/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 105ms/step - accuracy: 0.9702 - loss:
0.0852 - val_accuracy: 0.9544 - val_loss: 0.1470 - learning_rate:
0.0010
Epoch 12/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 97ms/step - accuracy: 0.9726 - loss:
0.0785 - val_accuracy: 0.9686 - val_loss: 0.1066 - learning_rate:
0.0010
Epoch 13/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 109ms/step - accuracy: 0.9717 - loss:
0.0865 - val_accuracy: 0.9780 - val_loss: 0.0723 - learning_rate:
0.0010
Epoch 14/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 100ms/step - accuracy: 0.9848 - loss:
```

```
0.0446 - val_accuracy: 0.9701 - val_loss: 0.1005 - learning_rate:
0.0010
Epoch 15/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 22s 109ms/step - accuracy: 0.9862 - loss:
0.0469 - val_accuracy: 0.9693 - val_loss: 0.0945 - learning_rate:
0.0010
Epoch 16/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 97ms/step - accuracy: 0.9795 - loss:
0.0615 - val_accuracy: 0.9756 - val_loss: 0.0669 - learning_rate:
0.0010
Epoch 17/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 107ms/step - accuracy: 0.9761 - loss:
0.0613 - val_accuracy: 0.9623 - val_loss: 0.1169 - learning_rate:
0.0010
Epoch 18/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 100ms/step - accuracy: 0.9852 - loss:
0.0473 - val_accuracy: 0.9654 - val_loss: 0.1157 - learning_rate:
0.0010
Epoch 19/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 103ms/step - accuracy: 0.9867 - loss:
0.0425 - val_accuracy: 0.9819 - val_loss: 0.0520 - learning_rate:
0.0010
Epoch 20/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 100ms/step - accuracy: 0.9921 - loss:
0.0267 - val_accuracy: 0.9709 - val_loss: 0.1008 - learning_rate:
0.0010
Epoch 21/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 23s 128ms/step - accuracy: 0.9857 - loss:
0.0424 - val_accuracy: 0.9780 - val_loss: 0.0800 - learning_rate:
0.0010
Epoch 22/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 99ms/step - accuracy: 0.9857 - loss:
0.0370 - val_accuracy: 0.9772 - val_loss: 0.0811 - learning_rate:
0.0010
Epoch 23/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 21s 101ms/step - accuracy: 0.9910 - loss:
0.0310 - val_accuracy: 0.9733 - val_loss: 0.1191 - learning_rate:
0.0010
Epoch 24/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 99ms/step - accuracy: 0.9937 - loss:
0.0252 - val_accuracy: 0.9788 - val_loss: 0.0750 - learning_rate:
8.0000e-04
Epoch 25/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 106ms/step - accuracy: 0.9918 - loss:
0.0228 - val_accuracy: 0.9756 - val_loss: 0.0910 - learning_rate:
8.0000e-04
Epoch 26/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 101ms/step - accuracy: 0.9931 - loss:
0.0169 - val_accuracy: 0.9882 - val_loss: 0.0461 - learning_rate:
```

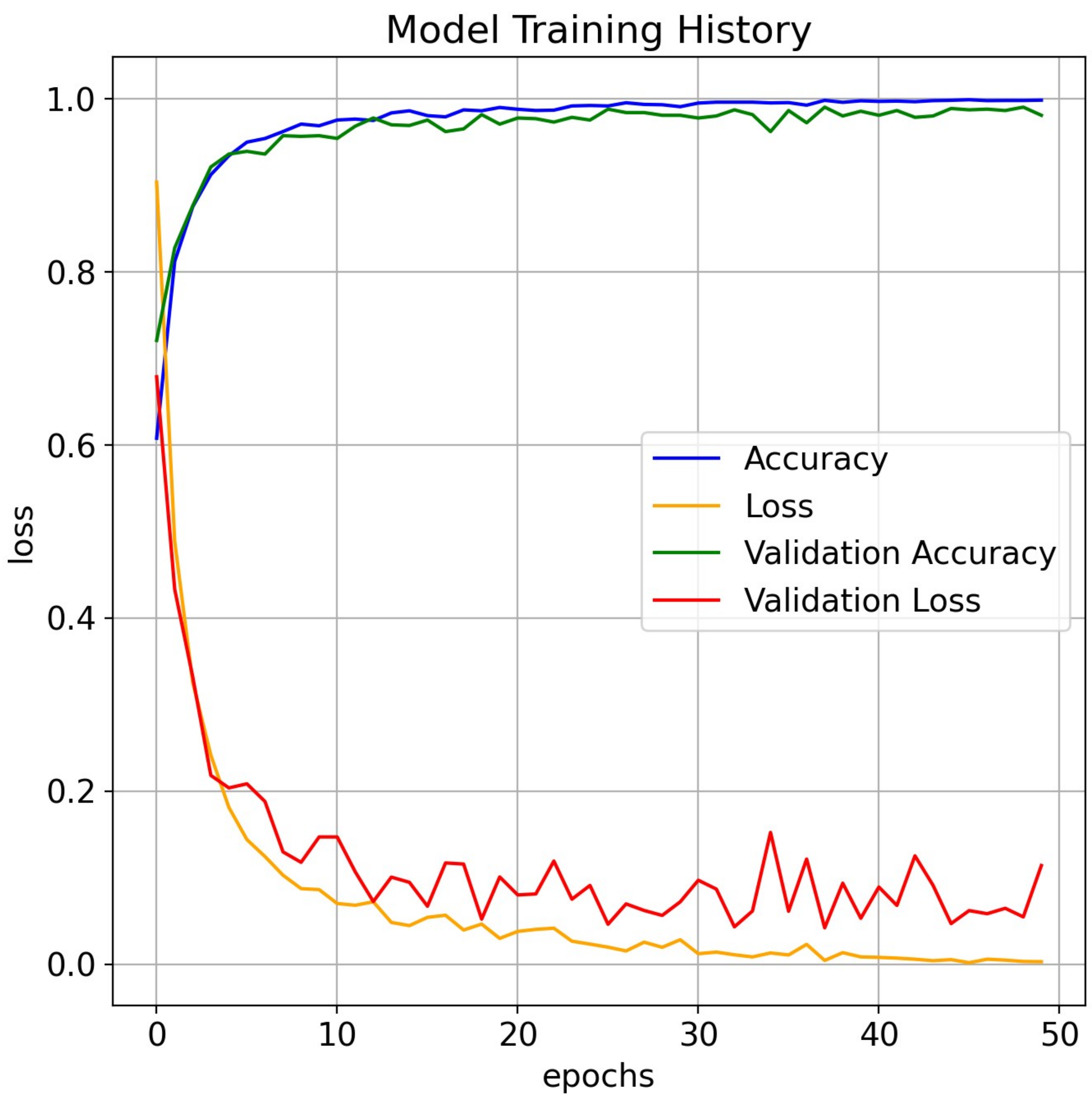
```
8.0000e-04
Epoch 27/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 109ms/step - accuracy: 0.9952 - loss:
0.0122 - val_accuracy: 0.9843 - val_loss: 0.0696 - learning_rate:
8.0000e-04
Epoch 28/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 97ms/step - accuracy: 0.9915 - loss:
0.0312 - val_accuracy: 0.9843 - val_loss: 0.0620 - learning_rate:
8.0000e-04
Epoch 29/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 102ms/step - accuracy: 0.9949 - loss:
0.0156 - val_accuracy: 0.9811 - val_loss: 0.0563 - learning_rate:
8.0000e-04
Epoch 30/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 97ms/step - accuracy: 0.9920 - loss:
0.0260 - val_accuracy: 0.9811 - val_loss: 0.0720 - learning_rate:
8.0000e-04
Epoch 31/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 103ms/step - accuracy: 0.9936 - loss:
0.0152 - val_accuracy: 0.9780 - val_loss: 0.0970 - learning_rate:
6.4000e-04
Epoch 32/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 101ms/step - accuracy: 0.9944 - loss:
0.0202 - val_accuracy: 0.9803 - val_loss: 0.0867 - learning_rate:
6.4000e-04
Epoch 33/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 100ms/step - accuracy: 0.9965 - loss:
0.0103 - val_accuracy: 0.9874 - val_loss: 0.0431 - learning_rate:
6.4000e-04
Epoch 34/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 97ms/step - accuracy: 0.9975 - loss:
0.0065 - val_accuracy: 0.9819 - val_loss: 0.0614 - learning_rate:
6.4000e-04
Epoch 35/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 22s 103ms/step - accuracy: 0.9964 - loss:
0.0103 - val_accuracy: 0.9623 - val_loss: 0.1522 - learning_rate:
6.4000e-04
Epoch 36/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 17s 97ms/step - accuracy: 0.9952 - loss:
0.0131 - val_accuracy: 0.9866 - val_loss: 0.0611 - learning_rate:
6.4000e-04
Epoch 37/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 103ms/step - accuracy: 0.9921 - loss:
0.0297 - val_accuracy: 0.9725 - val_loss: 0.1214 - learning_rate:
6.4000e-04
Epoch 38/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 106ms/step - accuracy: 0.9971 - loss:
0.0053 - val_accuracy: 0.9906 - val_loss: 0.0420 - learning_rate:
5.1200e-04
```

```
Epoch 39/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 103ms/step - accuracy: 0.9949 - loss:
0.0172 - val_accuracy: 0.9803 - val_loss: 0.0935 - learning_rate:
5.1200e-04
Epoch 40/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 97ms/step - accuracy: 0.9983 - loss:
0.0067 - val_accuracy: 0.9858 - val_loss: 0.0530 - learning_rate:
5.1200e-04
Epoch 41/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 22s 102ms/step - accuracy: 0.9968 - loss:
0.0090 - val_accuracy: 0.9811 - val_loss: 0.0891 - learning_rate:
5.1200e-04
Epoch 42/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 98ms/step - accuracy: 0.9986 - loss:
0.0059 - val_accuracy: 0.9866 - val_loss: 0.0678 - learning_rate:
5.1200e-04
Epoch 43/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 22s 105ms/step - accuracy: 0.9962 - loss:
0.0066 - val_accuracy: 0.9788 - val_loss: 0.1251 - learning_rate:
4.0960e-04
Epoch 44/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 97ms/step - accuracy: 0.9986 - loss:
0.0026 - val_accuracy: 0.9803 - val_loss: 0.0911 - learning_rate:
4.0960e-04
Epoch 45/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 19s 103ms/step - accuracy: 0.9984 - loss:
0.0041 - val_accuracy: 0.9890 - val_loss: 0.0468 - learning_rate:
4.0960e-04
Epoch 46/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 98ms/step - accuracy: 0.9995 - loss:
0.0014 - val_accuracy: 0.9874 - val_loss: 0.0619 - learning_rate:
4.0960e-04
Epoch 47/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 21s 103ms/step - accuracy: 0.9989 - loss:
0.0035 - val_accuracy: 0.9882 - val_loss: 0.0583 - learning_rate:
3.2768e-04
Epoch 48/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 100ms/step - accuracy: 0.9982 - loss:
0.0064 - val_accuracy: 0.9866 - val_loss: 0.0645 - learning_rate:
3.2768e-04
Epoch 49/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 20s 108ms/step - accuracy: 0.9980 - loss:
0.0044 - val_accuracy: 0.9906 - val_loss: 0.0546 - learning_rate:
3.2768e-04
Epoch 50/50
179/179 ━━━━━━━━━━━━━━━━━━━━ 18s 98ms/step - accuracy: 0.9990 - loss:
0.0022 - val_accuracy: 0.9811 - val_loss: 0.1140 - learning_rate:
3.2768e-04
```

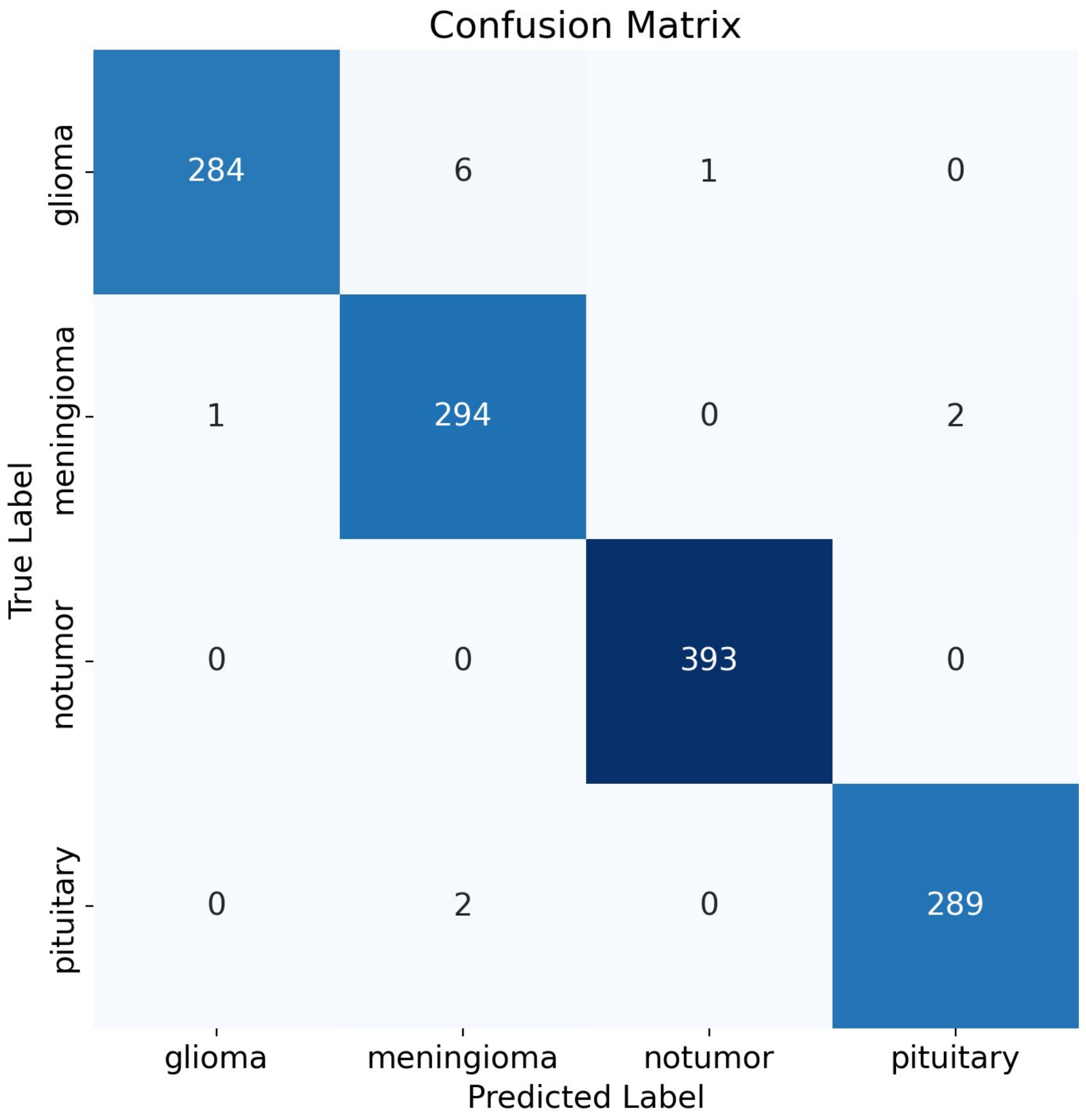
```python
# Loading saved model
model = load_model('model.keras')

# Evaluate model and test data accuracy
val_loss, val_acc = model.evaluate(val_ds_preprocessed)
print(f"Validation accuracy: {val_acc*100:0.4f}%")
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.9908 - loss:
0.0359
Validation accuracy: 99.0566%
```

```python
plt.figure(figsize=(7, 7))

# Plotting training and validation metrics
plt.plot(history.history['accuracy'], color='blue', linestyle='-',
label='Accuracy')
plt.plot(history.history['loss'], color='orange', linestyle='-',
label='Loss')
plt.plot(history.history['val_accuracy'], color='green',
linestyle='-', label='Validation Accuracy')
plt.plot(history.history['val_loss'], color='red', linestyle='-',
label='Validation Loss')

plt.title('Model Training History')
plt.xlabel('epochs')
plt.ylabel('loss')

plt.legend(loc='best')
plt.grid(True)

plt.tight_layout()
plt.show()
```



```
# Using validation data for true and preductions
true_labels = []
predicted_labels = []

# Iterate over dataset to collect predictions and true labels
# Unbatch to get sample-wise prediction
for images, labels in val_ds_preprocessed.unbatch():
    # Store true labels (Convert one-hot to index)
    true_label = np.argmax(labels.numpy())
    true_labels.append(true_label)

    # Get model prediction (Predict expects batch dimension)
    pred = model.predict(tf.expand_dims(images, 0), verbose=False)
    predicted_label = np.argmax(pred)
    predicted_labels.append(predicted_label)

def plot_confusion_matrix(true_labels, predicted_labels,
class_mappings, metrics=False, cmap='Blues'):
    # Compute  confusion matrix
    cm = confusion_matrix(true_labels, predicted_labels)
    plt.figure(figsize=(8, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap=cmap, cbar=False)
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")

    # Mapping of indices to class names in class_mappings
    plt.xticks(ticks=np.arange(num_classes) + 0.5,
labels=class_mappings.keys(), ha='center')
    plt.yticks(ticks=np.arange(num_classes) + 0.5,
labels=class_mappings.keys(), va='center')
    plt.show()

    if metrics:
        # Precision, Recall, and F1-Score for each class & Overall
accuracy
        precision = np.diag(cm) / np.sum(cm, axis=0)
        recall = np.diag(cm) / np.sum(cm, axis=1)
        f1_scores = 2 * precision * recall / (precision + recall)
        accuracy = np.sum(np.diag(cm)) / np.sum(cm)

        print("Class-wise metrics:")
        for i in range(len(class_mappings)):
            class_name = list(class_mappings.keys())[i]
            print(f"\033[94mClass: {class_name}\033[0m")
            print(f"Precision: {precision[i]:.4f}")
            print(f"Recall: {recall[i]:.4f}")
            print(f"F1-Score: {f1_scores[i]:.4f}\n")

        print(f"\033[92mOverall Accuracy: {accuracy:.4f}\033[0m")

# Confusion matrix and netrics from predictions
plot_confusion_matrix(true_labels,
                      predicted_labels,
                      class_mappings,
                      metrics=True)
```



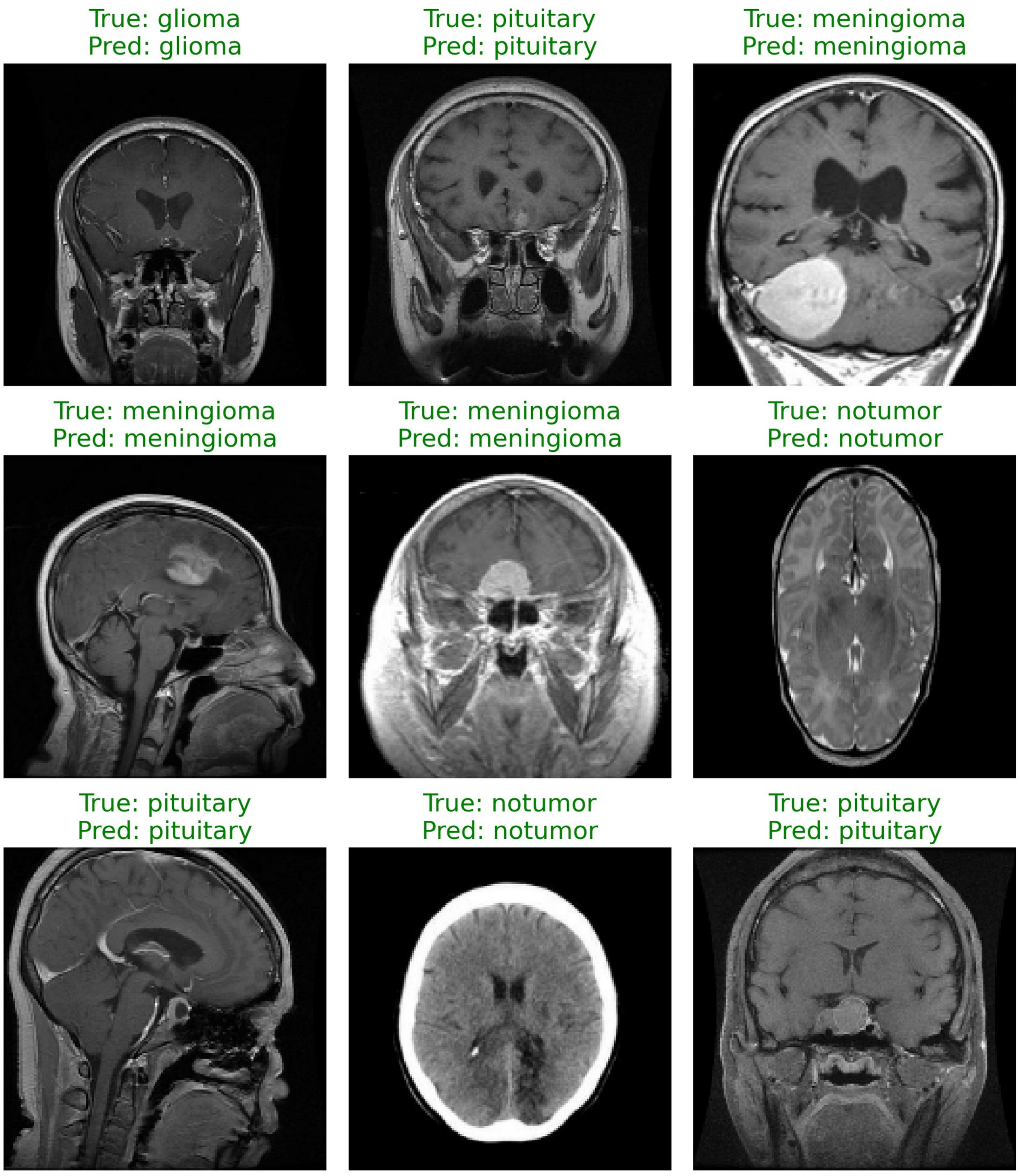
```
Class-wise metrics:
Class: glioma
Precision: 0.9965
Recall: 0.9759
F1-Score: 0.9861

Class: meningioma
Precision: 0.9735
Recall: 0.9899
F1-Score: 0.9816
```

```
Class: notumor
Precision: 0.9975
Recall: 1.0000
F1-Score: 0.9987

Class: pituitary
Precision: 0.9931
Recall: 0.9931
F1-Score: 0.9931

Overall Accuracy: 0.9906
```

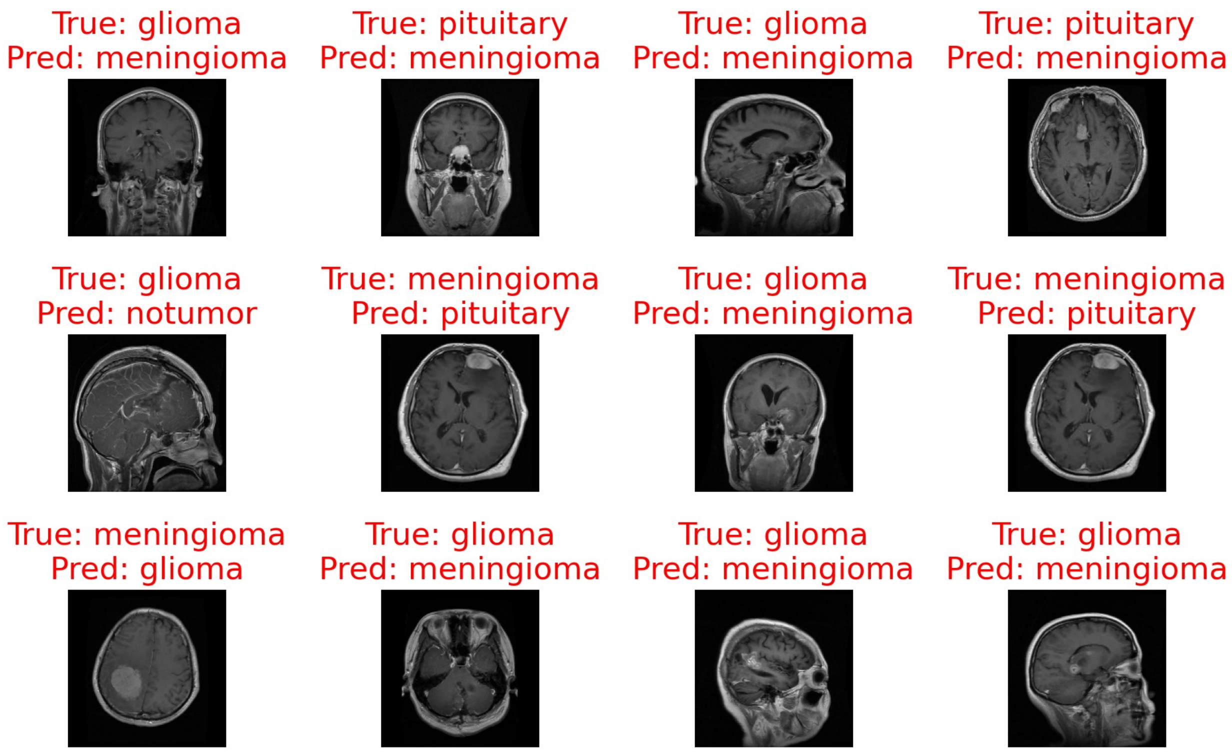
Model Training and Analysis, this section constructs a convolutional neural network (CNN) model with multiple Conv2D and MaxPooling layers followed by dense layers for classification, compiles it using the Adam optimizer and categorical crossentropy loss, and trains it with callbacks for learning rate reduction and model checkpointing to save the best-performing version. After training, the model is evaluated on the test dataset for accuracy, and training metrics are visualized through plots of accuracy and loss curves. Additionally, a confusion matrix is generated to assess per-class precision, recall, and F1-score, providing a detailed breakdown of the model's performance across all categories.

7. Model Testing and Deployment

```python
def plot_sample_predictions(model, dataset, index_to_class,
num_samples=9, figsize=(13, 12)):
    plt.figure(figsize=figsize)
    num_rows = num_cols = int(np.sqrt(num_samples))

    iterator = iter(dataset.unbatch())

    for i in range(1, num_samples + 1):
        image, true_label = next(iterator)
        image_batch = tf.expand_dims(image, 0)
        predictions = model.predict(image_batch, verbose=False)
        predicted_label = np.argmax(predictions, axis=1)[0]

        true_class_index = np.argmax(true_label.numpy())
        true_class = index_to_class[true_class_index]
        predicted_class = index_to_class[predicted_label]

        # Determine title color based on prediction accuracy
        title_color = 'green' if true_class_index == predicted_label
else 'red'

        plt.subplot(num_rows, num_cols, i)
        plt.imshow(image.numpy().squeeze(), cmap='gray')
        plt.title(f"True: {true_class}\nPred: {predicted_class}",
color=title_color)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Plottinng samples with predictions
plot_sample_predictions(model=model,
                        dataset=val_ds_preprocessed,
                        index_to_class=inv_class_mappings,
                        num_samples=9,
                        figsize=(10, 11.5))
```

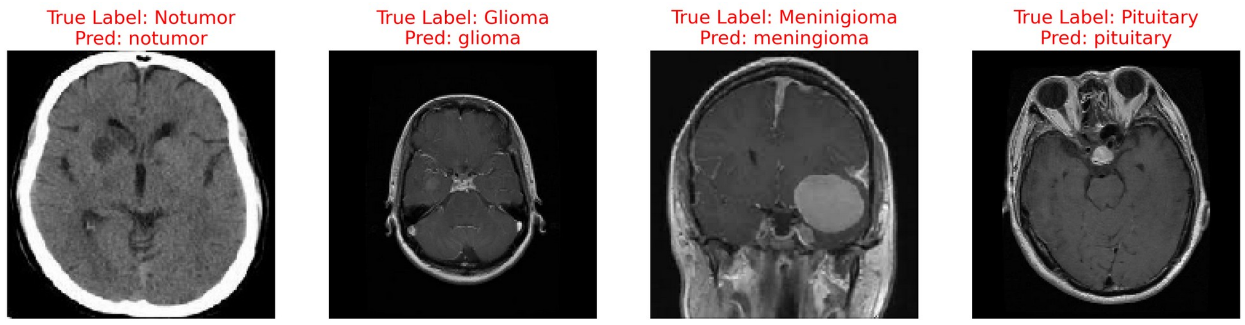


```python
def plot_misclassified_samples(model, dataset, index_to_class, 
figsize=(10, 10)):
    misclassified_images = []
    misclassified_labels = []
    misclassified_predictions = []

    # Iterate over dataset to collect misclassified images
    for image, true_label in dataset.unbatch():
        image_batch = tf.expand_dims(image, 0)
        predictions = model.predict(image_batch, verbose=False)
        predicted_label = np.argmax(predictions, axis=1)[0]
        true_class_index = np.argmax(true_label.numpy())

        if true_class_index != predicted_label:
            misclassified_images.append(image.numpy().squeeze())

misclassified_labels.append(index_to_class[true_class_index])

misclassified_predictions.append(index_to_class[predicted_label])

    # Determine number of rows and columns for subplot
    num_misclassified = len(misclassified_images)
    cols = int(np.sqrt(num_misclassified)) + 1
    rows = num_misclassified // cols + (num_misclassified % cols > 0)

    # Plotting misclassified images
    miss_classified_zip = zip(misclassified_images,
misclassified_labels, misclassified_predictions)
    plt.figure(figsize=figsize)
    for i, (image, true_label, predicted_label) in
enumerate(miss_classified_zip):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(image, cmap='gray')
        plt.title(f"True: {true_label}\nPred: {predicted_label}",
color='red')
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Plotting misclassified images
plot_misclassified_samples(
    model=model,
    dataset=val_ds_preprocessed,
    index_to_class=inv_class_mappings,
    figsize=(10, 6)
)
```



```
# Function to load and preprocess an image
def load_and_preprocess_image(image_path, image_shape=(168, 168)):
    img = image.load_img(image_path, target_size=image_shape,
color_mode='grayscale')
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add the batch
dimension
    return img_array

# Function to display a row of images with predictions
def display_images_and_predictions(image_paths, predictions,
true_labels, figsize=(20, 5)):
    plt.figure(figsize=figsize)
    for i, (image_path, prediction, true_label) in
enumerate(zip(image_paths, predictions, true_labels)):
        ax = plt.subplot(1, len(image_paths), i + 1)
        img_array = load_and_preprocess_image(image_path)
        img_array = np.squeeze(img_array)
        plt.imshow(img_array, cmap='gray')
        title_color = 'green' if prediction == true_label else 'red'
        plt.title(f'True Label: {true_label}\nPred: {prediction}',
color=title_color)
        plt.axis('off')
    plt.show()

# Load and preprocess the images from Testing directory
normal_image_path =
'/content/brain-tumor-mri-dataset/Testing/notumor/Te-no_0057.jpg'
glioma_image_path =
'/content/brain-tumor-mri-dataset/Testing/glioma/Te-gl_0013.jpg'
meningioma_image_path =
'/content/brain-tumor-mri-dataset/Testing/meningioma/Te-me_0086.jpg'
pituitary_tumor_path =
'/content/brain-tumor-mri-dataset/Testing/pituitary/Te-pi_0028.jpg'

# Load and preprocess the images from Testing directory (LOCAL)
# normal_image_path = 'brain-tumor-mri-dataset/Testing/notumor/Te-
no_0057.jpg'
# glioma_image_path = 'brain-tumor-mri-dataset/Testing/glioma/Te-
gl_0013.jpg'
# meningioma_image_path =
'brain-tumor-mri-dataset/Testing/meningioma/Te-me_0086.jpg'
# pituitary_tumor_path =
'brain-tumor-mri-dataset/Testing/pituitary/Te-pi_0028.jpg'


# Image paths
image_paths = [
    normal_image_path,
    glioma_image_path,
    meningioma_image_path,
    pituitary_tumor_path
]

# True labels for images
true_labels = ['Notumor', 'Glioma', 'Meninigioma', 'Pituitary']

# Load and preprocess images, then make predictions
images = [load_and_preprocess_image(path) for path in image_paths]
predictions = [model.predict(image) for image in images]

# Determine the predicted labels
predicted_labels = [inv_class_mappings[np.argmax(one_hot)] for one_hot
in predictions]

# Output the predictions
print(f'Class Mappings: {class_mappings}')
print("\nNormal Image Prediction:", np.round(predictions[0], 3)[0])
print("Glioma Image Prediction:", np.round(predictions[1], 3)[0])
print("Meningioma Image Prediction:", np.round(predictions[2], 3)[0])
print("Pituitary Image Prediction:", np.round(predictions[3], 3)[0])

# Display images with predictions
display_images_and_predictions(image_paths, predicted_labels,
true_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Class Mappings: {'glioma': 0, 'meningioma': 1, 'notumor': 2,
'pituitary': 3}

Normal Image Prediction: [0. 0. 1. 0.]
Glioma Image Prediction: [1. 0. 0. 0.]
Meningioma Image Prediction: [0. 1. 0. 0.]
Pituitary Image Prediction: [0. 0. 0. 1.]
```



Model Testing and Deployment, this section includes functions to visualize model predictions on sample images, highlighting correct predictions in green and incorrect ones in red for quick interpretation. It also defines and displays misclassified images to help identify potential weaknesses or patterns in the model's errors. Additionally, it demonstrates how to load and preprocess individual MRI images from the validation set, perform inference using the trained model, and display both the raw images and their corresponding predictions to validate real-world performance and readiness for deployment scenarios.

8. Running the Project with Streamlit

```
!pip install streamlit -q

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 3.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 90.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 134.0 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 7.4 MB/s eta
0:00:00

%%writefile app.py
import streamlit as st
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

```python
import time
import os

# --- Page Configuration ---
st.set_page_config(page_title="Brain MRI Tumor Classifier",
layout="centered")

# --- Define Class Mappings ---
class_mappings = {'Glioma': 0, 'Meninigioma': 1, 'Notumor': 2,
'Pituitary': 3}
inv_class_mappings = {v: k for k, v in class_mappings.items()}
class_emojis = {
    'Glioma': "□□",
    'Meninigioma': "□□",
    'Notumor': "□□",
    'Pituitary': "□□"
}
class_colors = {
    'Glioma': "#FF4B4B",
    'Meninigioma': "#FFD700",
    'Notumor': "#4CAF50",
    'Pituitary': "#9370DB"
}
image_dim = (168, 168)

# --- Load Model ---
model_path = 'model.keras'

@st.cache_resource
def load_brain_tumor_model(model_path):
    if not os.path.exists(model_path):
        st.error(f"Model file not found at {model_path}. Please ensure
the model file exists.")
        return None
    try:
        model = load_model(model_path)
        return model
    except Exception as e:
        st.error(f"Error loading model: {e}")
        return None

model = load_brain_tumor_model(model_path)
if model is None:
    st.stop()

# --- Image Preprocessing ---
def load_and_preprocess_image(uploaded_file, image_shape=(168, 168)):
    try:
        img = image.load_img(uploaded_file, target_size=image_shape,
color_mode='grayscale')
```

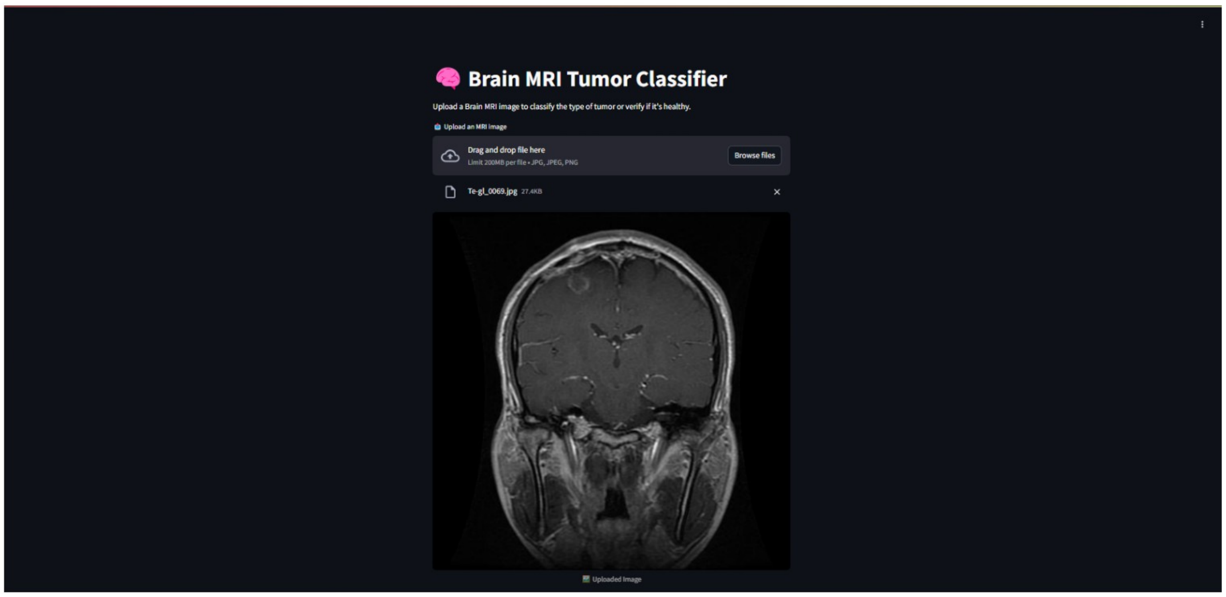
```
        img_array = image.img_to_array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)
        return img_array
    except Exception as e:
        st.error(f"Error processing image: {e}")
        return None

# --- App Title ---
st.title("□ Brain MRI Tumor Classifier")
st.markdown("Upload a Brain MRI image to classify the type of tumor or
verify if it's healthy.")

# --- Upload Image ---
uploaded_file = st.file_uploader("□ Upload an MRI image", type=["jpg",
"jpeg", "png"])

if uploaded_file:
    st.image(uploaded_file, caption=" Uploaded Image",
use_container_width=True)
    img_array = load_and_preprocess_image(uploaded_file,
image_shape=image_dim)

    if img_array is not None:
        st.subheader("□ Classification Progress")
        progress_bar = st.progress(0)
        status_text = st.empty()

        for percent in range(0, 101, 10):
            time.sleep(0.05)
            progress_bar.progress(percent)
            status_text.text(f"Processing: {percent}%")

        predictions = model.predict(img_array)
        predicted_label_index = np.argmax(predictions, axis=1)[0]
        predicted_class = inv_class_mappings[predicted_label_index]
        confidence_scores = predictions[0]

        # --- Prediction Output ---
        st.markdown(f"""
        <div style='text-align: center; font-size: 1.5em; font-weight:
bold; color: {class_colors[predicted_class]};'>
            Prediction: {class_emojis[predicted_class]} <br>
{predicted_class}
        </div>
        """, unsafe_allow_html=True)

        # --- Confidence Scores ---
        st.subheader("□ Confidence Scores")
        for class_name, score in zip(class_mappings.keys(),
confidence_scores):
```

```
            bar_percent = int(score * 100)
            st.markdown(f"""
                <div style='margin-bottom: 8px;'>
                    <b>{class_emojis[class_name]} {class_name}:</b>
{bar_percent:.2f}%
                    <div style='background-color: #eee; border-radius:
4px; height: 15px;'>
                        <div style='width: {bar_percent}%; background-
color: {class_colors[class_name]}; height: 100%; border-radius:
4px;'></div>
                    </div>
                </div>
            """, unsafe_allow_html=True)

# --- Footer and Credits ---
st.markdown("<br><hr style='margin-top: 30px; margin-bottom:
30px;'><br>", unsafe_allow_html=True)

st.markdown(
    """
    <div style='text-align: center; color: #4A4A4A; font-size:
0.9em;'>
        <p>Developed by: <b>Edwin P. Bayog Jr.</b><br>
        <i>BSCpE 3-A</i></p>
        <p style='margin-top: 5px;'>Course: <b>CpETE1 Embedded System
1 - Realtime Systems</b></p>
        <i>Technological University of the Philippines - Visayas</i>
    </div>
    <br>
    """,
    unsafe_allow_html=True
)
```

```
Writing app.py
```

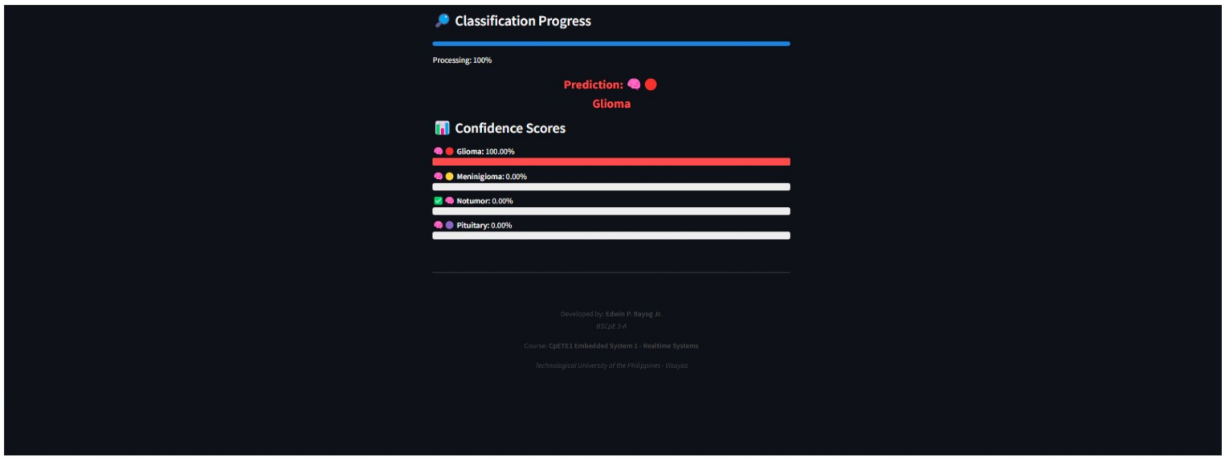
```
# RUN IF YOU ARE IN COLAB
!wget -q -O - ipv4.icanhazip.com
print("Above is your Colab's public IP address, use as Tunnel
Password.")

print("Starting Streamlit app in the background...")
!nohup streamlit run app.py --server.port 8501 --server.headless true
--server.enableCORS false > streamlit.log &

import time
time.sleep(5)

print("Attempting to start localtunnel...")
!npx localtunnel --port 8501
```

```
34.125.178.171
Above is your Colab's public IP address, use as Tunnel Password.
Starting Streamlit app in the background...
nohup: redirecting stderr to stdout
Attempting to start localtunnel...
-tigers-raise.loca.lt
^C

# RUN IF YOU ARE IN LOCAL JUPYTER NOTEBOOK
!streamlit run app.py & npx localtunnel --port 8501

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Image paths
image_path1 = '/content/streamlit_upload1.png'
image_path2 = '/content/streamlit_upload2.png'

# Load images
img1 = mpimg.imread(image_path1)
img2 = mpimg.imread(image_path2)

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(8, 8))

# Show first image
axes[0].imshow(img1)
axes[0].axis('off')

# Show second image
axes[1].imshow(img2)
axes[1].axis('off')

plt.tight_layout()
plt.show()
```



🧠 Brain MRI Tumor Classifier

Upload a Brain MRI image to classify the type of tumor or verify if it's healthy.



🔎 Classification Progress

Processing: 100%

Prediction: 🧠 🔴
Glioma

📊 Confidence Scores

🧠 🔴 Glioma: 100.00%

🧠 🟡 Meninigioma: 0.00%

✅ 🧠 Notumor: 0.00%

🧠 🟣 Pituitary: 0.00%

Developed by: Edwin P. Bayog Jr.
BSCpE 3-A

Course: CpETE1 Embedded System 1 - Realtime Systems

Technological University of the Philippines - Visayas

Running the Project with Streamlit, this section installs and configures **Streamlit** to deploy the trained brain tumor classification model as a user-friendly web application. It creates an `app.py` file that defines the app layout, loads the model, preprocesses uploaded MRI images, and displays predictions with confidence scores using progress bars and color-coded labels for enhanced visualization. The app supports image uploads, shows real-time classification results, and includes styling elements like emojis and custom fonts for better UX. Finally, instructions are provided to run the app in Google Colab or locally using a tunneling service to expose the local server to the web for external access.