National College of Ireland

Configuration Manual

MSc in DATA ANALYTICS

Ajay Kumar Bagam
Student ID:x23325721

School of Computing
National College of Ireland

Supervisor: Abdul Qayum

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ajay Kumar Bagam

Student ID: X23325721

Programme: MSc in Data Analytics **Year:** 2025

Module: Research Practicum

Supervisor: Abdul Qayum

Submission Due Date: 11-12-2025

Project Title: EFT-SA-Net: An Attention-Enhanced EfficientNetB6 Framework for Accurate and Explainable Brain Tumour Classification from MRI Scans

Word Count: **681** **Page Count**: 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ajay Kumar Bagam

Date: 11-12-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date	
Penalty Applied (if applicable):	

Configuration Manual: Brain Tumor Classification Using Deep Learning & Explainable AI (LIME)

1. Introduction

Brain tumor classification using MRI imaging has become a crucial area in medical AI due to the increasing need for accurate, fast, and reliable diagnostic support. Manual inspection of MRI scans can be time-consuming and prone to errors because tumor appearance varies widely in shape, size, and intensity.

This project utilizes the **Brain Tumor Classification MRI dataset from Kaggle** and applies multiple deep learning models—including CNN, InceptionV3, Xception, EfficientNet-B6, and an advanced hybrid model (EFT-SA-Net)—to automatically classify tumor types.

Explainable AI (XAI) using **LIME** is integrated to highlight important image regions influencing predictions, improving clinicians' trust and interpretability.

This project supports:
• Clinical decision support
• Early tumor identification
• AI-assisted radiology
• Cloud-based medical diagnostic platforms

2. Research Question

Primary Research Question:
How effectively can deep learning models classify brain tumors from MRI images, and what advantages do attention-based models (such as EFT-SA-Net) offer compared to conventional CNN architectures?

Key Components Investigated:

(A) Spatial Understanding
Does the model detect tumor-relevant regions such as:
• Tumor boundaries
• Abnormal intensity patterns
• Location-specific structural deviations

(B) Feature Understanding
Can attention-based models prioritize:
• Tumor texture
• Shape irregularities
• Size variations
• MRI contrast behavior

3. Research Objectives

Primary Objective

To develop an efficient and interpretable deep learning–based system capable of accurately classifying brain tumors from MRI images using multiple neural network architectures and Explainable AI (LIME).

Secondary Objectives

1. **To preprocess and standardize MRI images** using resizing, normalization, augmentation, and labeling techniques to ensure high-quality input for deep learning models.
2. **To train and evaluate multiple deep learning architectures** including CNN, Xception, InceptionV3, EfficientNet-B6, and EFT-SA-Net to compare their classification accuracy and robustness.
3. **To integrate Explainable AI (LIME)** for visualizing and interpreting which MRI regions influence classification decisions, enhancing trust and transparency in medical AI.
4. **To measure model performance** using metrics such as accuracy, precision, recall, F1-score, sensitivity, specificity, and confusion matrix.
5. **To develop a user-friendly GUI (Flask-based web application)** that enables clinicians to upload MRI images and receive tumor predictions with explanatory heatmaps.
6. **To analyze the impact of advanced attention-based models** (such as EFT-SA-Net) on feature extraction quality, tumor localization, and classification reliability.

3. Environment Setup

Correct environment setup ensures reproducibility and proper model training.

3.1 Recommended Hardware

Component	Recommended
GPU	NVIDIA RTX 3060 / T4 / A100
RAM	16 GB minimum
CPU	Quad-core
Storage	10–15 GB

3.2 Software Requirements

Supported OS:
• Windows 10/11
• macOS
• Ubuntu Linux

3.3 Python Version

Python 3.9–3.11 recommended.

3.4 Required Libraries

Core Libraries:
• numpy, pandas, matplotlib, seaborn

Deep Learning:
• tensorflow, keras
• EfficientNet / pretrained models
• scikit-learn

Explainable AI:
• lime

Imaging:
• opencv-python

3.5 Installation Commands

```
pip install numpy pandas matplotlib seaborn opencv-python
pip install tensorflow keras scikit-learn
pip install efficientnet
```

```
pip install lime
```

For GPU acceleration:

```
pip install tensorflow-gpu
```

4. Dataset Description

Dataset: **Brain Tumor MRI Classification (Kaggle)**
Classes include:
• Glioma
• Meningioma
• Pituitary
• No Tumor

Data characteristics:
• 7,000+ MRI images
• Multiple patient scans
• Varying contrast levels and orientations

https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

5. Data Preprocessing

Preprocessing steps include:
• Image loading via OpenCV

```
image_list, label_list = [], []  # initialize empty lists to store images and labels

try:  # start of try block
    print("[INFO] Loading images ...")  # print info message

    root_dir = [d for d in listdir(directory_root) if d != ".DS_Store"]  # list valid folders

    image_classes_folder_list = listdir("/kaggle/input/brain-tumor-classification-mri/Training")  # list class folders
    image_classes_folder_list = [d for d in image_classes_folder_list if d != ".DS_Store"]  # remove unwanted files

    for dataset in listdir('/kaggle/input/brain-tumor-classification-mri'):  # loop through dataset folders
        for tumor_class in image_classes_folder_list:  # loop through tumor class folders

            print(f"[INFO] Processing {tumor_class} ...")  # print current class

            class_path = f"{directory_root}/{dataset}/{tumor_class}/"  # generate class path
            image_files = [img for img in listdir(class_path) if img != ".DS_Store"]  # list image files

            for img_name in image_files:  # loop through each image
                img_path = f"{class_path}{img_name}"  # construct full image path

                if img_path.lower().endswith((".jpg", ".jpeg", ".png")):  # check valid image format
                    image_list.append(image_array(img_path))  # append image array
                    label_list.append(tumor_class)  # append label

    print("[INFO] Image loading completed")  # print completion message
```

• Label encoding using LabelBinarizer

```
print("[INFO] Encoding labels with LabelBinarizer...")  # printing info before encoding
class_labels = LabelBinarizer()  # initializing label binarizer
Y = class_labels.fit_transform(Y)  # encoding labels into one-hot format

pickle.dump(class_labels, open('/kaggle/working/label_transform.pkl', 'wb'))  # saving label binarizer for later use

cls = len(class_labels.classes_)  # counting total unique classes
print(f"[INFO] Label encoding complete. Total classes: {cls}")  # printing class count summary
```

• Image resizing (e.g., 128×128)

```
image_size = 128  # image size for resizing
default_image_size = tuple((128, 128))  # default target size as a tuple
labels = os.listdir('/kaggle/input/brain-tumor-classification-mri/Training')  # list of class folders
directory_root = '/kaggle/input/brain-tumor-classification-mri'  # root path of dataset
classes = len(labels)  # total number of classes
BATCHZ_SIZE = 32  # batch size for training
```

• Pixel normalization

```
def image_array(dir):  # function to read and convert image to array
    try:
        img = cv2.imread(dir)  # read image
        if img is not None :  # check if successfully read
            img = cv2.resize(img, default_image_size)  # resize to default size
            return img_to_array(img)  # convert to array
        else :  # if image is empty
            return np.array([])  # return empty array
    except Exception as e:  # catch errors
        print(f"Error : {e}")  # print error
        return None  # return None on exception
```

• Train/Validation/Test split (80:10:10)

```
# Splitting into Train (90%) and Test (10%) sets while maintaining class balance
x_train, x_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.1, stratify=Y, random_state=1)

# Further splitting Train → Train (90%) and Validation (10%)
x_train, x_val, y_train, y_val = train_test_split(
    x_train, y_train, test_size=0.1, stratify=y_train, random_state=1)
```

6. Exploratory Data Analysis

These help in understanding dataset balance and visual tumor patterns.

```
import matplotlib.image as mpimg  # import module for reading images
plt.figure(figsize=(12, 8))  # set plot size

BASE_URL = '/kaggle/input/brain-tumor-classification-mri/Training/'  # base dataset path
image_count = 1  # counter for subplot

for folder in os.listdir(BASE_URL):  # iterate folders
    if folder.startswith('.'):  # skip hidden files
        continue

    folder_path = os.path.join(BASE_URL, folder)  # full folder path

    for idx, filename in enumerate(os.listdir(folder_path)):  # iterate images
        if idx == 1:  # show only first image per class
            break

        plt.subplot(2, 2, image_count)  # create 2x2 grid subplot
        image_count += 1  # increment subplot index

        img = mpimg.imread(os.path.join(folder_path, filename))  # read image
        plt.imshow(img)  # show image
        plt.title(folder.upper(), fontsize=12, fontweight='bold')  # show class name
        plt.axis('off')  # hide axes

        for spine in plt.gca().spines.values():  # white border around image
            spine.set_edgecolor('white')  # set white color
            spine.set_linewidth(3)  # set thickness
```

7. Model Architectures

Models implemented:

1. **Custom CNN** – baseline model

```
#CNN model
depth=3  # number of channels (RGB = 3)
model = Sequential()  # initialize sequential CNN model
inputShape = (image_size,image_size, depth)  # input image shape
chanDim = -1  # channel dimension
if K.image_data_format() == "channels_first":  # check if channels are first
    inputShape = (depth,image_size,image_size)  # adjust shape if channels first
    chanDim = 1  # update channel dimension
model.add(Conv2D(16, (3, 3), padding="same",input_shape=inputShape))  # first conv layer
model.add(Activation("relu"))  # apply ReLU activation
model.add(BatchNormalization(axis=chanDim))  # normalize activations
model.add(MaxPooling2D(pool_size=(3, 3)))  # downsample feature maps
model.add(Dropout(0.25))  # regularization to reduce overfitting
model.add(Conv2D(16, (3, 3), padding="same"))  # second conv layer
model.add(Activation("relu"))  # apply ReLU activation
model.add(BatchNormalization(axis=chanDim))  # normalize activations
model.add(Conv2D(16, (3, 3), padding="same"))  # third conv layer
model.add(Activation("relu"))  # apply ReLU activation
model.add(BatchNormalization(axis=chanDim))  # normalize activations
model.add(MaxPooling2D(pool_size=(2, 2)))  # further downsampling
model.add(Dropout(0.25))  # dropout layer
model.add(Conv2D(32, (3, 3), padding="same"))  # fourth conv layer
model.add(Activation("relu"))  # apply ReLU activation
model.add(BatchNormalization(axis=chanDim))  # normalize activations
model.add(Conv2D(32, (3, 3), padding="same"))  # fifth conv layer
model.add(Activation("relu"))  # apply ReLU activation
model.add(BatchNormalization(axis=chanDim))  # normalize activations
```

2. **Xception** – deep feature extractor

```
base_model = Xception(weights='imagenet', include_top=False, input_shape=(128,128,3))  # load pretrained Xception model without top layers
for layer in base_model.layers:  # loop through all layers
    layer.trainable = True  # enable fine-tuning of all layers

x = Flatten()(base_model.output)  # flatten the feature maps
x = Dense(64, activation = 'relu')(x)  # add dense layer with 64 neurons and ReLU activation
x = Dropout(0.5)(x)  # apply dropout for regularization
x = Dense(28, activation = 'relu')(x)  # additional dense layer with 28 neurons
x = Dropout(0.5)(x)  # apply dropout again to prevent overfitting
x = Dense(cls, activation = 'softmax')(x)  # output layer for multi-class classification

model = Model(inputs = base_model.input, outputs = x)  # define the final model
model.summary()  # print the model architecture
```

3. **InceptionV3** – multi-scale feature fusion

```
from tensorflow.keras.applications.inception_v3 import InceptionV3  # import InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(128,128,3))  # load pre-trained InceptionV3 base
for layer in base_model.layers:  # loop through all layers
    layer.trainable = True  # allow all layers to be trainable

x = Flatten()(base_model.output)  # flatten the features
x = Dense(64, activation = 'relu')(x)  # fully connected layer with 64 neurons
x = Dropout(0.5)(x)  # apply dropout to reduce overfitting
x = Dense(28, activation = 'relu')(x)  # another dense layer with 28 neurons
x = Dropout(0.5)(x)  # apply dropout again
x = Dense(cls, activation = 'softmax')(x)  # output layer with softmax for classification

model = Model(inputs = base_model.input, outputs = x)  # create final model
model.summary()  # print model architecture
```

4. **EfficientNet-B6** – high-performance transfer learning

```
base_model = EfficientNetB6(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))  # load EfficientNetB6 without top layer
model = base_model.output  # get base model output tensor
model = GlobalAveragePooling2D()(model)  # apply global average pooling to reduce spatial dimensions
model = Dropout(0.3)(model)  # apply dropout to prevent overfitting
model = Dense(128, activation='relu')(model)  # fully connected dense layer with ReLU activation
model = Dropout(0.5)(model)  # additional dropout for regularization
model = Dense(cls, activation='softmax')(model)  # output layer with softmax for multi-class classification
model = Model(inputs = base_model.input, outputs=model)  # define the complete model
model.summary()  # print model architecture
```

5. **EFT-SA-Net** – EfficientNetB6 + Spectral + Temporal + Self-Attention

```
import tensorflow as tf  # import tensorflow
from tensorflow.keras.models import Model  # import Model class
from tensorflow.keras.layers import (  # import required layers
    Input, Dense, GlobalAveragePooling2D, Conv2D, Add, Multiply,
    Reshape, Layer, Activation, Lambda, Concatenate)
from tensorflow.keras.applications import EfficientNetB6  # import EfficientNetB6
import tensorflow.keras.backend as K  # import Keras backend
from tensorflow.keras.optimizers import Adam  # import Adam optimizer

# -------------------------
#  Spectral (Channel) Attention
# -------------------------
def spectral_attention(x, reduction_ratio=8):  # define spectral attention
    channels = int(x.shape[-1])  # get number of channels

    avg_pool = tf.keras.layers.GlobalAveragePooling2D()(x)  # apply global avg pooling
    avg_pool = tf.keras.layers.Reshape((1, 1, channels))(avg_pool)  # reshape avg pool

    max_pool = tf.keras.layers.GlobalMaxPooling2D()(x)  # apply global max pooling
    max_pool = tf.keras.layers.Reshape((1, 1, channels))(max_pool)  # reshape max pool

    shared_dense_one = Dense(channels // reduction_ratio, activation='relu', kernel_initializer='he_normal')  # first dense
    shared_dense_two = Dense(channels, activation='sigmoid', kernel_initializer='he_normal')  # second dense

    avg_out = shared_dense_two(shared_dense_one(avg_pool))  # forward avg
    max_out = shared_dense_two(shared_dense_one(max_pool))  # forward max
```

The advanced model enhances tumor-feature detection via multiple attention layers.

8. Training Pipeline

Training steps:
• Initialize models with chosen hyperparameters

```
Total params: 43,602,454 (166.33 MB)

Trainable params: 43,378,015 (165.47 MB)

Non-trainable params: 224,439 (876.72 KB)
```

• Batch size, epochs, learning rate configurable

```
Epoch 12/50
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 463ms/step - accuracy: 0.9961 - loss: 0.0185
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
48/48 ━━━━━━━━━━━━━━━━━━━━ 23s 477ms/step - accuracy: 0.9961 - loss: 0.0187 - val_accuracy: 0.9379 - val_loss: 0.2580 - learning_rate: 0.0010
...
Epoch 22/50
48/48 ━━━━━━━━━━━━━━━━━━━━ 23s 476ms/step - accuracy: 0.9997 - loss: 8.2317e-04 - val_accuracy: 0.9675 - val_loss: 0.1742 - learning_rate: 2.5000e-04
Epoch 22: early stopping
Restoring model weights from the end of the best epoch: 17.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

• Callbacks: ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

```python
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

# Learning rate scheduler
lr_scheduler = ReduceLROnPlateau(
    monitor='val_accuracy',        # Monitor validation MAE
    factor=0.5,                # Reduce LR by this factor
    patience=3,                # Wait 3 epochs before reducing LR
    min_lr=1e-6,               # Do not go below this LR
    verbose=1
)
# Early stopping to prevent overfitting by monitoring validation accuracy
early_stopping = EarlyStopping(
    monitor='val_accuracy', # Monitor validation accuracy
    patience=5, # Stop training if no improvement for 5 epochs
    restore_best_weights=True, # Restore model weights from the best epoch
    verbose=1  # Print messages when stopping
)
```

• Validation monitoring during training
• Save best performing weights

9. Evaluation Metrics

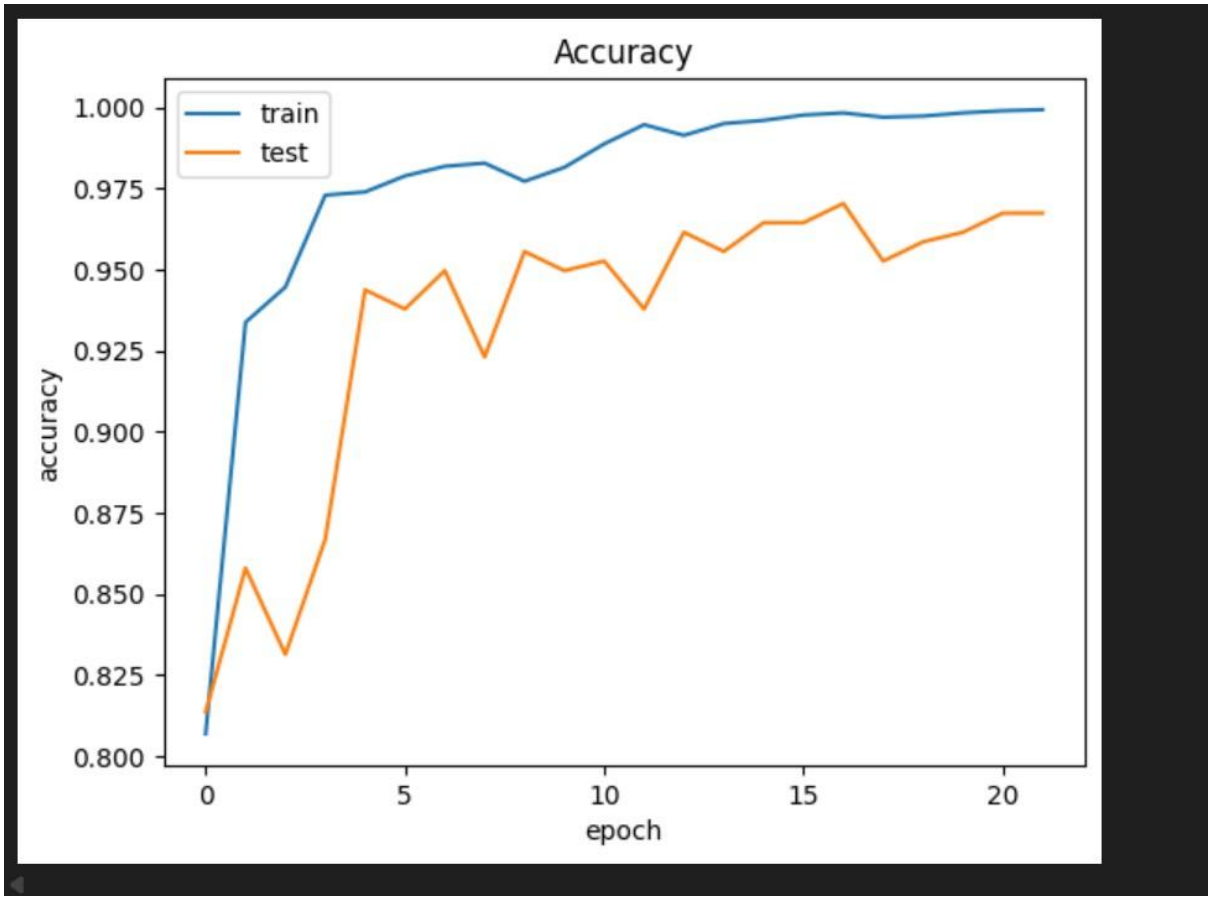
Metrics computed:
• Accuracy
• Precision, Recall, F1-Score
• Confusion Matrix

Model	Accuracy	Macro Precision	Macro Sensitivity (Recall)	Macro Specificity	Macro F1
CNN	0.84	0.84	0.84	0.82	0.83
Exception Net	0.87	0.89	0.87	0.87	0.87
Inception	0.95	0.96	0.95	0.95	0.95
EfficientNet-B6	0.97	0.97	0.97	0.97	0.97
EFT-SA-Net → EfficientNetB6 + Spectral + Temporal + Self-Attention	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**

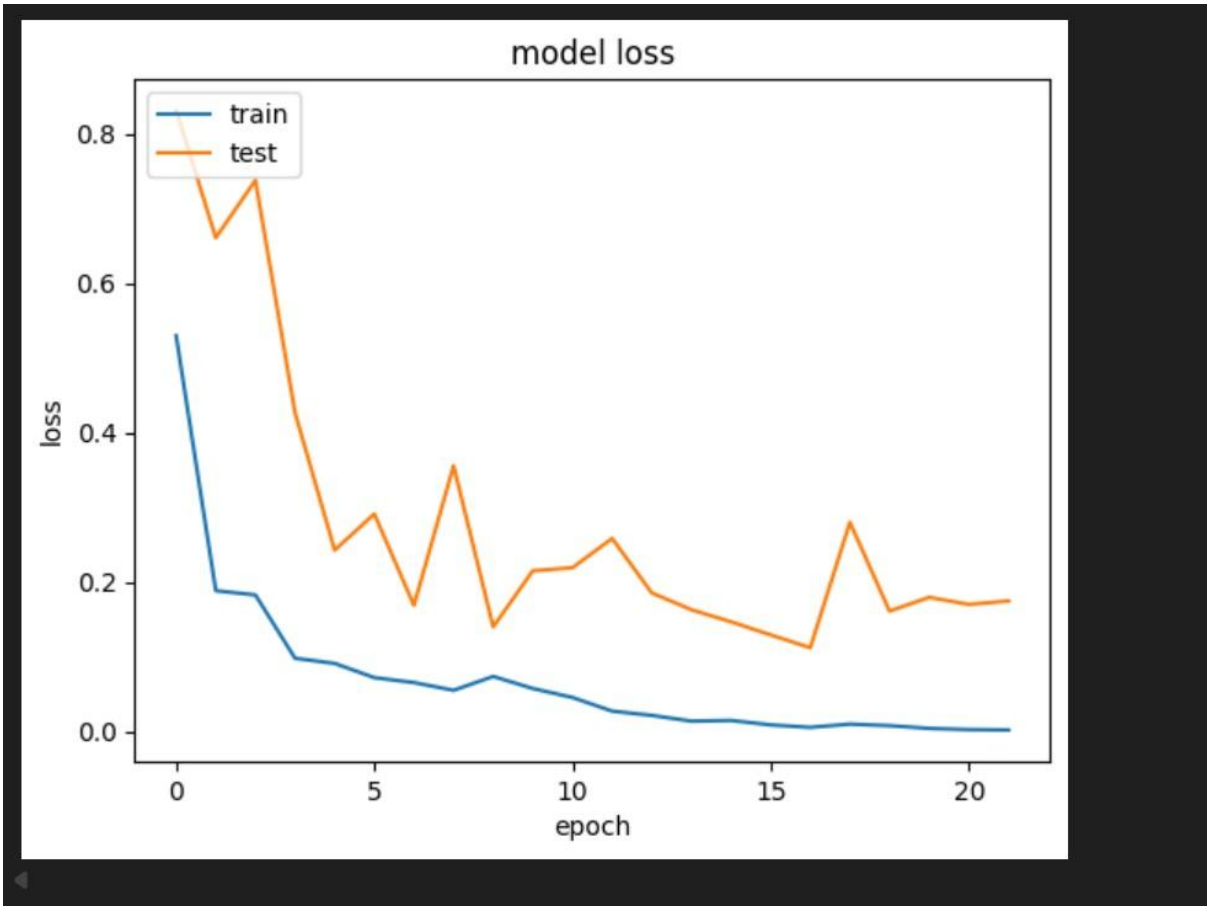
• Sensitivity & Specificity

	class	sensitivity	specificity
0	0	0.989362	0.956989
1	1	0.985765	0.957447
2	2	1.000000	1.000000
3	3	0.992883	0.989362

Visualization:
• Training vs Validation Accuracy



• Training vs Validation Loss



10. Model Testing

The best saved model is tested on unseen MRI images to measure real-world performance. Predictions include:

• Tumor class

```
import tensorflow as tf
from tensorflow.keras.models import load_model
image = '/kaggle/input/brain-tumor-classification-mri/Testing/no_tumor/image(101).jpg'

try:
    img = cv2.imread(image)   # ← Read the input image
    if img is not None :
        img = cv2.resize(img, default_image_size)  # ← Resize to model input size (e.g., 224x224)
        img = img_to_array(img)                    # ← Convert to NumPy array
        img = np.expand_dims(img, axis=0)          # ← Expand to shape (1, H, W, 3) for prediction
    else:
        print('error')  # ← Image not found or unreadable
except Exception as e:
    print(f"Error : {e}")  # ← Handle unexpected errors

prediction = model.predict(img)  # ← Model predicts probabilities
ans = np.argmax(prediction)      # ← Pick class with highest probability
classes = class_labels.classes_  # ← Class labels (must be already defined)
print(classes[ans])              # ← Final predicted class
```

11. Explainable AI (LIME)

LIME is integrated to explain individual model predictions by highlighting influential image regions.
Benefits:
• Identifies tumor-focused areas
• Improves interpretability for clinicians
• Provides transparency for medical AI applications

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from lime import lime_image
from skimage.segmentation import mark_boundaries

# ----------------------------
# Preprocess image (CV2 + resize + array)
# ----------------------------
def preprocess_image(img_path, target_size=(128,128)):
    img = cv2.imread(img_path)                    # read image using cv2 (BGR format)
    img = cv2.resize(img, default_image_size)     # resize to model input size
    return img_to_array(img)                      # convert to numpy array

# Load test image
image_path = "/kaggle/input/brain-tumor-classification-mri/Testing/glioma_tumor/image(16).jpg"
X_test = preprocess_image(image_path)

# Add batch dimension ⇒ shape becomes (1, 128, 128, 3)
X_test = np.expand_dims(X_test, axis=0)

# ----------------------------
# Model Prediction
# ----------------------------
pred = model.predict(X_test)                                  # get prediction probabilities
```