National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Ashay Nishan
Student ID: X23268611

School of Computing
National College of Ireland

Supervisor: Dr. Abid Yaqoob

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ashay Nishan

Student ID: X23268611

Programme: MSc Data Analytics **Year:** 2024-25

Module: MSc Research Practicum

Supervisor: Abid Yaqoob

Submission Due Date: 15st September 2025

Project Title: Configuration Manual

Word Count: **Page Count**: 25

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: Ashay Nishan

Date: 15st September 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

Ashay Nishan
Student ID: x23268611

1 Introduction

This report will provide us with a step-by-step guide on how we will set up and implement the project.

2 Environment Setup

In this section, information is provided on the environment preparation and the system requirement involved in actualizing the project; here the code has been run on Kaggle note-book.

2.1 System Configuration

Table 1: Software Configuration

Requirement	**Specification**
Programming Language	Python Version 3
Tools	Kaggle Notebook, Jupyter
Data Storage	Kaggle's local file system and datasets
Operating System	Debian GNU/Linux 12 (bookworm)

Table 2: Hardware Configuration

Requirement	**Specification**
Processor	Intel Xeon CPU @ 2.20GHz (4 cores, 8 threads)
RAM	31 GB
GPU	NVIDIA Tesla P100 PCIe 16GB VRAM

2.2 Dataset

The available data set in the project is from kaggle and has 4 class glioma_tumor, menin-gioma_tumor, no_tumor, pituitary tumor. It is created in the format of supervised learning

regime, labeled images will be utilized to train and evaluate the deep learning models in order to conduct automated brain tumor classification. The dataset facilitates improved diagnosis as it narrows down the process of interpretation of MRI that has been traditionally interpreted manually.

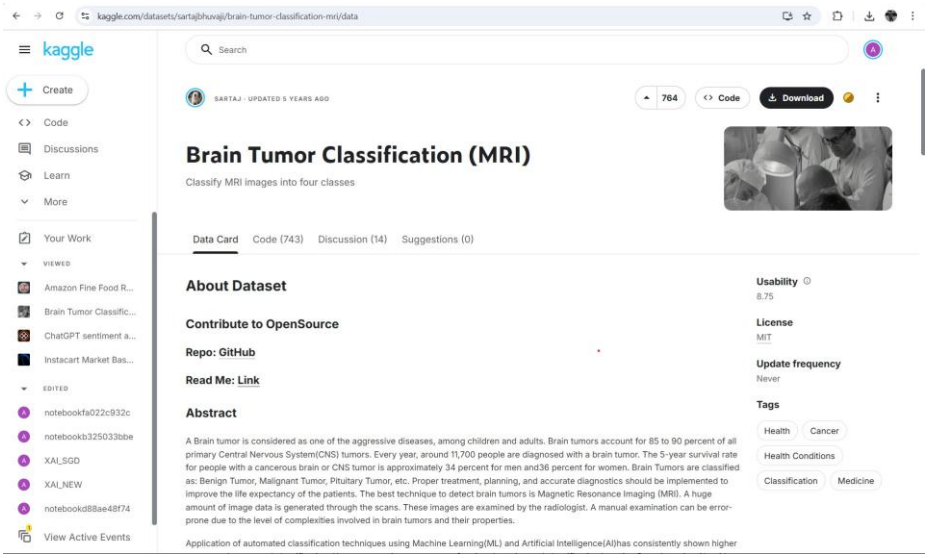


Figure 1: Data Source

2.3 Kaggle Notebook Setup

The dataset and code are executed within the Kaggle Notebook environment, which provides seamless access to datasets directly within the platform. The MRI images are uploaded and organized into training and testing directories, as shown in Figure 2.

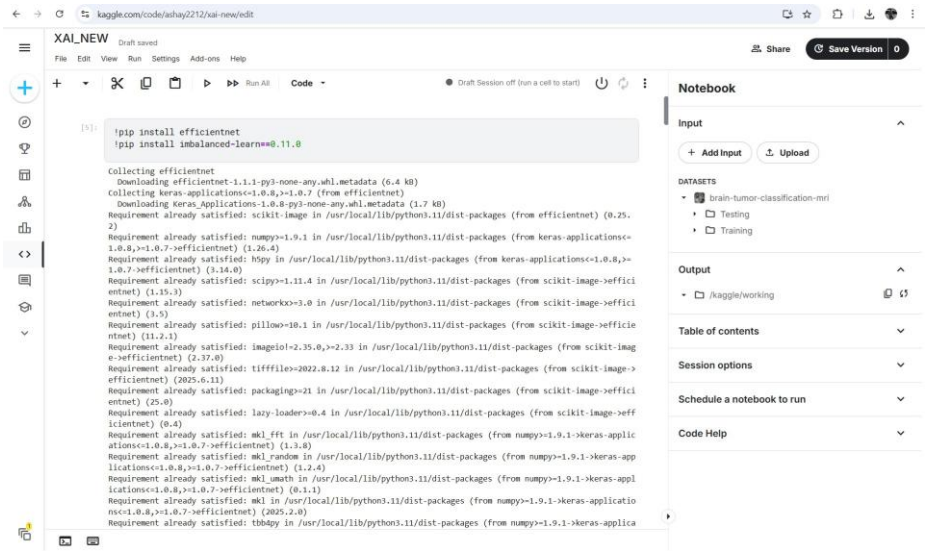


Figure 2: Kaggle Notebook

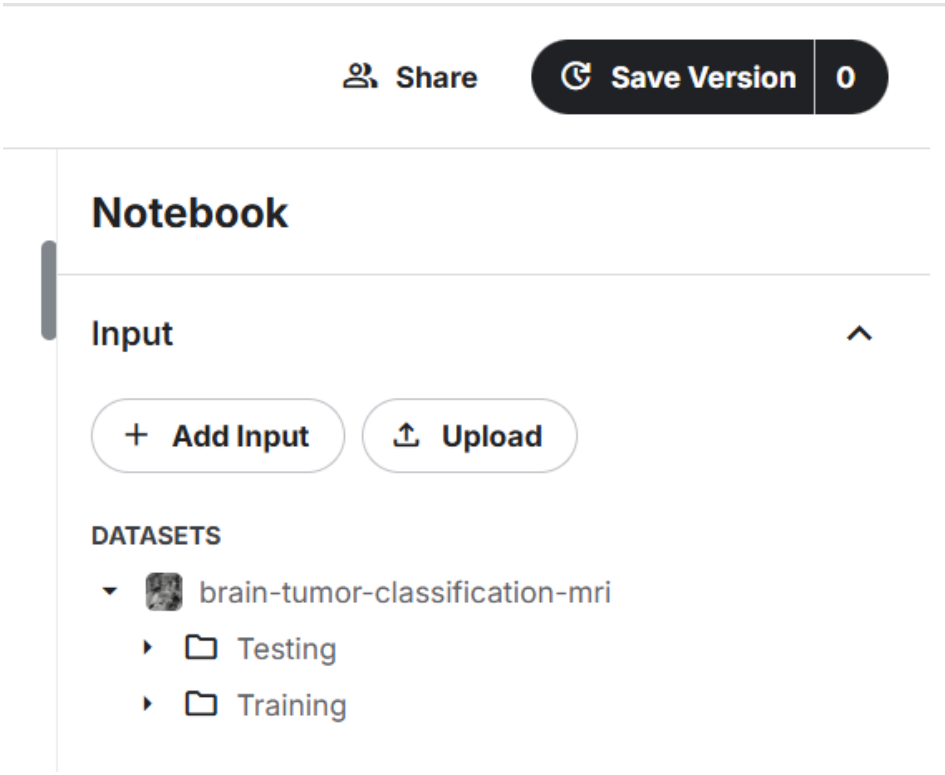


Figure 3: Dataset in Kaggle dataset interface

Kaggle notebooks allow you to check the box to utilize hardware accelerators, e.g., GPUs, to accelerate deep learning mode training. The settings of the run time can be seen in figure 3 & 4, where users are able to select a GPU such as the NVIDIA Tesla P100 to provide further speed in computations.

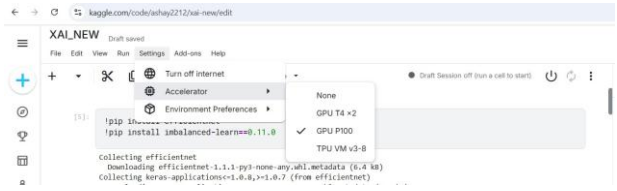


Figure 4: Kaggle inbuilt GPU accelerator

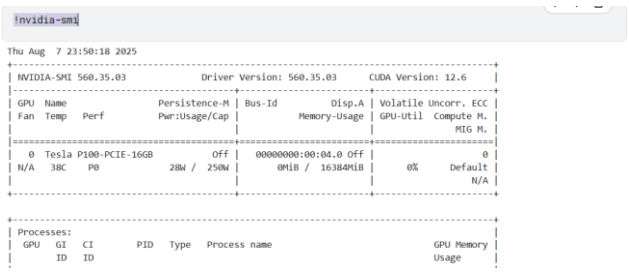
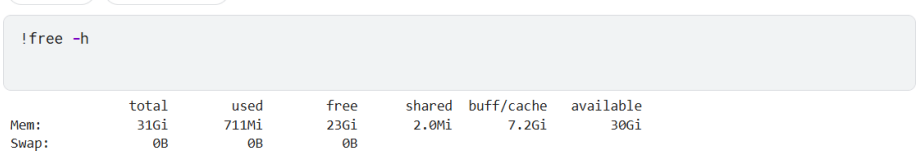


Figure 5: Kaggle GPU interface

The system setup is comprised of 31 GB of RAM and 16 GB of VRAM in the GPU in figure 6 that shall have enough resources to train any complex model. It is also pre-prepared with Python 3 and popular machine learning libraries which makes set-up a trivial thing and development occurs very quickly.



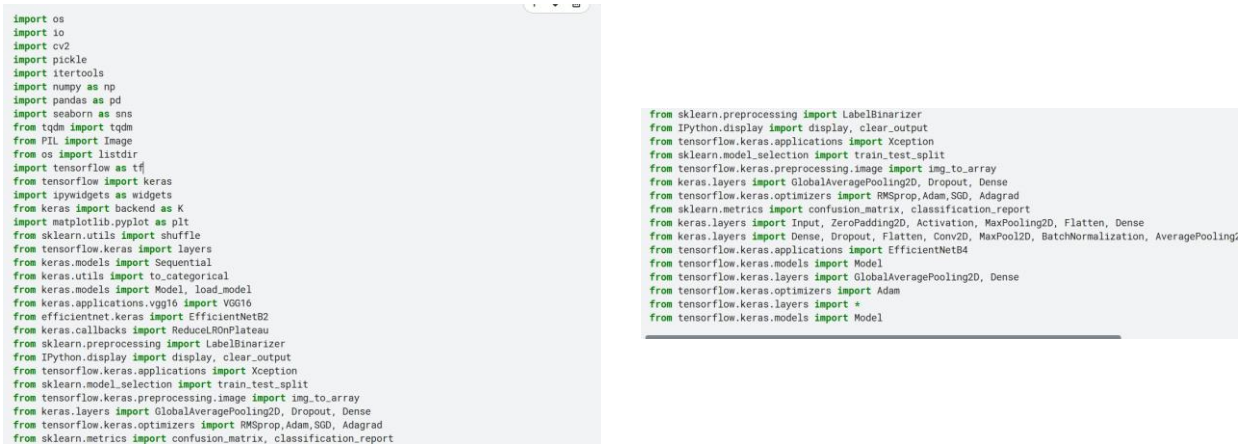
```
!free -h
```

```
              total        used        free      shared  buff/cache   available
Mem:           31Gi       711Mi        23Gi       2.0Mi       7.2Gi        30Gi
Swap:            0B          0B          0B
```

Figure 6: Enter Caption

2.4 Importing Libraries

Here is what we call libraries in your notebook with an emphasis on the meat: TensorFlow/Keras: Constructing CNN architectures (e.g. VGG16, EfficientNet), training loops and transfer learning.



```
import os
import io
import cv2
import pickle
import itertools
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
from PIL import Image
from os import listdir
import tensorflow as tf
from tensorflow import keras
import ipywidgets as widgets
from keras import backend as K
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from tensorflow.keras import layers
from keras.models import Sequential
from keras.utils import to_categorical
from keras.models import Model, load_model
from keras.applications.vgg16 import VGG16
from efficientnet.keras import EfficientNetB2
from keras.callbacks import ReduceLROnPlateau
from sklearn.preprocessing import LabelBinarizer
from IPython.display import display, clear_output
from tensorflow.keras.applications import Xception
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array
from keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.optimizers import RMSprop,Adam,SGD, Adagrad
from sklearn.metrics import confusion_matrix, classification_report
```

```
from sklearn.preprocessing import LabelBinarizer
from IPython.display import display, clear_output
from tensorflow.keras.applications import Xception
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array
from keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.optimizers import RMSprop,Adam,SGD, Adagrad
from sklearn.metrics import confusion_matrix, classification_report
from keras.layers import Input, ZeroPadding2D, Activation, MaxPooling2D, Flatten, Dense
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization, AveragePooling2
from tensorflow.keras.applications import EfficientNetB4
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
```

Figure 7: Imported libraries and modules used in the project for data processing, model building, training, and evaluation.

3 Data Preprocessing

In Figure 8 code reads Kaggle's dataset folders, as it outputs training & testing directories together with subfolders labeled for each tumor type. It checks the dataset's directory.

```
[3]:
image_size=128
default_image_size = tuple((128, 128))
labels = os.listdir('/kaggle/input/brain-tumor-classification-mri/Training')
directory_root = '/kaggle/input/brain-tumor-classification-mri'
classes = len(labels)
BATCHZ_SIZE=32
```

Figure 8: Kaggle dataset folder structure with tumor classes.

The number of records against each target class is depicted in Figure 9

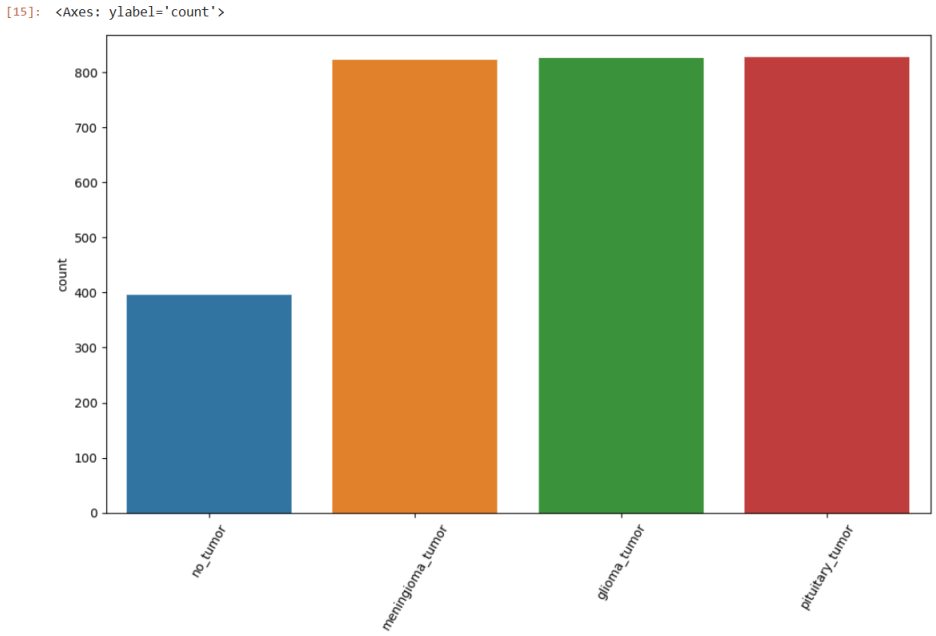


Figure 9: Count Plot

The profile is imbalanced in the information that is retrieved on the repository; and as such, it needs to be balanced such that respective algorithms can be applied on the data. In that regard, the simple way how the minority class can be replicated is performed through SMOTE. Synthetic Oversampling Mining Technique is known as SMOTE. The problem of data imbalance is handled on the data through oversampling of minority labels of the dataset as illustrated in Figure 10.

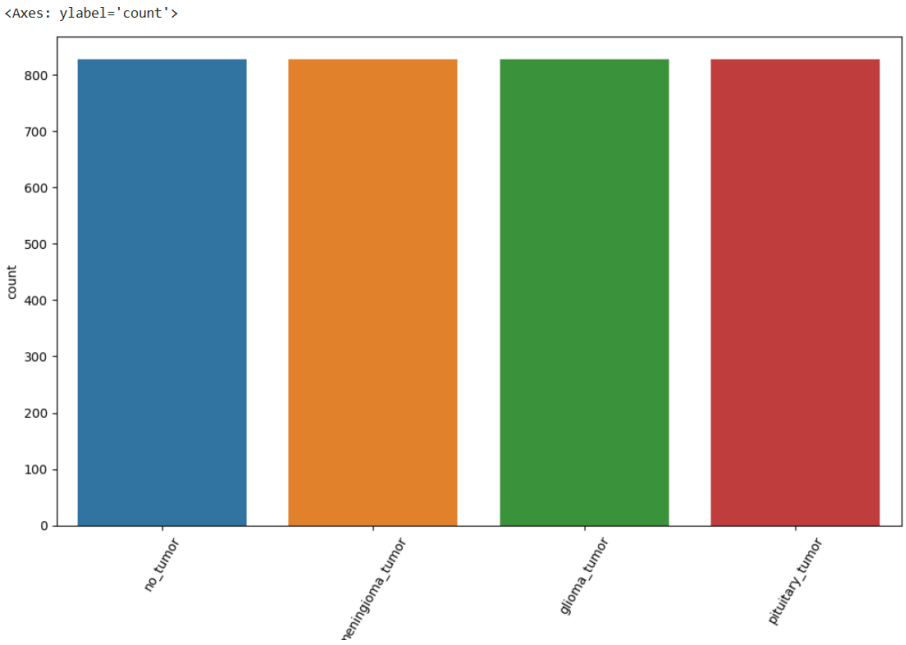


Figure 10: Balanced Data

4 Optimization Techniques

4.1 Optimizer Comparison: SGD vs Adam

The models that classified brain tumors were trained with two different optimization algorithms including Stochastic Gradient Descent (SGD) and Adam. The effect of these optimizers to the performance of the model is as summarized below.

4.2 EfficientNetB4 + ATT with SGD

It refers to a deep learning model, the backbone of which is EfficientNetB4 to which a Convolutional Block Attention Module (CBAM) is connected. The CBAM applies the notion of channel and spatial attention in ensuring that the model focuses on important things in the input. A 4-class classification is performed by using global average pooling and dense layers implemented after the attention block. The code of the model involves SGD optimizer and categorical cross entropy loss.

```python
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB4
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model

# ----- CBAM MODULE -----
def cbam_block(input_feature, reduction_ratio=16):
    channel = input_feature.shape[-1]

    # --- Channel Attention ---
    shared_dense_one = Dense(channel // reduction_ratio, activation='relu', use_bias=True)
    shared_dense_two = Dense(channel, use_bias=True)

    avg_pool = GlobalAveragePooling2D()(input_feature)
    avg_pool = shared_dense_one(avg_pool)
    avg_pool = shared_dense_two(avg_pool)

    max_pool = GlobalMaxPooling2D()(input_feature)
    max_pool = shared_dense_one(max_pool)
    max_pool = shared_dense_two(max_pool)

    channel_attention = Add()([avg_pool, max_pool])
    channel_attention = Activation('sigmoid')(channel_attention)
    channel_attention = Reshape((1, 1, channel))(channel_attention)
    channel_refined = Multiply()([input_feature, channel_attention])

    # --- Spatial Attention ---
    avg_pool = Lambda(
        lambda x: tf.reduce_mean(x, axis=-1, keepdims=True),
        output_shape=lambda s: (s[0], s[1], s[2], 1)
    )(channel_refined)

    max_pool = Lambda(
        lambda x: tf.reduce_max(x, axis=-1, keepdims=True),
        output_shape=lambda s: (s[0], s[1], s[2], 1)
    )(channel_refined)

    spatial_attention = Concatenate(axis=-1)([avg_pool, max_pool])
    spatial_attention = Conv2D(1, kernel_size=7, padding='same', activation='sigmoid')(spatial_attention)
```

Figure 11: Definition of the EfficientNetB4-based model incorporating CBAM and compiled using the SGD optimizer.

```python
# ----- MODEL BUILD -----
def build_cbam_efficientnetb4(input_shape=(128, 128, 3), num_classes=5):
    base_model = EfficientNetB4(include_top=False, input_shape=input_shape, weights='imagenet')
    inputs = Input(shape=input_shape)

    x = base_model(inputs)
    x = cbam_block(x)  # Apply CBAM
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    # x = Dropout(0.)(x)
    outputs = Dense(4, activation='softmax')(x)

    model = Model(inputs, outputs)
    return model

# ----- Instantiate and Compile -----
model = build_cbam_efficientnetb4(input_shape=(128, 128, 3), num_classes=5)
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Figure 12: Implementation of the Convolutional Block Attention Module (CBAM) combining channel and spatial attention mechanisms.

This training log shows the performance of the CNN model with CBAM and Efficient-NetB4 backbone over 10 epochs. The model's accuracy steadily improves, reaching a final training accuracy of approximately 85.8% and validation accuracy of 87.3%, indicating good learning and generalization on the dataset.

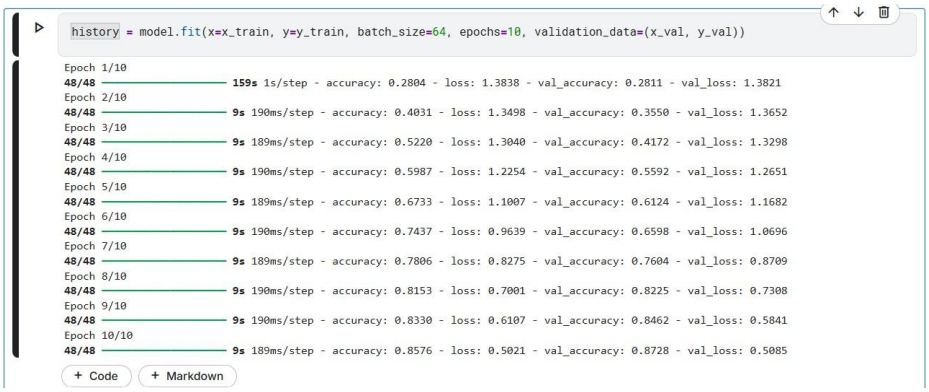


Figure 13: Training and validation accuracy and loss over 10 epochs for the CBAM-enhanced EfficientNetB4 model.

The confusion matrix summarizes the results of the model performance in the tumor categorization into four classes indicating the number of instances that were classified correctly and incorrectly. Classification report includes summary of precision, recall, and F1- score on both classes but the overall accuracy for classification is 81%, which implies a good fit of predictive performance.

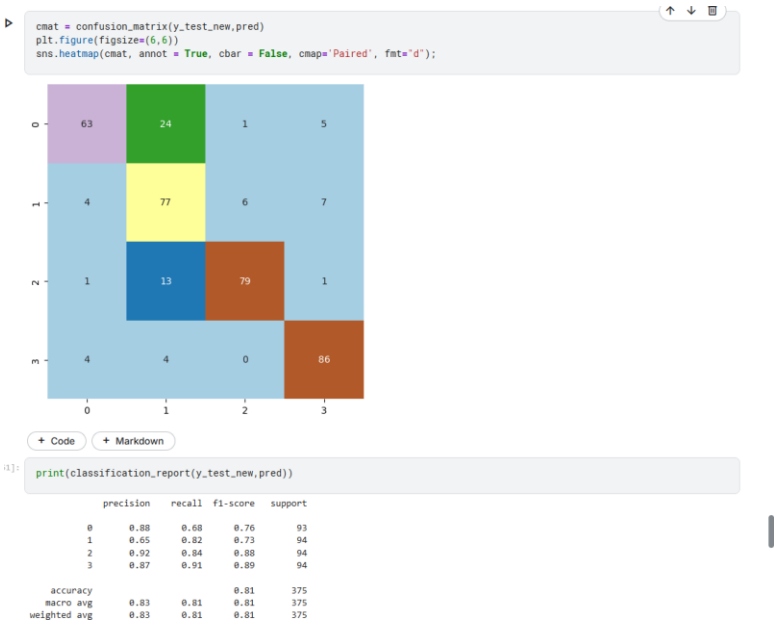


Figure 14: Confusion matrix and classification report for the CBAM-enhanced EfficientNetB4 model on the test dataset.

4.3 EfficientNetB4 + ATT with ADAM

The efficiencyNet B4 is a deep learning model and a CBAM attention block was added to this model to extract features better. Then Applies global average pooling, next the fully connected layers that should have drop out and then it has soft max layer that are used

in multi class classification. Adam optimizer and categorical cross-entropy loss are used to gather the parameters.

```
    refined_feature = Multiply()([channel_refined, spatial_attention])
    return refined_feature

# ----- MODEL BUILD -----
def build_cbam_efficientnetb4(input_shape=(128, 128, 3), num_classes=5):
    base_model = EfficientNetB4(include_top=False, input_shape=input_shape, weights='imagenet')
    inputs = Input(shape=input_shape)

    x = base_model(inputs)
    x = cbam_block(x)  # Apply CBAM
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    outputs = Dense(4, activation='softmax')(x)

    model = Model(inputs, outputs)
    return model

# ----- Instantiate and Compile -----
model = build_cbam_efficientnetb4(input_shape=(128, 128, 3), num_classes=5)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Figure 15: CBAM EfficientNetB4 model with Adam optimizer.

The fisrt image specifies a deep learning model that is provided with EfficientNetB4 integration in conjunction with a CBAM attention blocks to have a superior feature learning. It is made up of global pooling, dense and dropout layers, and softmax output as a multi class classification. The second picture demonstrates the realization of the CBAM module considering channel, spatial attention.

```
[63]:
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.applications import EfficientNetB2
from tensorflow.keras import Input

# --- CBAM Block ---
def cbam_block(input_feature, reduction_ratio=16):
    channel = input_feature.shape[-1]

    shared_dense_one = Dense(channel // reduction_ratio, activation='relu', use_bias=True)
    shared_dense_two = Dense(channel, use_bias=True)

    avg_pool = GlobalAveragePooling2D()(input_feature)
    avg_pool = shared_dense_one(avg_pool)
    avg_pool = shared_dense_two(avg_pool)

    max_pool = GlobalMaxPooling2D()(input_feature)
    max_pool = shared_dense_one(max_pool)
    max_pool = shared_dense_two(max_pool)

    channel_attention = Add()([avg_pool, max_pool])
    channel_attention = Activation('sigmoid')(channel_attention)
    channel_attention = Reshape((1, 1, channel))(channel_attention)
    channel_refined = Multiply()([input_feature, channel_attention])

    avg_pool = Lambda(lambda x: tf.reduce_mean(x, axis=-1, keepdims=True))(channel_refined)
    max_pool = Lambda(lambda x: tf.reduce_max(x, axis=-1, keepdims=True))(channel_refined)
```

Figure 16: Implementation of the CBAM attention block with channel and spatial attention mechanisms.

MODEL TESTING

+ Code + Markdown

[57]:
```
from tensorflow.keras.models import load_model
import tensorflow as tf

# Define the custom lambda functions
def reduce_mean(x): return tf.reduce_mean(x, axis=-1, keepdims=True)
def reduce_max(x): return tf.reduce_max(x, axis=-1, keepdims=True)

model = build_cbam_efficientnetb4(input_shape=(128, 128, 3), num_classes=4)
model.load_weights('/kaggle/working/new_efficienetB4+att.h5')
```

Figure 17: Definition of the EfficientNetB4-based classification model integrating CBAM for improved feature representation.

The following snippet of code loads an already trained EfficientNetB4 + CBAM model and performs a prediction on an indivual MRI scan of a pituitary tumor. It runs preprocessing the image size and transformed to the required format of the input to the model and the model predicts the label of tumor and outputs printed label.

[74]:
```
import tensorflow as tf
from tensorflow.keras.models import load_model
#model=load_model('/kaggle/working/new_efficienetB2+att.h5')
image='/kaggle/input/Training/pituitary_tumor/p (110).jpg'
try:
    img = cv2.imread(image)
    if img is not None :
        img = cv2.resize(img, default_image_size)
        img= img_to_array(img)
        img = np.expand_dims(img, axis=0)
    else :
        print('error')
except Exception as e:
    print(f"Error : {e}")

prediction=model.predict(img)
ans=np.argmax(prediction)
classes=class_labels.classes_
print(classes[ans])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step
pituitary_tumor
```

Figure 18: Code for loading a trained model and performing single-image brain tumor classification.

This code runs a form of explainable AI (XAI) in the form of LIME in explaining model predictions on MRI images. It cleans a given input image, and calculates a prediction function that suits LIME and invokes an initial explainer of LIME to establish a visual explanation which will include vital zones that influence a decision.

XAI

```
[75]:
from lime import lime_image
from skimage.segmentation import mark_boundaries
import matplotlib.pyplot as plt
import numpy as np
import cv2

image='/kaggle/input/Training/no_tumor/2.jpg'
# Function to preprocess image for LIME
def preprocess_image(image_path, target_size):
    img = cv2.imread(image_path)
    img = cv2.resize(img, target_size)
    img = img.astype(np.float32) / 255.0  # normalize if needed
    return img

# Function that LIME will call
def predict_fn(images):
    return model.predict(np.array(images))

# Prepare image for explanation
lime_image_input = preprocess_image(image, default_image_size)

# Initialize LIME explainer
explainer = lime_image.LimeImageExplainer()
```

Figure 19: LIME-based explainability implementation for interpreting brain tumor classification results.

This code runs the LIME explainer on a selected MRI image by which it generates an exposition on the marked spots that have roles in defining the model output. It removes a mask matching the top guessed category and covers the transparent sections of the original image so that the decision regarding the model becomes understandable.

```
from skimage.color import label2rgb
# Run LIME
explanation = explainer.explain_instance(
    lime_image_input,
    classifier_fn=predict_fn,
    top_labels=1,
    hide_color=0,
    num_samples=1000
)

# Get explanation mask
temp, mask = explanation.get_image_and_mask(
    label=explanation.top_labels[0],
    positive_only=False,
    hide_rest=False,
    num_features=3,  #number or areas/superpixels should be highlighted
    min_weight=0.0
)

# Overlay important regions on the original image
original_img_bgr = cv2.imread(image)
original_img_bgr = cv2.resize(original_img_bgr, default_image_size)
original_img_rgb = cv2.cvtColor(original_img_bgr, cv2.COLOR_BGR2RGB)

highlight = mark_boundaries(original_img_rgb / 255.0, mask)
```

```
100% 1000/1000 [00:09<00:00, 104.55it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

Figure 20: LIME explanation overlay showing key image regions contributing to the brain tumor classification.

This picture depicts the original scan of MRI and the LIME explanation on top of it. The areas of significance where the model was relying in discussing the presence of a pituitary tumor is shown in the overlay (drawn in yellow) and can be used by way of illustration on how the model comes up with a conclusion.

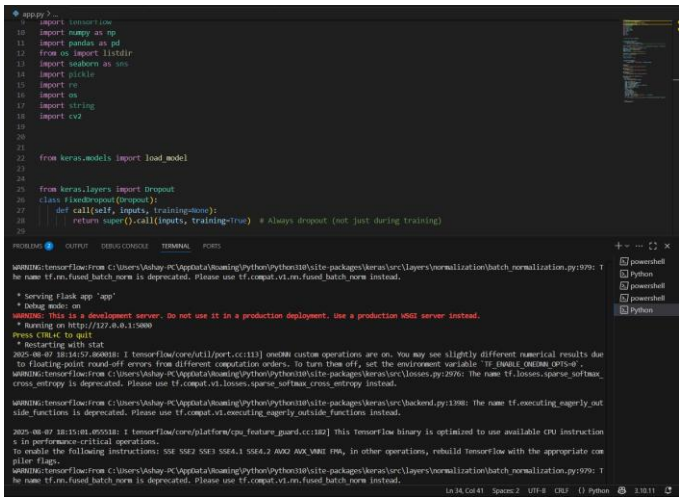


Figure 22: Terminal output indicating the Flask server.

Figure 21: Original MRI image and LIME-generated explanation highlighting critical areas influencing the pituitary tumor classification.

5 User Interface (WEBPAGE):

The output of running a Flask web application to serve a TensorFlow model using the command line. It contains deprecated TensorFlow functions warnings and a note that the server is starting in development mode, and can be accessed locally via URL http://127.0.0.1:5000.

5.1 Webpage

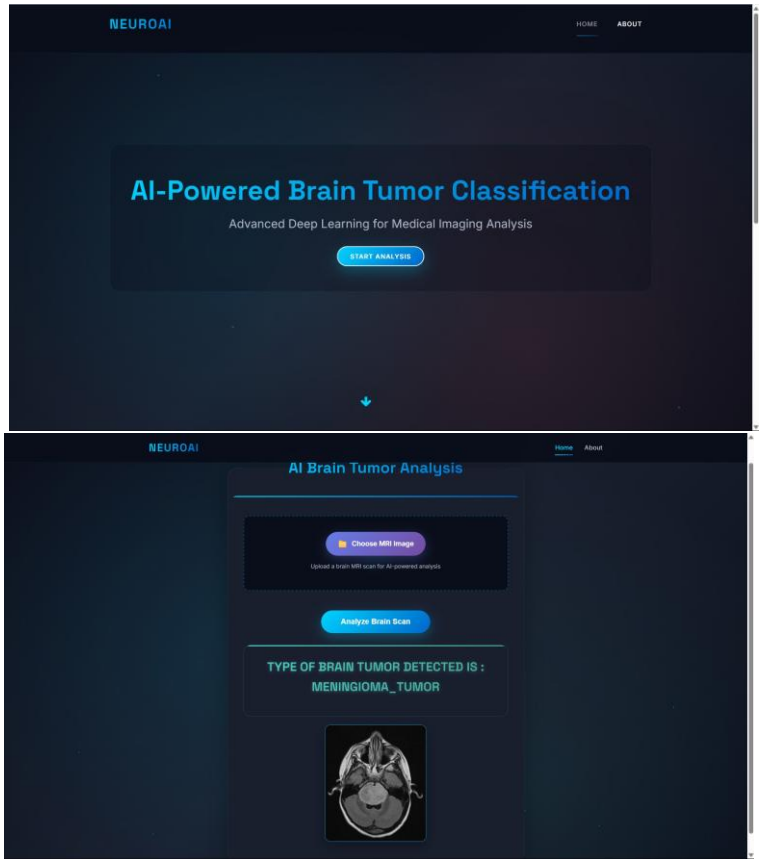


Figure 23: Flask based AI Brain Tumor Analysis web application