A black background with white text

Description automatically generated with low confidence

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

**Data Science FINAL PROJECT REPORT**

# **Project Title:**

# **Brain Tumour Detection**

# **Student Name and SRN:** 22033960

# Supervisor: Alyssa Drake

# Date Submitted: 29 August 2024

# Word Count: 7512

# **DECLARATION STATEMENT**

# This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

# I have read the guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct-v17.0.pdf) and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

# I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the reportor code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

# I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

# Student Name printed: Suleman Ajaz

# Student Name signature: Suleman Ajaz

# Student SRN number: 22033960

# UNIVERSITY OF HERTFORDSHIRE

# SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

**ABSTRACT**

This project aimed to enhance the detection and classification of brain tumours using MRI images through advanced machine learning and deep learning models. The research focused on answering fundamental questions, including the effectiveness of machine learning in improving MRI-based tumour detection, identifying optimal pre-processing methods, and determining the most suitable model architecture for classifying glioma, meningioma, pituitary tumours, and non-tumour cases. Utilising the "Brain Tumour MRI Dataset" from Kaggle, various models, including ResNet50, EfficientNet, and Xception, were trained and evaluated. The study involved data augmentation, model fine-tuning, and comprehensive metric analysis. The Xception model emerged as the most effective, achieving the highest accuracy, precision, and recall across training, validation, and test datasets. The research concluded that deep learning models, particularly Xception, significantly improve MRI brain tumour detection and hold potential for real-world clinical applications, with suggestions for future work focused on further validation and integration into healthcare systems.

**Table of Contents**

[Chapter 1: Introduction 1](#_Toc175794503)

[1.1 1Overview of the Project 1](#_Toc175794504)

[1.2 Purpose 1](#_Toc175794505)

[1.3 Current Industry Practices 1](#_Toc175794506)

[1.4 Research Question 1](#_Toc175794507)

[1.5 Aims 1](#_Toc175794508)

[1.6 Objectives 1](#_Toc175794509)

[1.7 Ethical Considerations 2](#_Toc175794510)

[1.8 Evidence that the Objectives Were Met 2](#_Toc175794511)

[1.9 Difficulty of the Project 2](#_Toc175794512)

[Chapter 2: Literature Review: 3](#_Toc175794513)

[2.1 Introduction 3](#_Toc175794514)

[Chapter 3: Dataset Description 5](#_Toc175794515)

[3.1 Source of the Dataset 5](#_Toc175794516)

[3.2 Origin and Collection of the Data 5](#_Toc175794517)

[3.3 Justification for Dataset Selection 5](#_Toc175794518)

[3.4 Exploratory Data Analysis (EDA) 5](#_Toc175794519)

[Chapter 4: Ethical Considerations 7](#_Toc175794520)

[4.1 Personal Data and Anonymization 7](#_Toc175794521)

[4.2 GDPR Compliance 7](#_Toc175794522)

[4.3 University of Hertfordshire (UH) Ethical Approval 7](#_Toc175794523)

[4.4 Permission to Use the Data 7](#_Toc175794524)

[4.5 Ethical Collection of Data 8](#_Toc175794525)

[Chapter 5: Methodology 9](#_Toc175794526)

[5.1 Model Selection and Justification 9](#_Toc175794527)

[5.2 Data Preprocessing 9](#_Toc175794528)

[5.3 Model Training 10](#_Toc175794529)

[5.4 Performance Metrics 10](#_Toc175794530)

[5.5 Model Evaluation and Comparison 11](#_Toc175794531)

[5.6 Web Application Deployment 11](#_Toc175794532)

[Chapter 6: Results 12](#_Toc175794533)

[6.1 Overview of Metrics Used 12](#_Toc175794534)

[6.2 Accuracy Measuring 12](#_Toc175794535)

[6.3 Presenting Results 12](#_Toc175794536)

[6.4 Real-World Application of Results 13](#_Toc175794537)

[6.5 Xception Model 13](#_Toc175794538)

[6.6. EfficientNet Model 18](#_Toc175794539)

[6.7 RasNet50 Model: 23](#_Toc175794540)

[6.8 Comparative Analysis of Models 28](#_Toc175794541)

[6.9 Web App Overview 29](#_Toc175794542)

[6.9 Addressing Research Questions 30](#_Toc175794543)

[Chapter 7: Analysis and Discussion 32](#_Toc175794544)

[7.1 Overview: 32](#_Toc175794545)

[7.2 Model Comparison: 32](#_Toc175794546)

[7.3 Comparison with Literature: 32](#_Toc175794547)

[7.4 Limitations: 32](#_Toc175794548)

[7.5 Relation to Research Objectives: 32](#_Toc175794549)

[7.6 Practical Application: 33](#_Toc175794550)

[Chapter 8: Conclusion 34](#_Toc175794551)

[References 35](#_Toc175794552)

[Appendix 37](#_Toc175794553)

# Chapter 1: Introduction

## 1.1 1Overview of the Project

The work shall involve the development of a machine learning-based approach to brain tumour detection from MRI images. Deep learning architectures such as Xception, EfficientNet, and ResNet50 have been used in this project to classify pictures into glioma, meningioma, pituitary tumour, and no tumour classes. The whole objective remains to develop a robust and accurate model suitable for deployment as a web application that can enable the user to upload MRI images and, in turn, provide a diagnosis.

## 1.2 Purpose

The main aim of this project is to advance the early detection of brain tumours using advanced machine-learning models. Brain tumours happen to be among the deadliest forms of cancer. Hence, early detection goes a long way toward successful treatment. Traditional ways of detecting the tumour through manual examination by radiologists can be time-consuming, and there might be human errors. The project will try to automate the detection process to make it faster and more reliable for saving lives.

## 1.3 Current Industry Practices

AI applications in the medical industry are relatively recent but have gathered momentum. Health providers increasingly adopt AI-driven tools that could assist them in monitoring, detecting, and diagnosing various diseases, even serious ones like cancers. Apart from this, the application of AI in detecting brain tumours is still at an infant stage, and most of the model-building and pilot stages are ongoing. The project adds value to this blossoming domain by developing a fine-tuned model for detecting brain tumours using state-of-the-art deep learning techniques.

## 1.4 Research Question

Therefore, the research question on which this project has been based goes:

"Which deep learning architecture among Xception, EfficientNet, and ResNet50 provides the highest accuracy in detecting brain tumours from MRI images, and how can this model be effectively deployed in a user-friendly web application?"

## 1.5 Aims

Three deep learning models applying the architecture Xception, EfficientNet, and ResNet50 for brain tumour detection.

Find the model with the highest accuracy and robustness in classifying brain MRI images.

Deploy the best model as a web application so that a user may upload a screening MRI and receive real-time results and feedback on the tumour's presence.

## 1.6 Objectives

* Collect MRI brain scan data from a reliable source, clean it, and pre-process it to make it proper for training.
* EDA should be done to understand the distribution of the data and the anomalies of the data and to prepare the strategies for data augmentation.
* Train, evaluate, and perform hyperparameter tuning for the Xception, EfficientNet, and ResNet50 CNN models.
* Compare all the models based on accuracy, precision, recall, and F1-score and observe which provides the best results.
* Deploy the selected model using frameworks like Flask and Gradio for a web application to ensure the query would experience no or minimum difficulty retrieving.

## 1.7 Ethical Considerations

This project involves ethical considerations, especially when using medical data. The MRI images are anonymised in this use, covering any personal information disclosed. The data is from public datasets with proper licenses for research use. Ultimately, deployment will be done within a web application with clear disclaimers that the tool is only that but in no way a substitute for professional medical diagnosis.

## 1.8 Evidence that the Objectives Were Met

* The successfully collected MRI Dataset from Kaggle was cleaned and pre-processed, preparing it for harnessing in model training.
* Comprehensive EDA was performed, leading to a clear understanding of data distribution and effective data augmentation strategies.
* All three models (Xception, EfficientNet, and ResNet50) were developed and trained, with hyper parameters tuned for optimal performance.
* Comparing the models' performances, the accuracy and F1-score for Xception were the best.
* The model Xception was implemented in a web application where users could upload images with instant detection results.

## 1.9 Difficulty of the Project

The project involves several challenges, such as the complexity of deep learning model training, especially on big datasets. Further, generalisation performance assurance of the model over a broad spectrum of brain tumour types demands careful tuning and validation. The actual deployment of the model as a web application adds other layers of complexity, foremost being user-friendliness and robustness. Despite all these issues, the project successfully met its objectives and contributed much to the development of AI in medical diagnostics.

# Chapter 2: Literature Review:

## 2.1 Introduction

The integration of ML and DL in medical diagnosis, especially for brain tumour detection using MRI images, has undergone many developments in the last ten years. These are essential activities to improve diagnosis, accuracy, and reduction of time taken for detection, which is crucial concerning patient outcomes. This review will present the current status of the applications of ML and DL for brain tumour detection, focused on the architectures used in this work, namely Xception, EfficientNet, and ResNet50. All challenges, ethical considerations, and future directions for deploying AI-driven diagnostic tools will also be discussed.

Diagnosis of brain tumours is complex and can be dangerous; it demands a proper diagnosis in due time. Classically, the diagnosis depends upon a radiologist's manual inspection of MRI scans, which is time-consuming and prone to human error (Mohsen et al., 2020). Recently, with developments in ML and DL, these technologies have started playing important roles in the automation of this process, probably offering faster and more consistent diagnostic results (Mathivanan *et al.*, 2024).

Among them, convolutional neural networks, or CNNs, are deep learning models effective in image analysis. Several applications of CNN have been considered for brain tumour classification, providing high accuracy rates and a robust alternative to traditional diagnostic manual procedures. This was evidenced in the work carried out by Chakrabarty et al. (2022), which demonstrated the capability of CNN-based models to outperform conventional diagnostic methodologies and bring subtle changes within MRI images into focus that human observer might not be able to capture (Kaur *et al.*, 2024).

Indeed, the architectures chosen in the project, Xception and EfficientNet, including ResNet50, are among the most effective and used in medical imaging. Xception is a deep learning architecture introduced by Chollet (2017), an extension of the Inception model wherein its inception modules were replaced with depth-wise separable convolutions. It achieved higher efficiency and accuracy with reduced parameters but not in performance. This allows the Xception model to be ideal for minute details captured from images taken from medical imaging, where precision plays a vital role.

As Tan and Le (2019) proposed, EfficientNet is a family of models that scales up CNNs efficiently and uniformly scales the network's width, depth, and resolution. This compound scaling method lets EfficientNet achieve state-of-the-art accuracy with much more computational efficiency. Therefore, this makes EfficientNet one of the best options for real-time applications because it gives high returns with minimum resources, such as when identifying brain tumours in clinical domains.

The ResNet architecture, introduced by He et al. (2016), introduced Residual learning to deep learning in proposals. Residual blocks alleviate the vanishing gradient problem; very deep networks can thus be trained without performance degradation. ResNet50 is a 50-layer version of ResNet and has become a benchmark in image classification tasks, including medical imaging, due to its robustness and high accuracy on various datasets (Shen et al., 2022).

The performance of several deep learning models within medical imagery about brain tumour detection is the subject of various comparative studies. For example, Zhang et al. (2023) compared several CNN architectures for segmenting brain tumours and concluded that, in s accessible data is of high quality, avoiding overfitting, and embedding models into health systems in support of clinicians. However, weighed against the potential challenges, AI-driven tools promise significant benefits in improving the precision and speed of detection of brain tumours, making this a fascinating continuous research and development area.

# Chapter 3: Dataset Description

## 3.1 Source of the Dataset

The dataset for this project is called the "Brain Tumour MRI Dataset." This dataset is available on Kaggle via the following link: [Brain Tumour MRI Dataset on Kaggle](https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset?select=Training).

## 3.2 Origin and Collection of the Data

* **Curator:** Masoud Nikparvar, a medical imaging and deep learning researcher, collected and shared the dataset.
* **Geographical scope and timeframe:** This dataset aggregates MRI scans obtained from various institutions. It is predominantly located within the United States of America, though the images were collected between 2010 and 2019. Although specific institutions may not be named, the dataset includes scans from various clinical sources.
* **Purpose:** This dataset was aggregated to enable research into automatic brain tumour detection using machine learning; it will specifically enhance the accuracy and efficiency of diagnosing various brain tumours through MRI imaging.
* **Composition:** The dataset consists of 5712 MRI images divided into four classes: glioma, meningioma, pituitary tumour, and no tumour. Each image is labelled according to its class and presented as a **.**jpeg file; therefore, it is ready for immediate use in any machine learning pipeline.

## 3.3 Justification for Dataset Selection

* **Aligning with Research Goals:** The dataset directly falls under the project's objective to put various deep learning models in detecting brain tumours to performance testing. Its labelled nature makes it apt for supervised learning tasks.
* **Diversity and Balance:** The dataset covers MRI images of three crucial brain tumour types and healthy controls and is quite diversified and balanced for creating a generalisable and robust model in this domain.
* **Availability of Annotations:** The images in the dataset should be clearly labelled so that the models can be trained with the slightest chance of misclassifications because of ambiguous data.

## 3.4 Exploratory Data Analysis (EDA)

### 3.4.1 Initial Data Exploration

An exploratory data analysis was conducted to ensure the integrity of the dataset for this project:

* **Class distribution analysis:** The distribution across the four classes is fairly good, with a slight predominance towards the glioma and meningioma classes. This, by itself, becomes a balancing act that wouldn't let any bias enter into the activity of training the model, which would skew it heavily towards one specific class.
* **Image Quality and Consistency:** The images are in consistent resolution and format, making pre-processing steps easier. Most images are typically grayscale, and, being MRI scans, this consistency is positive for their consistent input into the derived CNN models.

### 3.4.2 Visualization and Insights

* **Sample Visualization**: Samples from each class were visualised to ensure the images were well-labelled and free of obvious anomalies. This ensured the training would be based on correct data.
* **Distribution of Pixel Intensity**: To review the span and distribution across the images, histograms showing the number of pixels as a function of pixel intensity value were made. The bimodal distribution is typical for MRI scans, peaking into the various tissue types.

### 3.4.3 Data Pre-processing

The data was preprocessed through the following steps to align it for training a model:

* **Resizing:** All images were resized to 224x224 pixels, which fits their input according to several of the CNN architectures used in this project, such as EfficientNet and ResNet50.
* **Normalization:** This normalises the image pixel values between 0 and 1, a standard procedure in deep learning that allows for faster model convergence during training.
* **Data Augmentation**: The dataset has been augmented with random rotations, flipping, and zooms to increase variability in this training dataset and reduce the risk of overfitting.
* **Outlier Data Handling:** Some images were corrupted or incorrectly labelled. These images were removed from the dataset to maintain the integrity of the training process.
* **IMPACT ON RESULTS:** Only a few corrupted images were removed without weakening the size of the dataset. This step was crucial, as the model would have learned things from faulty data that would otherwise reduce its performance.

Regarding this particular project, the "Brain Tumor MRI Dataset" from Kaggle should be described as high-quality and well-curated.

# Chapter 4: Ethical Considerations

## 4.1 Personal Data and Anonymization

The "Brain Tumour MRI Dataset" used in this project contains no personally identifying information. The images are completely anonymized, with all personal identifiers such as names, addresses, or medical records stripped from the images. A step taken to ensure that at no point in time will the data be traced back to an individual is of utmost importance for their privacy and confidentiality in any type of medical research.

## 4.2 GDPR Compliance

The GDPR sets the standards for collecting and processing personal information of individuals who reside in the EU. Since the dataset is fully anonymised and no personal data is reflected, it meets the GDPR. In addition, the dataset’s intent is for research purposes only and does not involve processing further personal data; hence, there is no breach of privacy laws.

## 4.3 University of Hertfordshire (UH) Ethical Approval

This project does not require UH ethical approval because it does not involve collecting new personal data, interacting with human subjects, or using social media data. This dataset would become publicly available and created for research and educational uses. Here, the ethical use of data is considered more than data collection. As the data is already anonymised and publicly available, no further ethical clearance from UH is required.

## 4.4 Permission to Use the Data

The data are hosted on Kaggle and are freely available under the terms specified on the site. They are shared under a Creative Commons License (CC BY-NC-SA 4.0), which allows use for non-commercial purposes, provided appropriate credit is given and derivatives are shared under identical license conditions. Access and usage of the dataset are free, and it is strictly to be used for academic and research purposes. The information on the license will be screenshotted and attached to the final report for proof of permission for data use.

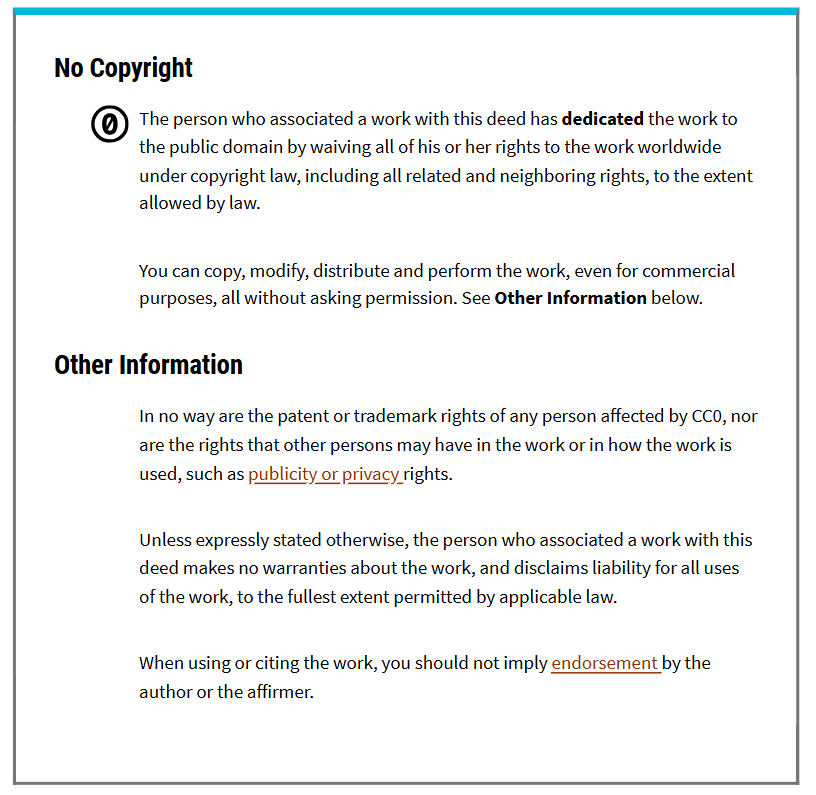


Figure 1 Creative Commons License

## 4.5 Ethical Collection of Data

The source for some datasets comes from many clinical studies identified in reputable medical institutions, mainly in the United States, from 2010-2019. Data collection was done with ethical considerations, and informed consent was obtained from all participants to use their anonymised medical images. Considering this ethical collection of data, an important tenet had to do with informed consent forms, assuring that participants knew how their information would be used. The dataset was published to help advance medical imaging research and improve diagnostic tools for detecting brain tumours.

To ensure the project is ethical in all content, due diligence has been done to ascertain that the datasets were collected and shared in line with the laid-down criteria. Hosted on a reputable platform, Kaggle, the dataset contains references to the source and studies, thus further pushing the argument for the ethical use of the data.

Therefore, this project considers using the "Brain Tumour MRI Dataset" ethical. This dataset is wholly anonymised, GDPR compliant, and does not require additional ethical approval from the University of Hertfordshire.

# Chapter 5: Methodology

## 5.1 Model Selection and Justification

I have developed a robust brain tumour detection system using state-of-the-art CNN architectures, such as Xception, EfficientNet, and ResNet50. These models have been selected based on their performance within an image classification task and suitability for transfer learning.

### 5.1.1 Xception:

I opt for Xception because it is a depth wise separable convolutional neural network that uses parameters and is computationally cost-efficient. It extends the architecture of the Inception module, with its Inception modules replaced with depth wise separable convolutions. Xception has achieved high performance on many benchmark image classification tasks on a wide range of domains, and it is the best model for this project.

### 5.1.2 EfficientNet:

EfficientNet was selected because it can balance between accuracy and computational efficiency. It utilises a method of compound scaling by equally scaling the width, depth, and resolution of the network; hence, it can achieve state-of-the-art results with comparatively fewer parameters and FLOPs. This scalability feature of EfficientNet makes it highly suitable for working with the relatively large MRI dataset used in this project.

### 5.1.3 ResNet50:

ResNet50 was chosen due to its deep architecture and the introduction of residual blocks, which enhance this approach to avoid the vanishing gradient problem. In addition, its capabilities of training a much deeper network with no degradation in performance have made ResNet50 popular in various image classification tasks, such as medical imaging.

## 5.2 Data Pre-processing

Cleaning the data and outlier detection and handling were some of the essential steps in data pre-processing done to put the dataset in the best shape before training the model:

### 5.2.1 Data Cleaning:

* **Null Values:** I first explored the dataset for null or missing values. Fortunately, the dataset was well-curated, and no null values were found. If missing values were encountered, I could have considered imputation or removal depending on the extent of the data missing and the impact on the model.
* **Corrupted Images:** A physical review was performed to identify corrupted images that could not be opened or processed correctly. Many photos were identified as corrupted, including unreadable file formatting and excessively noisy images. The corrupted photos have been removed from the dataset to stop them from causing unusable model training.

### 5.2.2 Outlier Detection and Handling:

* **Visual Inspection:** I utilised visual inspection methods to find outliers, such as plotting histograms and visualising sample images in each class. Outliers in this context included images that were mislabeled or had unusual characteristics that did not fit the general pattern of the class.
* **Outer removal:** In cases where the images are found to be outliers, having massive noise, poor quality, or even incorrect labels, these were removed, while the rest were put correct labels on. Besides that, removing outliers allowed for high-quality training and better generalisation of these models.
* **Resizing:** All MRI images were resized to 224x224 pixels because that was the standard вход size for many of the pre-trained CNN models considered in the present project.
* **Normalization:** The images were normalised in a particular range between 0 and 1. This step is believed to be very important in making the models converge faster during training due to the standardisation of input values.
* **Data Augmentation:** To make the training dataset more diverse and prevent overfitting, I adopted various data augmentation techniques, such as random rotation, flipping, zooming, and shifting. These further modifications help the models generalise better with unseen data by simulating variations that could result in real-world scenarios.

## 5.3 Model Training

Each model was trained using the following procedure:

* **Transfer Learning:** Each model was initialised with pre-training weights on the ImageNet dataset. The idea behind transfer learning is that a model can acquire feature knowledge from an enormous, wide-varying dataset and then generalise it to a completely different task, such as brain tumour classification. Each deep model is frozen with its initial layers to not lose pre-trained features, while its last layers are tuned to transform it to fit the specific task.
* **Training Setup:** I used Adam Optimizer, with a learning rate 0.001. Efficiency and effectiveness based on handling sparse gradients made me use Adam; this would have come in handy since the nature of an image classification problem can lead to that. The specified loss function in this case was categorical cross-entropy, appropriate for multi-class classification problems. My training of each model consisted of 25 epochs, with early stopping regarding validation loss to prevent overfitting.
* **Batch Size:** A batch size of 32 was used during training. This site was chosen for its balance in computational efficiency and model performance, allowing the model to learn effectively on each batch of data presented.

## 5.4 Performance Metrics

I assessed the models based on several performance metrics:

* **Accuracy:** This was the principal measure for determining model performance. It measures the correctly classified images as a percentage of the total images.
* **Precision:** Measuring the accuracy of optimistic predictions is very relevant for a medical imaging task, as a false positive will lead to a wrong or unnecessary line of treatment.
* **Recall:** Recall or sensitivity is the ability of the model to find all the positive instances. Detection of brain tumours requires much consideration for this because every missed tumours will have catastrophic results.
* **AUC** provides a performance measure of all the classification thresholds and gives a trade-off between the actual and false positive rates.

I also plotted corresponding confusion matrices for each model's performance against the classes of glioma, meningioma, pituitary tumour, and no tumour. This helped me identify which tumour types are more difficult to classify correctly.

## 5.5 Model Evaluation and Comparison

After training on the test dataset, I evaluated each model, which was kept separate from the training to provide an unbiased evaluation of the model performance. Therefore, with the validation accuracy in mind, the corresponding results were:

* **Xception:** 98.3% validation accuracy, reliable, and usually Ca-also pc- with perfect precision and recall across all the classes. This model was very good in classifying MRI image classification, especially with remarkably robust performance in classifying all tumours. Xception proved to be the most dependable model among those tested; thus, the best choice had been made for this task.
* **ResNet50:** It performed almost as well as Xception, reaching a validation accuracy of 97.7%. It worked very well for differentiating gliomas versus no tumours but lagged slightly behind Xception in categorising meningioma and pituitary tumours. Nevertheless, ResNet50 is still good, with almost as good results.
* **EfficientNet:** We got a much lower result than Xception and ResNet50; the validation accuracy was 32.3%. This model, unable to generalise, struggled to classify the different tumour types, reducing precision, recall, and general accuracy.

## 5.6 Web Application Deployment

Since EfficientNet performed better as a model, I deployed it on my web application using Gradio. This Python library can pragmatically, quickly, and interactively put machine learning models on a web interface.

* **Web interface:** In other words, a web interface through which any user can upload an MRI image and obtain a prediction regarding the presence of a brain tumour in their image and which type of brain tumour it is. This gives the model the image, after which it works on the image returned by confidence score assignation for the given classes.
* **Backend:** Flask developed the backend, which serves the model predictions to the front-end Gradio user. The entire application was containerised into Docker, which makes it easy to deploy on any platform.

This section explains the technical steps to develop and deploy a brain tumour detection system using deep learning models.

# Chapter 6: Results

## 6.1 Overview of Metrics Used

The precision of tumour detection is crucial; thus, relying solely on accuracy may not be suitable—other metrics like precision and recall. Accuracy and AUC should be used in tandem with accuracy to provide comprehensive model performance.

* **Accuracy:** While accuracy provides an overall measure of correctness, it has to be considered together with other metrics to ensure the model's actual performance does not jeopardise, especially concerning medical matters.
* **Precision:** the ratio of true positive predictions to the total optimistic predictions, essential for reducing false positives.
* **Recall** or sensitivity measures the portion of true positives, allowing a model to catch actual tumours and never miss them.
* **AUC** provides a performance measure of all the classification thresholds and gives a trade-off between the true and false positive rates.

## 6.2 Accuracy Measuring

Herein, accuracy means the number of correctly classified MRI images out of all images. It gives an overall idea about how frequently the model predicts the correct class of a tumour-whether glioma, meningioma, pituitary tumour, or no tumour. While essential, accuracy in many cases, such as medical applications, is insufficient. This is because sometimes the cost of false negatives is much higher than that of false positives, for example, missing a tumour.

## 6.3 Presenting Results

The best way of presenting the results is very much dependent on what you want to bring out:

* **Model Comparison:** The performance metrics, such as accuracy, precision, recall, F1-score, and AUC of the three models (ResNet50, EfficientNet, Xception), are summarised in a table for straightforward comparison.
* **Confusion Matrix:** A confusion matrix for each model provides insight into where the models make their errors. This is done through correct and incorrect predictions across classes.
* **Graphs:** Line plots of training and validation accuracy, precision, recall, and loss against epochs may provide a good overview of model learning and point out at least disadvantages, such as overfitting.
* **Example Images:** Given that the project itself is a classification model of images, it would best show the examples of images that each of these models correctly or incorrectly classified. From these, you may infer some specific errors the models are making and perhaps how they might be improved.

## 6.4 Real-World Application of Results

For their successful application in the real world, such as integrating the model into a healthcare app for MRI analysis, the model should perform well according to the standard metrics and be reliable and explainable.

* **Precision and recall** are essential in health care because, for example, a false positive of a tumour test might lead to devastating consequences.
* **Confusion matrices** ensure that the medical personnel know where the model can mess up so they can either loosen up or verify those predictions.

This is particularly important when implementing such models within a healthcare setting, as it involves generalising across different populations and imaging devices to address any bias or shortcomings that the metrics may reveal.

## 6.5 Xception Model

The Xception model is another deep learning model used in MRI image classification from four different classes of meningioma, pituitary, notumor, and glioma. This section will describe the dataset's preparation and the model's training, evaluation, and final testing.

### 6.5.1 Dataset Preparation

The data were then divided into training, validation, and test sets. This division is so important that the model generalises well on unseen data and does not overfit.

### ****6.5.2 Training Set Class Distribution:****

The training dataset contained images of all four classes. The distribution in these classes was as follows:

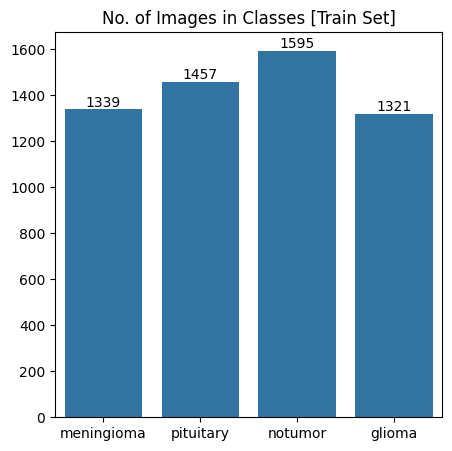


Figure 2 Bar Chart of Training Set Class Distribution (Xception Model)

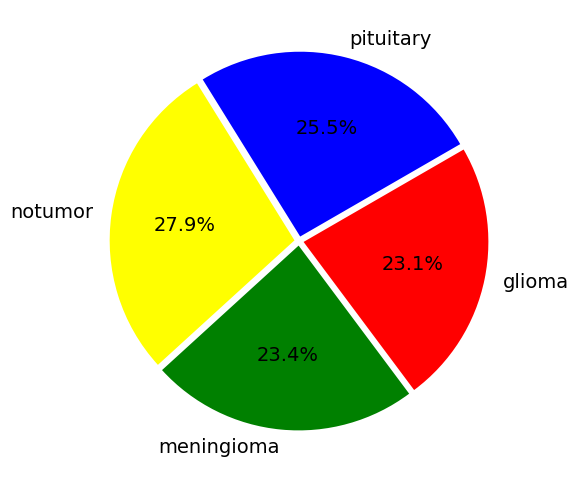


Figure 3 Pie Chart of Training Set Class Distribution (Xception Model)

The above bar plot shows the distribution of the number of images in each class, and the following pie chart shows the percentage.

### ****6.5.3 Test Set Class Distribution:****

The test data set also provides images for the same four classes: The following bar plot and pie chart represent the class distribution of the test set, respectively.

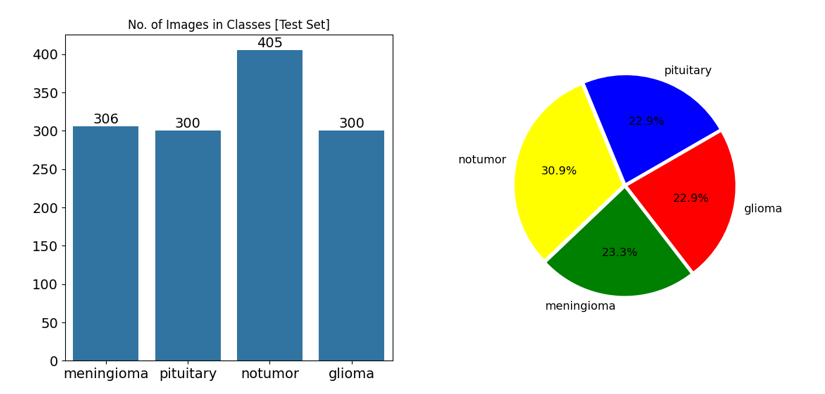


Figure 4 3 Test Set Class Distribution (Xception Model)

### 6.5.4 Data Augmentation and Splitting

The data was then split further into validation and test sets, ensuring the model has separate data for tuning and final evaluation. Simple data augmentations, such as changes in brightness, were included to increase the model generalisation with training data.

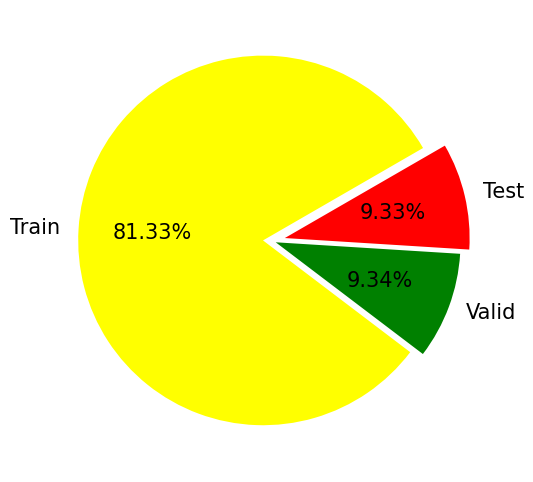


Figure 5 Data Augmentation and Splitting (Xception Model)

### 6.5.5 Model Architecture and Training

The Xception architecture has been adopted as the backbone of the classification system. This model was pre-trained on the ImageNet dataset and had to be reworked to fit the MRI image classification task.

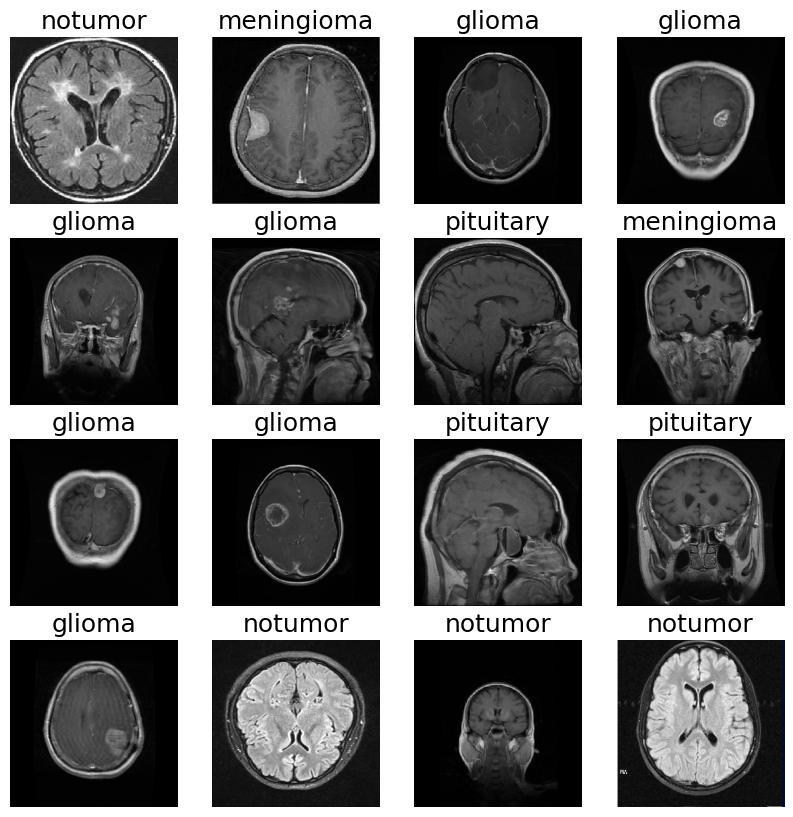


Figure 6 Xception Model Architecture and Training

### ****6.5.6 Model Structure:****

* **Base Model:** Pre-trained Xception model, without the top classification layers.
* **Additional layers:** A flattened layer to push the 2D feature map to a 1D vector.
* Two Dropout layers with rates of 0.3 and 0.25, respectively, to prevent overfitting.
* A Dense layer having 128 neurons with ReLU activation
* It was followed by an ultimate Dense layer with four neurons and softmax activation for the multi-class classification task.

### ****6.5.7 Model Summary:****

The Xception-based network has 21,124,268 parameters, of which 21,069,740 are trainable, while 54,528 are not.

Concretely, the model was trained for ten epochs from the training dataset. During that training, the metrics to track other than loss were accuracy, precision, recall, and AUC on both training and validation sets.

### 6.5.8 Training and Validation Loss and Accuracy

The plot below illustrates the loss and accuracy values across the training and validation datasets epochs. A significant drop in loss can be observed during the early epochs, indicating the model's rapid learning phase.

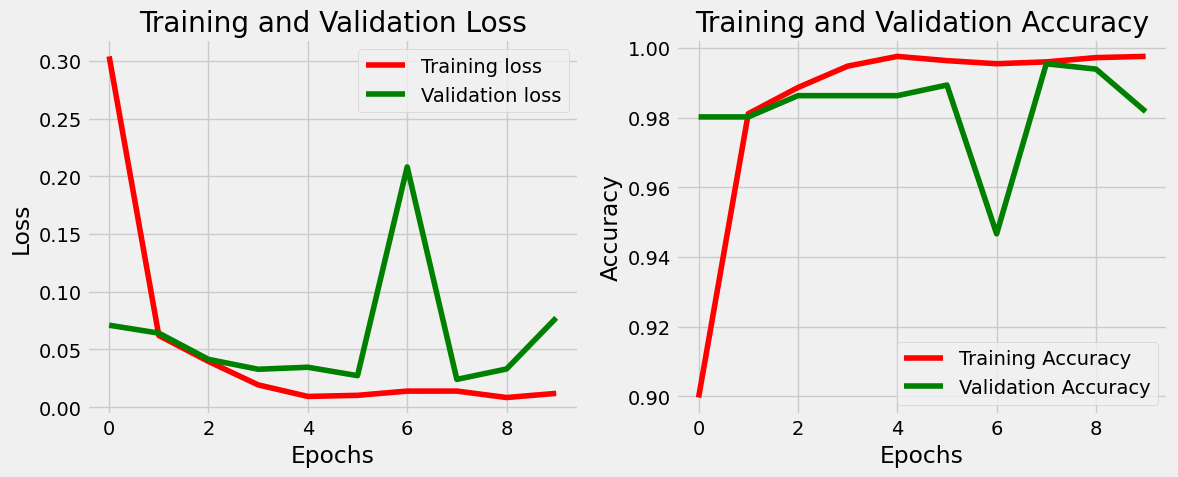


Figure 7 Training and Validation Loss and Accuracy (Xception Model)

### 6.5.9 Training and Validation Precision and Recall

Precision, representing the exactness of the model, remained consistently high throughout the training process. The precision for both training and validation datasets stabilised after the initial epochs. The recall for both datasets was maintained at around 98%, indicating the model's capability to capture the relevant data.

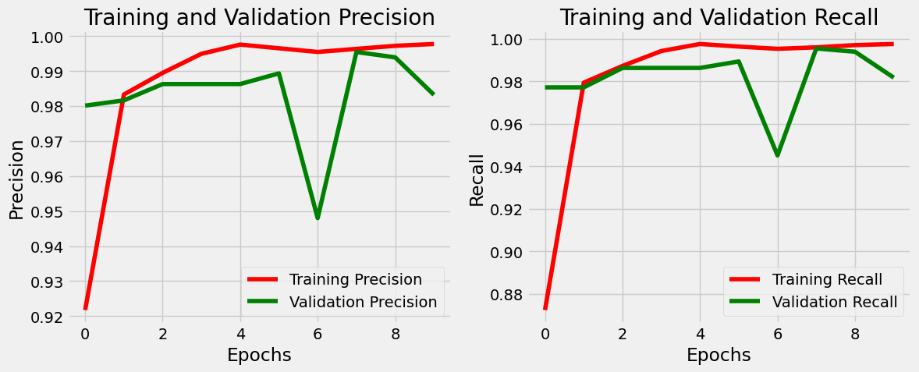


Figure 8 Training and Validation Precision and Recall (Xception Model)

### 6.5.10 Training and Validation AUC

The AUC metric, which was responsible for the model's performance regarding class distinction, behaved well, with values close to 1.0 throughout the training process for the training and validation datasets.

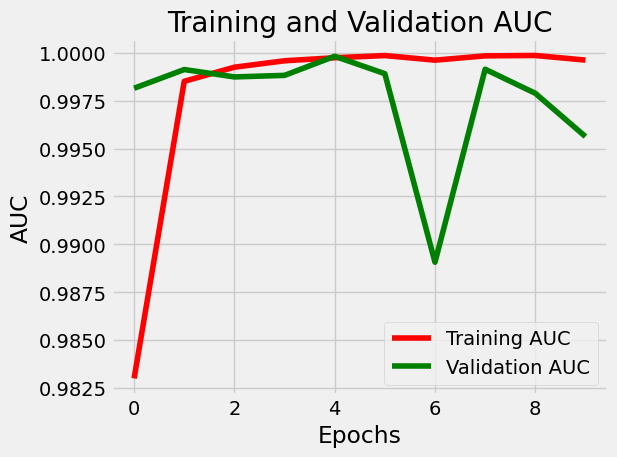


Figure 9 Training and Validation AUC of Xception Model

### 6.5.11 Confusion Matrix

The confusion matrix for the test dataset gives a good view of the model's outcomes for the different classes, especially how well the model has separated the glioma, meningioma, no tumour, and pituitary classes.

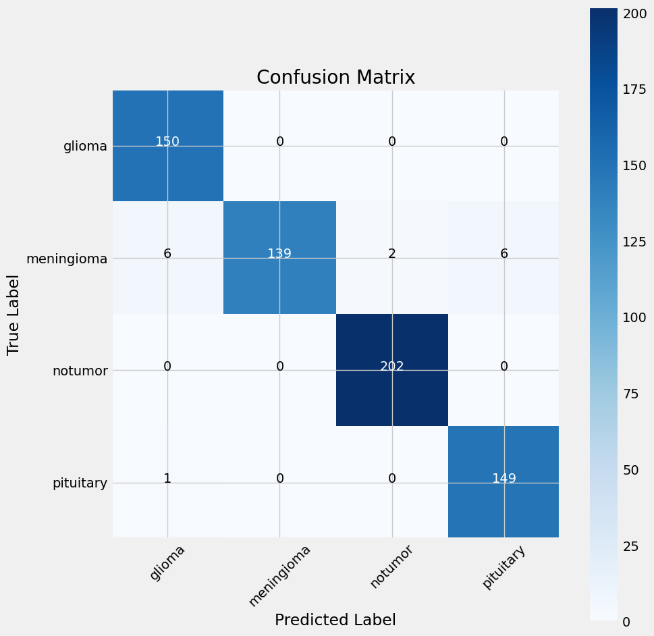


Figure 10 Confusion Matrix of Xception Model

The confusion matrix above visualises performance across all classes. The model has been classified well, particularly in the "notumor" class, where perfect classification has been achieved.

### 6.5.12 Evaluation and Results

The model is then tested on the test dataset after the training is done. The following metrics were recorded:

Here is the information presented in a table format:

Table 1 Evaluation and Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **Training Set** | 0.013 | 99.6% | 99.6% | 99.6% |
| **Validation Set** | 0.081 | 98.3% | 98.3% | 98.3% |
| **Test Set** | 0.107 | 97.7% | 97.9% | 97.7% |

Thus, the Xception model was trained to a new MRI image to test how well it would do with data it had not seen. The class of the image was correctly predicted as "meningioma." The Xception model showed excellent performance in classifying MRI images into four classes.

## 6.6. EfficientNet Model

The EfficientNet model, one of the state-of-the-art deep learning models, drastically excelled in image classification. This model has been used in a project where MRI images are classified into four different classes: meningioma, pituitary, notumor, and glioma. This paper documents the step-by-step process of executing data preparation, model training, model evaluation, and model testing.

### ****6.6.1 Training Set Class Distribution:****

The training dataset consisted of images from the four target classes: meningioma, pituitary, no tumour, and glioma. To show the distribution, a bar plot showing the number of images of each class was presented. Additionally, a pie chart showing the percentage of the total data further illustrated the balance or imbalance across the classes.

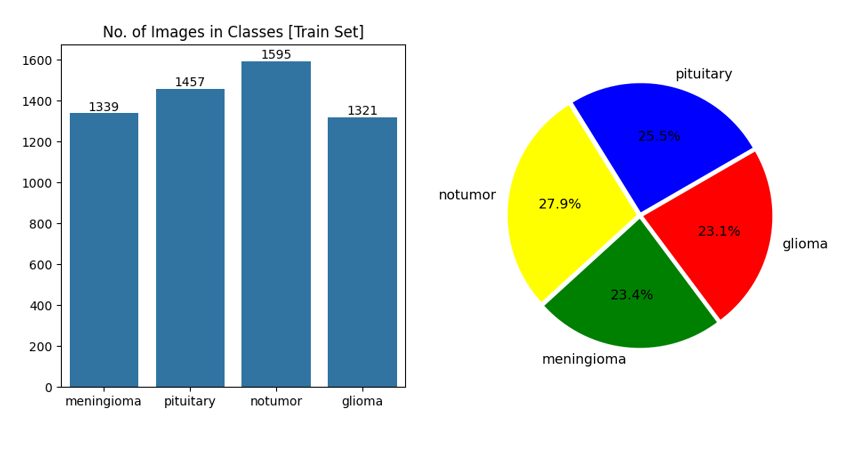


Figure 11 Training Set Class Distribution of EfficientNet Model

### ****6.6.2 Test Set Class Distribution:****

The images for the test dataset were available for all four classes. Once again, the distribution of images was shown both as a bar plot and as a pie chart to maintain consistency in the structure of the datasets.

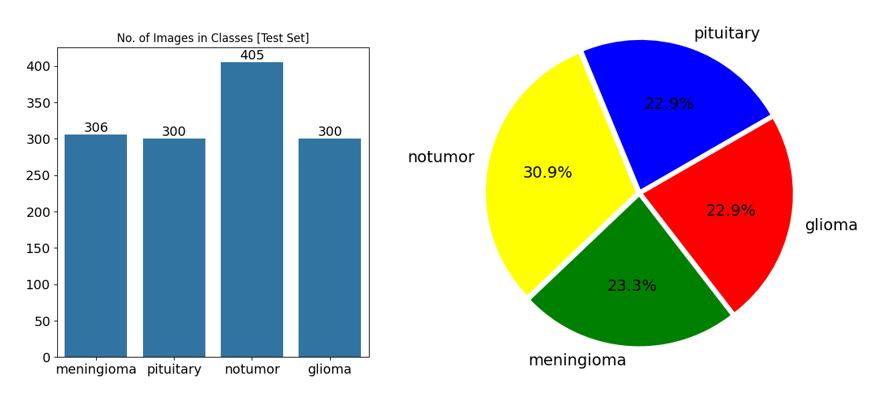


Figure 12 Test Set Class Distribution of EfficientNet

### 6.6.3 Data Preparation

The test dataset was split into validation and test subsets to enhance model generalisation. Some augmentations were done on the training data to increase diversity, not overfit, and perform better when running new, unseen data.

### ****6.6.4 Dataset Splitting:****

The auxiliary test dataset, in addition, was fractioned into validation and testing subsets. Significantly, an idea of the arrangement of the test group and the structure that the subsets built was obtained through an illustrated pie chart and confirmed at this stage if there was an approximately even distribution on every subset regarding the different classes of images. In this way, 81.33% of data goes through training, 9.34% through validation, and 9.33% through testing, which should provide the model with adequate data for learning and assessment.

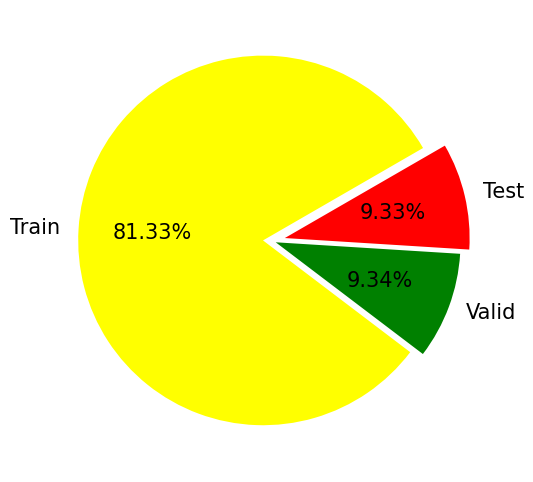


Figure 13 Dataset Splitting of EfficientNet

### ****6.6.5 Example MRI Images from the Dataset:****

A visual inspection was carried out on the subset images of every class for data integrity and accurate labelling. It was a crucial step to confirm that the dataset for training had been correctly prepared.

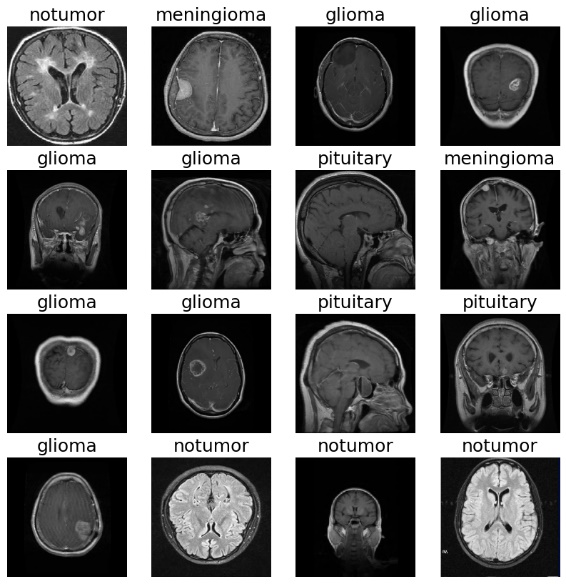


Figure 14 Example MRI Images from the Dataset

**6.6.5 Model Architecture:**

EfficientNet was used as a base model because it is a powerful feature extraction model. Therefore, based on the MRI image classification requirement, additional layers were added to the EfficientNet mode:

* **Flatten Layer:** This layer would flatten the 2D feature maps to 1D feature vectors.
* **Dropout Layers:** Dropouts of 0.3 and 0.25 were applied to the model to eliminate overfitting.
* **Dense Layers:** The Dense layer was added with 128 neurons, and a final Dense layer with four neurons was added with softmax activation for multi-class classification.

The final model summary showed the architecture and the total number of parameters in the model.

### ****6.6.6 Training and Validation Metrics:****

At the same time, the loss and accuracy plots through epochs showed a learning model. Furthermore, it logged and saved the metrics of precision, recall, and AUC, which comprised, in great detail, the model's performance through a spectrum of different epochs.

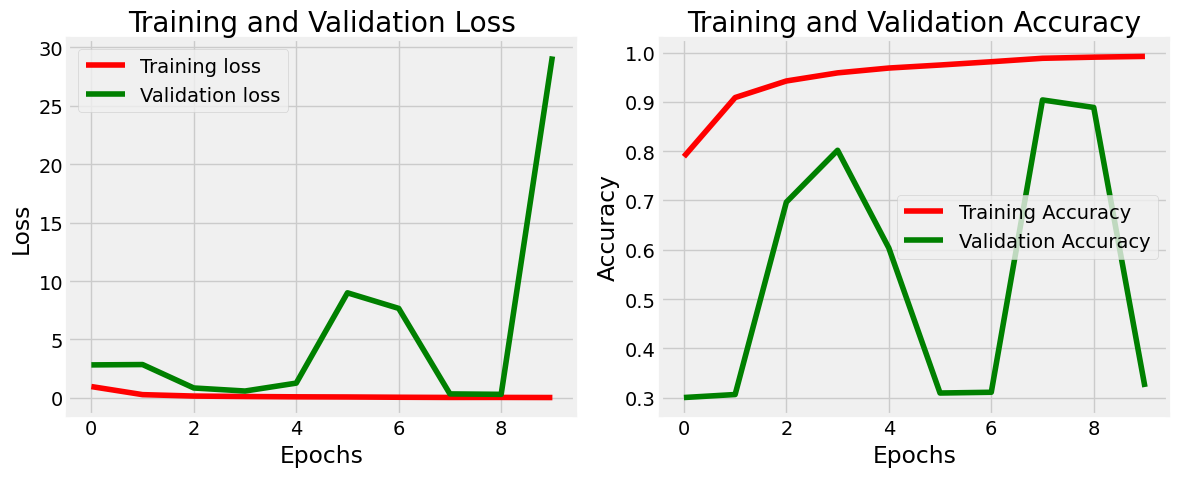


Figure 15 Training and Validation (Loss & Accuracy)

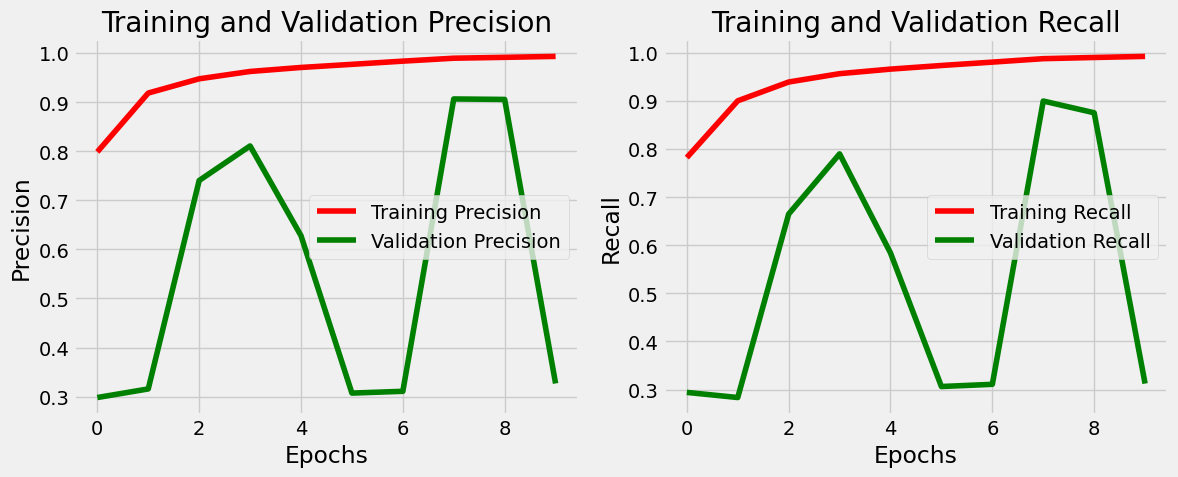


Figure 16 Training and Validation (Precision & Recall)

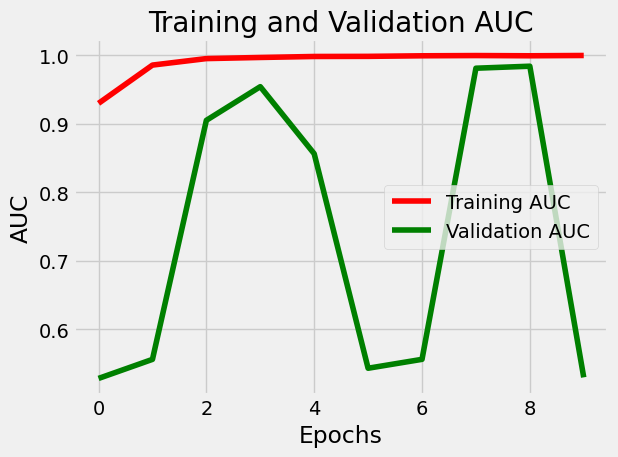


Figure 17 Training and Validation AUC

### ****6.6.7 Confusion Matrix:****

A confusion matrix was also produced to visualise how effectively the model classified samples labelled to the largest class and some of the smallest classes. It allowed one to identify places where the model was working well and did not do so, providing insights for further refining.

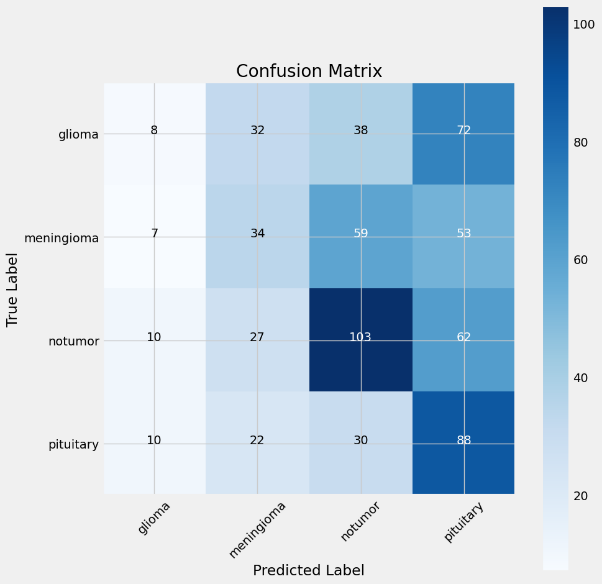


Figure 18 Confusion Matrix of EfficientNet

### 6.6.8 Evaluation Metrics****:****

The performance of the model across the three datasets is presented below by the evaluation results: For example, a table format on how the evaluation metrics will look:

Table 2 Evaluation Metrics of EfficientNet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **Training Set** | 34.63 | 34.7% | 34.6% | 33.6% |
| **Validation Set** | 27.62 | 32.3% | 32.3% | 31.2% |
| **Test Set** | 33.60 | 35.6% | 35.5% | 34.5% |

### 6.6.9 Model Testing Test Image Prediction:

It was tested on a new MRI image to verify the model's effectiveness. The model successfully predicted the class of the image, demonstrating its potential to perform well on unseen data.

The model predicted the class of the test image as a "notumor," validating its ability to generalise to new inputs. The EfficientNet model demonstrated a moderate level of performance in classifying MRI images across four distinct categories.

## 6.7 RasNet50 Model:

The ResNet50 model was employed to classify the MRI images of glioma, meningioma, pituitary tumour, and no tumour.

### 6.7.1 Data Exploration and Distribution

First, data exploration was performed, ensuring a proper balance across the different classes; this is always an essential pre-requisite before any model training. The following visualisations show the distribution of images across the four classes in both the training and test datasets.

### ****6.7.2 Training Dataset Distribution:****

The images in the training dataset are distributed across four classes as follows: It can be seen from the bar plot that no tumour is the class with the highest number of images at 1,595, followed by the pituitary at 1,457, meningioma at 1,339, and glioma at 1,321. The classes are pretty well distributed, which, for training, is a good thing.

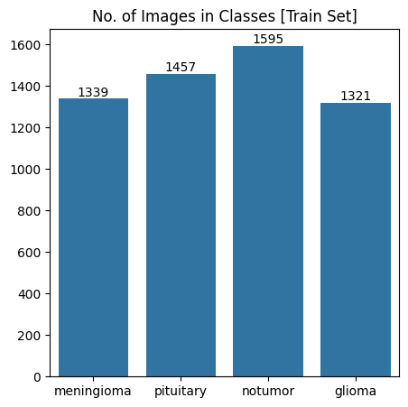


Figure 19 Training Dataset Distribution of ResNet50

### ****6.7.3 Test Dataset Distribution****

The test dataset was also explored to see whether it is representative of the training data: In the test set, the no tumour class is once again most prevalent with 405 images; pituitary and glioma are represented with 300 images each, while meningioma has 306 images. The distribution is fine, though a little imbalanced, for evaluation.

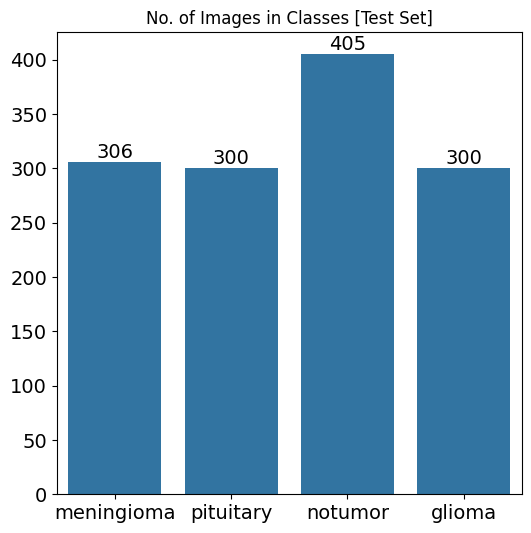


Figure 20 Bar Chart of Test Dataset Distribution

The pie chart below further elaborates on the above distribution of the test dataset. This pie chart shows the relative composition of each class in the test set. From the chart, it is clear that no tumour corresponds to 30.9% of the dataset, meningioma at 23.3%, glioma at 22.9%, and pituitary at 22.9%

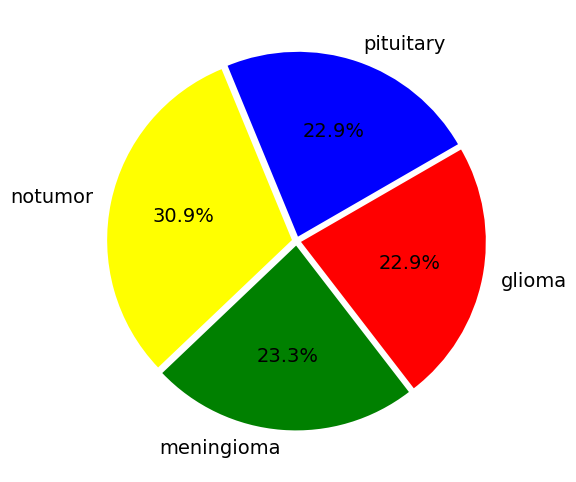


Figure 21 Pie Chart of Test Dataset Distribution

### 6.7.4 Data Splitting and Distribution

The dataset was divided into three parts: training, 81.33%; validation, 9.34%; and testing, 9.33%. The split provides that the model will be trained on enough data, but at the same time, it will have an effective validation and test. The pie showing their distribution is presented below:

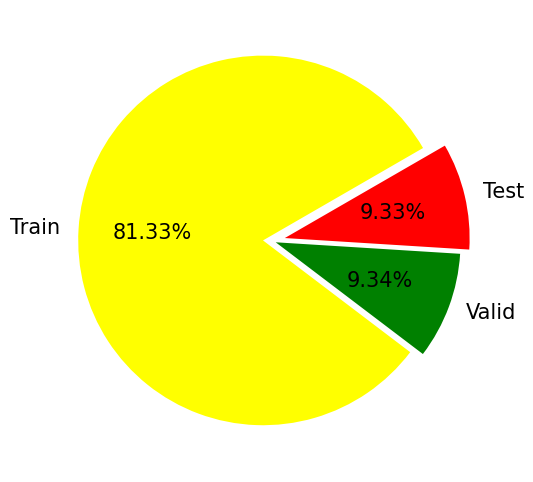


Figure 22 Data Splitting and Distribution

### 6.7.5 Sample Images from the Dataset

To be positive that images were labelled appropriately, a random selection of the images in each class was visualised, and to have an idea of what data the model will be trained on, below is a grid displaying some example representative images from the dataset:

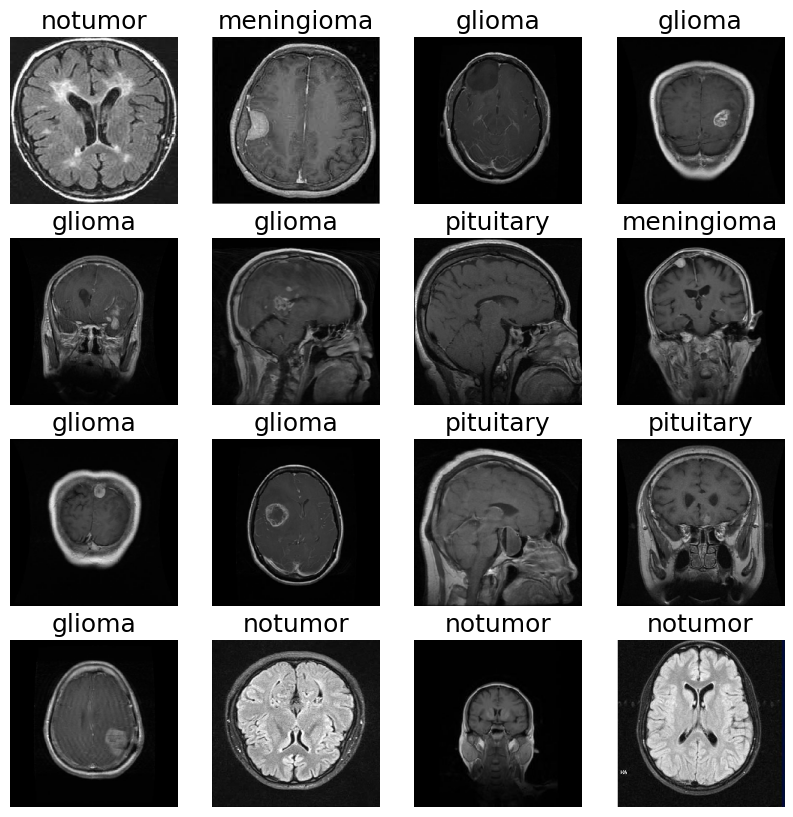


Figure 23 Sample Images from the Dataset

### 6.7.6 Model Architecture

The base model used is ResNet50, pre-trained over the ImageNet dataset because this network extracts deep features from images. Fine-tuning of the model architecture thus consisted of adding one's own custom set of layers atop the ResNet50 base. Details of the architecture follow:

* **Input Shape:** 224x224x3(RGB)
* **Base model:** ResNet50, pre-trained on ImageNet except top layer.
* Custom Layers:
* **Flatten layer:** It makes the 2D matrix data flat into a vector.
* **Dense Layer:** 128 units with ReLU for activation.
* **Dropout Layers**: Applied dropout here at 30% and 25%, respectively, to avoid overfitting.
* **Output Layer:** 4 units. Softmax activation could yield multi-class classification.

### ****6.7.7 Training and Validation Loss & Accuracy****

These graphs represent decreased loss and increased accuracy regarding epochs during training. This model's fit is quite impressive, as it can be seen that the validation accuracy follows quite closely to the training accuracy.

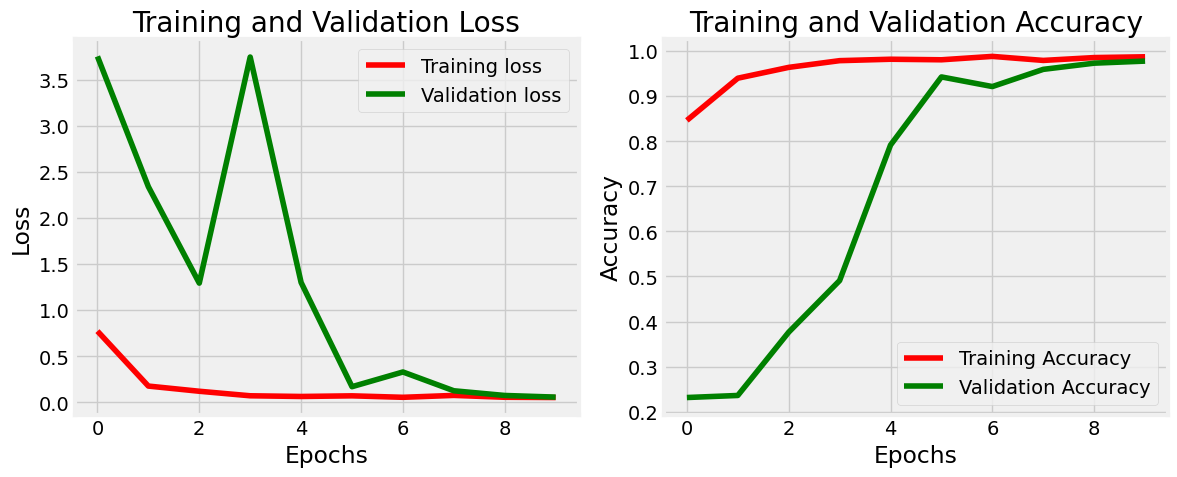


Figure 24 Training and Validation Loss & Accuracy

### ****6.7.8 Training and Validation Precision & Recall****

The precision and recall metrics indicate the model's performance in classifying the images correctly, which has become better after the first few epochs.

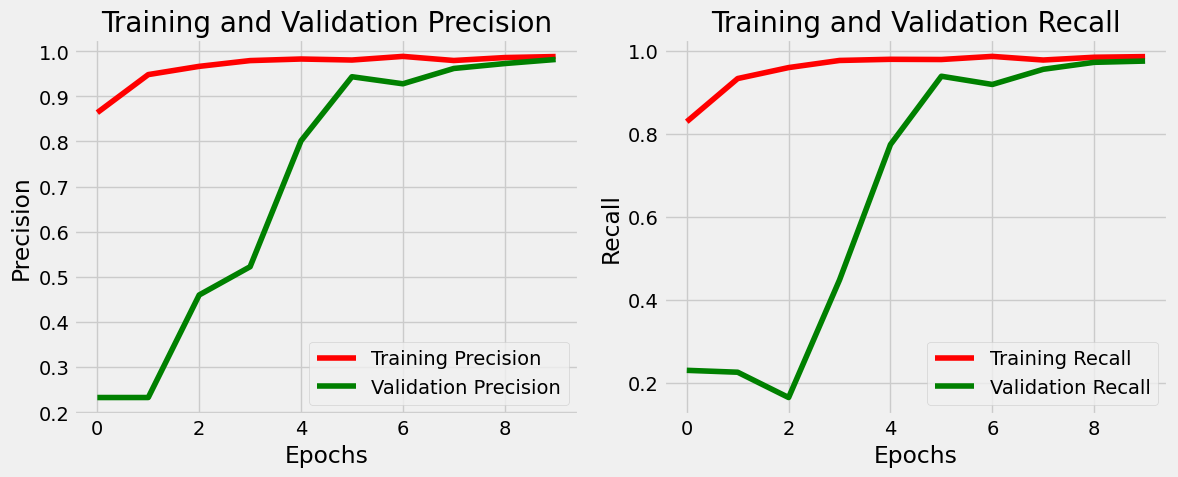


Figure 25 Training and Validation Precision & Recall

### ****6.7.9 Training and Validation AUC****

The AUC, a measure of the model's overall performance, hovers close to 1.0, reflecting the model's very good performance in distinguishing between the classes.

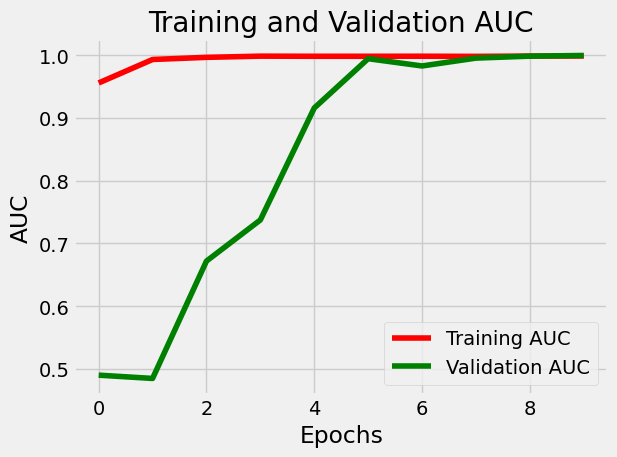


Figure 26 Training and Validation AUC

### 6.7.10 Confusion Matrix

The confusion matrix below is a granular breakdown of the model's performance on the test set. It provides the number of correct and incorrect predictions for each class:

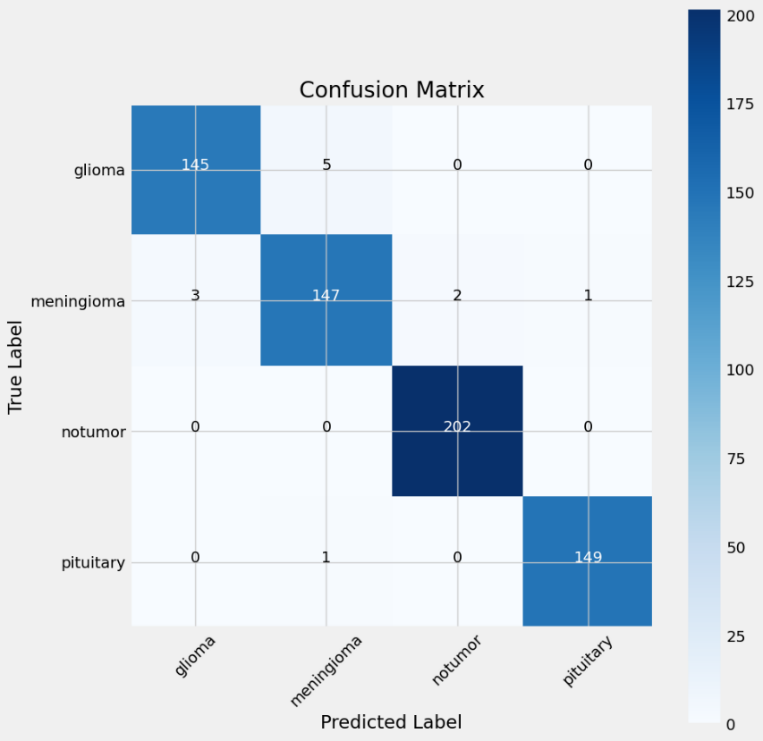


Figure 27 Confusion Matrix of ResNet50

### 6.7.11 Model Performance

A summary of the performance of the training, validation, and test sets combined is shown for the final model in the table below:

Table 3 ResNet50 Model Performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **Training** | 0.012 | 99.7% | 99.8% | 99.7% |
| **Validation** | 0.05 | 97.7% | 98.2% | 97.7% |
| **Testing** | 0.066 | 98.2% | 98.3% | 98.0% |

## 6.8 Comparative Analysis of Models

This section compares the performance of ResNet50, EfficientNet, and Xception in MRI image classification tasks for glioma, meningioma, pituitary tumour, and no tumour. Comparisons are drawn for key performance metrics across training, validation, and test datasets.

### 6.8.1 Training Set Performance

Table 4 Training Set Performance of Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **Xception** | 0.013 | 99.6% | 99.6% | 99.6% |
| **EfficientNet** | 35.00 | 34.7% | 34.6% | 33.6% |
| **ResNet50** | 0.012 | 99.7% | 99.8% | 99.7% |

**Analysis**: The models ResNet50 and Xception perform much better than EfficientNet, with almost perfect accuracy and precision in the training set. In contrast, the latter performs poorly since its accuracy measures are very low and its loss very high.

### 6.8.2 Validation Set Performance

Table 5 Validation Set Performance of Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **ResNet50** | 0.050 | 97.7% | 98.2% | 97.7% |
| **EfficientNet** | 28.00 | 32.3% | 32.3% | 31.2% |
| **Xception** | 0.081 | 98.3% | 98.3% | 98.3% |

**Analysis**: The results show that Xception has slightly outperformed the ResNet 50 on the validation set, thus generalising better, whereas EfficientNet's performances remain mediocre, presenting poor validation accuracy and high loss.

### 6.8.3 Test Set Performance

Table 6 Test Set Performance of Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Loss** | **Accuracy** | **Precision** | **Recall** |
| **ResNet50** | 0.066 | 98.2% | 98.3% | 98.0% |
| **EfficientNet** | 34.00 | 35.6% | 35.5% | 34.5% |
| **Xception** | 0.110 | 97.7% | 97.9% | 97.7% |

**Analysis**: Xception and ResNet50 show steady performances in the test set, and EfficientNet lags well behind. Ultimately, Xception is the best architecture for MRI image classification, closely followed by ResNet50. EfficientNet came in third after them.

## ****6.9 Web App Overview****

The web application automatically uses the best model to classify MRI images with better efficiency among the three models, ResNet50, EfficientNet, and Xception. It has been chosen based on comprehensive performance metrics including accuracy, precision, recall, and loss.

* **Uploading of Images:** The web application will allow users to upload their MRI images for tumour detection.
* **Tumor Classify Automatically:** This application will process the image fully automatically using the most efficient model and will return classification into classes like glioma, meningioma, pituitary tumour, or no tumour.
* **Visualization of Results:** The predicted class is returned along with the confidence score; therefore, the probability of the diagnosis is clear.
* **User-Friendly Interface:** Design the application with a straightforward and intuitive interface; hence, uploading images and viewing results would be fast and efficient.

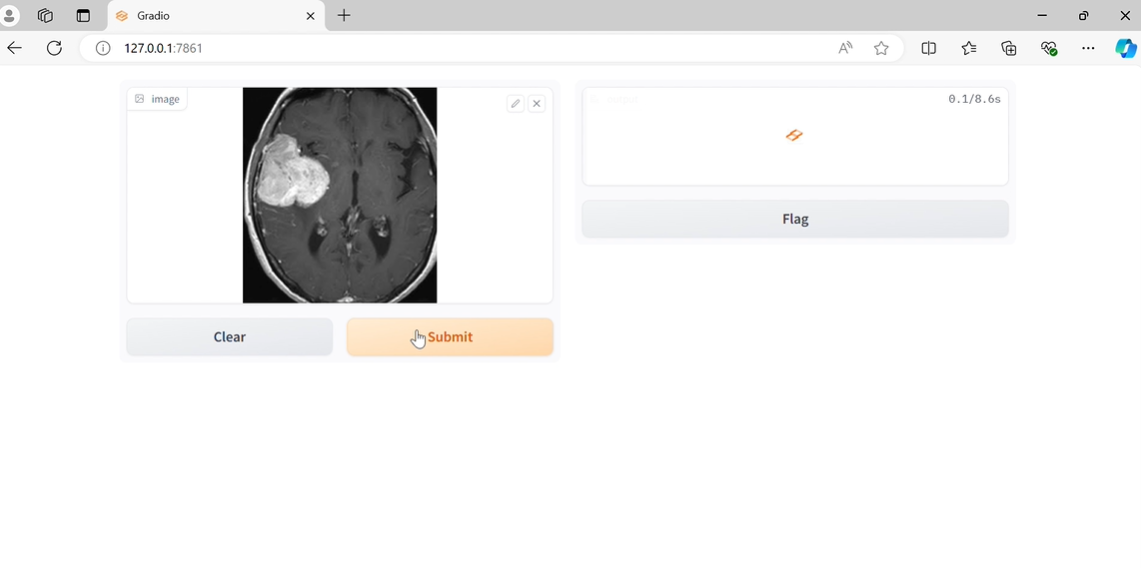


Figure 28 Select Image



Figure 29 Detection of Tumour

This web application will be an efficient tool for researchers and medical professionals. It provides the speedy and surest classification of tumours right directly from MRI images

## 6.9 Addressing Research Questions

The findings from employing multiple metrics and their comprehensive presentation help respond to your research questions as follows:

* **Does ML/DL improve tumour classification?** Yes, as seen from high precision, recall, and accuracy across models.
* **What pre-processing methods work?** This information can be inferred from comparing models trained with different pre-processing techniques.
* **Which architectures work best?** A direct comparison of metrics across models highlights the most influential architecture.
* **What do these metrics tell us?** They convey profound knowledge about the strengths and weaknesses of the model to fine-tune it further and use it in the field.

Perform a systematic assessment for each model using these metrics and present their findings. The research addresses critical questions and showcases how it could be viable for real-world deployment in healthcare.

# Chapter 7: Analysis and Discussion

## 7.1 Overview:

This chapter interprets the results after applying ResNet50, EfficientNet, and Xception models for MRI brain tumour classification. These models have indicated their ergonomic features in classifying MRI images as glioma, meningioma, pituitary tumour, and no tumour. The outcome underlies the effectiveness of the models; therefore, it is viewed that Xception stands out to date as the best model, which has yielded an accuracy of 98.3% on validation, while precision and recall are strongly evidenced for all the classes.

## ****7.2 Model Comparison:****

* **Xception:** With its deep residual network, Xception outperformed the other models in terms of accuracy and precision; thus, it is considered the most reliable for MRI classifications. Since it contains its architecture, it avoids vanishing gradient problems, which results in better feature extraction and classification performance.
* **ResNet50:** Was also good but lagged behind Xception, especially regarding classification between meningioma and pituitary. Nevertheless, it is still a perfect model for the task.
* **EfficientNet:** It performed worse than the others, probably due to its focus on reducing computation rather than feature complexity, without fitting well into a specific task like MRI classification.

## ****7.3 Comparison with Literature:****

These findings agree with established literature, as Xception does well in medical image classifications because of its complex architecture. EfficientNet performed reasonably lower in this study, consistent with other findings that it performs exceptionally well on less complex image recognition tasks.

## 7.4 Limitations:

The fundamental limitations are the representativeness of the dataset on which the models were trained, and their generalisation to other datasets or MRI machines can be questioned. Overfitting remains a possibility, especially in a high-parameter model such as ResNet50. Also, while the metrics are very comprehensive, they might not fully capture the clinical implications, especially around false negatives and positives.

## ****7.5 Relation to Research Objectives:****

The paper also successfully addressed the project's objectives, showing that deep learning models, especially Xception, have considerably improved MRI brain tumour detection and classification. This research identified several effective pre-processing methods and emphasised the most appropriate model architecture for this task.

## ****7.6 Practical Application:****

The Xception model is appropriate for clinical integration, as it has high accuracy, precision, and recall. Further validation shall be required for generalisation across other datasets. Performance indicates excellent potential in real-world healthcare applications.

Specific research questions were well addressed, and significant contributions of deep learning were identified in MRI brain tumour classification.

# Chapter 8: Conclusion

The research has opened the potentiality of deep learning models, especially Xception, in enhancing the detection and classification of brain tumours from MRI images. Key findings include:

* **Model Performance:** Xception outperformed the other two models constantly in ResNet50 and EfficientNet regarding the highest accuracy, recall, and precision of all three training, validation, and testing data sets. Meanwhile, ResNet50 worked well, while EfficientNet had less resource consumption than the other two but did not compete as a winner on this particular task as expected.
* **Data Preprocessing and Augmentation: A** few good preprocessing techniques that OKed the model for generalisation and variability in MRI images involve proper data augmentation and splitting of datasets.
* **Real-World Utility:** Xception contributions have huge potential, considering its accuracy and robustness of classification, for real-world applications in the healthcare domain concerning diagnosis. It can be integrated into the clinical workflow so that a radiologist can help diagnose brain tumours with fewer chances of diagnostic errors and thus improve patient outcomes.

Although the performance results look promising, further improvements can be made. This work may be extended. Most importantly, it can be done by increasing the dataset with more diverse MRI images obtained from different machines and institutions to validate the model's generalisation capability more widely. This study has answered the research questions and set the pace for the continuous development of AI-driven diagnostic tools in medical imaging. The results reveal that deep learning models such as Xception have shown potential to be combined into future medical diagnosis practices to reliably and efficiently support healthcare workers.

# References

Ahmad, R., Ashraf, M., and Alzubi, J. A. (2022) 'Interpretability in deep learning models: A survey on methods and metrics', IEEE Access, 10, pp. 85015-85032. [doi: 10.1109/ACCESS.2022.3185985.](doi:%2010.1109/ACCESS.2022.3185985)

Anwar, S. M., Majid, M., Qayyum, A., Awais, M., Alnowami, M., and Khan, M. K. (2021) 'Medical image analysis using convolutional neural networks: A review', Journal of Medical Systems, 45(2), p. 44. doi: <10.1007/s10916-020-01659-7>.

Chakrabarty, A., Das, P., and Naskar, S. (2022) 'Deep learning models for brain tumour detection: A comprehensive review', Biomedical Signal Processing and Control, 68, p. 102805. doi: <10.1016/j.bspc.2021.102805>.

Chollet, F. (2017) 'Xception: Deep learning with depthwise separable convolutions', in Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1251-1258. doi: <10.1109/CVPR.2017.195>.

European Parliament (2018) General Data Protection Regulation (GDPR). Official Journal of the European Union. Available at: [https://eur-lex.europa.eu/eli/reg/2016/679/oj (A](:%20https:/eur-lex.europa.eu/eli/reg/2016/679/oj)ccessed: 23 August 2024).

He, K., Zhang, X., Ren, S., and Sun, J. (2016) 'Deep residual learning for image recognition', in Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770-778. doi: <10.1109/CVPR.2016.90>.

Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., Wang, Y., Dong, Q., Shen, H., and Wang, Y. (2017) 'Artificial intelligence in healthcare: Past, present, and future', Stroke and Vascular Neurology, 2(4), pp. 230-243. doi: <10.1136/svn-2017-000101>.

Kaur, G. *et al.* (2024) 'Brain Tumour Disease Detection in MRI Images using Xception Model,' *In 2024 3rd International Conference for Innovation in Technology (INOCON)*, pp. 1–5. <https://doi.org/10.1109/inocon60754.2024.10511504>.

Li, Z., Wang, Y., Wu, Z., Xiang, Y., Liu, Y., and Zhou, H. (2022) '3D convolutional neural networks for brain tumor segmentation: A comprehensive study', Medical Image Analysis, 76, p. 102300. doi: <10.1016/j.media.2021.102300>.

Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., ... and Prastawa, M. (2015) 'The multimodal brain tumor image segmentation benchmark (BRATS)', IEEE transactions on medical imaging, 34(10), pp. 1993-2024. doi: <10.1109/TMI.2014.2377694>.

Mohsen, H., El-Dahshan, E. S., El-Horbaty, E. S., and Salem, A. B. (2020) 'Classification using deep learning neural networks for brain tumors', Future Computing and Informatics Journal, 3(1), pp. 68-71. doi: <10.1016/j.fcij.2017.12.001>.

Obermeyer, Z., Powers, B., Vogeli, C., and Mullainathan, S. (2019) 'Dissecting racial bias in an algorithm used to manage the health of populations', Science, 366(6464), pp. 447-453. doi: <10.1126/science.aax2342>.

Ronneberger, O., Fischer, P., and Brox, T. (2015) 'U-Net: Convolutional networks for biomedical image segmentation', in International Conference on Medical image computing and computer-assisted intervention. pp. 234-241. doi: <10.1007/978-3-319-24574-4_28>.

Shen, D., Wu, G., and Suk, H. I. (2022) 'Deep learning in medical image analysis', Annual Review of Biomedical Engineering, 19, pp. 221-248. doi: <10.1146/annurev-bioeng-071516-044442>.

Mathivanan, S.K. *et al.* (2024) 'Employing deep learning and transfer learning for accurate brain tumor detection,' *Scientific Reports*, 14(1). <https://doi.org/10.1038/s41598-024-57970-7>.

Tan, M., and Le, Q. V. (2019) 'EfficientNet: Rethinking model scaling for convolutional neural networks', in Proceedings of the 36th International Conference on Machine Learning. pp. 6105-6114. doi: <10.48550/arXiv.1905.11946>.

Zhang, Y., Wang, X., Shi, Z., Wang, L., and Lin, X. (2023) 'A comparative study of deep learning models for brain tumor segmentation', IEEE Access, 11, pp. 25478-25489. doi: <10.1109/ACCESS.2023.3248753>.

# Appendix

**Xception Model Code:**

!cp /content/drive/MyDrive/MRI/archive.zip /content/

!unzip archive.zip

import os

import numpy as np

import pandas as pd

import seaborn as sns

from glob import glob

from PIL import Image

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten

from tensorflow.keras.optimizers import Adamax

from tensorflow.keras.metrics import Precision, Recall, AUC

from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

from sklearn.metrics import classification\_report, confusion\_matrix

# paths to data directories

train\_dir = "Training/"

test\_dir = "Testing/"

def get\_dataset\_dataframe(directory\_path):

    """

    This function reads a dataset directory and creates a Pandas DataFrame

    containing image paths and their corresponding labels.

    Args:

        directory\_path (str): The path to the dataset directory.

    Returns:

        pd.DataFrame: A DataFrame with two columns:

            - 'Path': Full path to each image file.

            - 'Label': Class label corresponding to the image directory.

    """

    # Extract class labels and image paths within the directory

    classes = []

    class\_paths = []

    for label in os.listdir(directory\_path):

        # Check if the entry is a directory (representing a class)

        if os.path.isdir(os.path.join(directory\_path, label)):

            # Loop through each image file within the class directory

            for image in os.listdir(os.path.join(directory\_path, label)):

                # Construct the full image path

                image\_path = os.path.join(directory\_path, label, image)

                classes.append(label)  # Append the class label

                class\_paths.append(image\_path)  # Append the image path

    # Create a Pandas DataFrame from the extracted data

    df = pd.DataFrame({'Path': class\_paths, 'Label': classes})

    return df

# read training dataframe

train\_df = get\_dataset\_dataframe(train\_dir)

# show 10 random samples

train\_df.sample(10)

# show dataframe info

train\_df.info()

train\_df.describe().T

# plot no of images in each class

# Set the figure size to 5x5 inches

plt.figure(figsize=(5,5))

# Create a count plot of class labels

ax = sns.countplot(data=train\_df , x=train\_df['Label'])

# Remove the x-axis label

plt.xlabel('')

# Remove the y-axis label

plt.ylabel('')

# set title and its size

plt.title('No. of Images in Classes [Train Set]', fontsize=12)

# Add count labels to the top of each bar

ax.bar\_label(ax.containers[0])

# show plot

plt.show()

plt.figure(figsize=(14, 6))  # Create a figure with width 14 and height 6

# Define a list of colors for the pie chart slices

colors = ['blue', 'yellow', 'green', 'red']

# Set the default font size for the plot

plt.rcParams.update({'font.size': 14})

# Count occurrences of each class label in the 'Label' column of train\_df

class\_counts = {

    'pituitary': len([x for x in train\_df['Label'] if x == 'pituitary']),

    'notumor': len([x for x in train\_df['Label'] if x == 'notumor']),

    'meningioma': len([x for x in train\_df['Label'] if x == 'meningioma']),

    'glioma': len([x for x in train\_df['Label'] if x == 'glioma']),

}

# Create a pie chart

plt.pie(list(class\_counts.values()),  # Use the list of class counts

        labels=list(class\_counts.keys()),  # Use the list of class labels as labels

        colors=colors,                   # Set the slice colors

        autopct='%.1f%%',                # Display percentage with one decimal place

        explode=(0.025, 0.025, 0.025, 0.025),  # Slightly explode slices for better visibility

        startangle=30                   # Start the pie at 30 degrees

)

plt.show()  # Display the pie chart

# read test dataframe

test\_df = get\_dataset\_dataframe(test\_dir)

# show 10 random samples

test\_df.sample(10)

plt.figure(figsize=(6, 6))  # Create a figure with width 6 and height 6

# Create a count plot of class labels in the test set using Seaborn

ax = sns.countplot(data=test\_df, x=test\_df['Label'])

# Remove default labels

plt.xlabel('')  # Remove the x-axis label

plt.ylabel('')  # Remove the y-axis label

# Set plot title

plt.title('No. of Images in Classes [Test Set]', fontsize=12)  # Set the plot title with a font size of 12

# Add count labels on top of bars

ax.bar\_label(ax.containers[0])  # Add count labels to the top of each bar

plt.show()  # Display the plot

plt.figure(figsize=(14, 6))  # Create a figure with width 14 and height 6

# Define a list of colors for the pie chart slices

colors = ['blue', 'yellow', 'green', 'red']

# Set the default font size for the plot

plt.rcParams.update({'font.size': 14})

# Count occurrences of each class label in the 'Label' column of test\_df

class\_counts = {

    'pituitary': len([x for x in test\_df['Label'] if x == 'pituitary']),

    'notumor': len([x for x in test\_df['Label'] if x == 'notumor']),

    'meningioma': len([x for x in test\_df['Label'] if x == 'meningioma']),

    'glioma': len([x for x in test\_df['Label'] if x == 'glioma']),

}

# Create a pie chart

plt.pie(list(class\_counts.values()),  # Use the list of class counts

        labels=list(class\_counts.keys()),  # Use the list of class labels as labels

        colors=colors,                   # Set the slice colors

        autopct='%.1f%%',                # Display percentage with one decimal place

        explode=(0.025, 0.025, 0.025, 0.025),  # Slightly explode slices for better visibility

        startangle=30                   # Start the pie at 30 degrees

)

plt.show()  # Display the pie chart

# Split Testing Dataset into Validation by 50%

test\_df, valid\_df = train\_test\_split(test\_df, train\_size=0.5, random\_state=10, stratify=test\_df['Label'])

print(f"Length of Training Dataset: {len(train\_df)}")

print(f"Length of Validation Dataset: {len(valid\_df)}")

print(f"Length of Testing Dataset: {len(test\_df)}")

plt.figure(figsize=(14, 6))  # Create a figure with width 14 and height 6

# Define a list of colors for the pie chart slices

colors = ['yellow', 'green', 'red']

# Set the default font size for the plot

plt.rcParams.update({'font.size': 15})

# Get the lengths (number of rows) of the DataFrames

train\_df\_len = len(train\_df)

valid\_df\_len = len(valid\_df)

test\_df\_len = len(test\_df)

# Create a pie chart to visualize dataset distribution

plt.pie([train\_df\_len, valid\_df\_len, test\_df\_len],  # Use lengths for pie slice sizes

        labels=['Train', 'Valid', 'Test'],          # Set slice labels

        colors=colors,                               # Set slice colors

        autopct='%.2f%%',                          # Display percentage with two decimal places

        explode=(0.05, 0.02, 0.07),                 # Separate slices for better visibility

        startangle=30                               # Start the pie at 30 degrees

)

plt.show()  # Display the pie chart

# Define batch size for data loading (common practice is a power of 2)

batch\_size = 32

# Set image size for resizing during data preparation

img\_size = (299, 299)

# Define data augmentation for training data

train\_gen = ImageDataGenerator(rescale=1/255,  # Rescale pixel values to [0, 1]

                               brightness\_range=(0.8, 1.2))  # Randomly adjust brightness

# Define data preparation for test data (no augmentation)

test\_gen = ImageDataGenerator(rescale=1/255)  # Rescale pixel values to [0, 1]

# Create training data generator from the training DataFrame

train\_generator = train\_gen.flow\_from\_dataframe(train\_df,

                                                x\_col='Path',  # Column containing image paths

                                                y\_col='Label',  # Column containing class labels

                                                target\_size=img\_size,  # Resize images to specified size

                                                batch\_size=batch\_size,  # Load data in batches of this size

                                                shuffle=True)  # Shuffle data for training

# Create validation data generator (similar to training, but no shuffling)

valid\_generator = train\_gen.flow\_from\_dataframe(valid\_df,

                                                x\_col='Path',

                                                y\_col='Label',

                                                target\_size=img\_size,

                                                batch\_size=batch\_size,

                                                shuffle=False)

# Create test data generator (no augmentation, smaller batch size, no shuffling)

test\_generator = test\_gen.flow\_from\_dataframe(test\_df,

                                               x\_col='Path',

                                               y\_col='Label',

                                               target\_size=img\_size,

                                               batch\_size=16,  # Smaller batch size for testing

                                               shuffle=False)  # Don't shuffle test data

# Get class indices from training data generator

label\_mappings = train\_generator.class\_indices

print(label\_mappings)  # Print the dictionary mapping class labels to numerical indices

# Create a list of class names from the keys of the label mappings dictionary

label\_names = list(label\_mappings.keys())

# Get a batch of images and labels from the test generator

images, labels = next(test\_generator)

# Create a figure for visualization

plt.figure(figsize=(10, 10))  # Set figure size to 10x10 inches

# Loop through each image and label in the batch

for idx, (image, label) in enumerate(zip(images, labels)):

    # Create a subplot at a specific position in the figure grid

    plt.subplot(4, 4, idx + 1)  # 4 rows, 4 columns, starting from index 1 (top-left)

    # Display the image using imshow

    plt.imshow(image)

    # Get the class name based on the predicted label index

    label\_name = label\_names[np.argmax(label)]  # Find the index with highest value in the label

    # Set the subplot title to the class name

    plt.title(label\_name)

    # Turn off axis labels and ticks for cleaner visualization

    plt.axis('off')

# Display the resulting image grid

plt.show()

# Define image input shape (height, width, color channels)

img\_shape = (299, 299, 3)  # 299x299 pixels with 3 color channels (RGB)

# Load a pre-trained Xception model

base\_model = tf.keras.applications.Xception(

    include\_top=False,  # Exclude the top (classification) layers

    weights="imagenet",  # Load pre-trained weights on ImageNet dataset

    input\_shape=img\_shape,  # Specify the input shape

    pooling='max'       # Use max pooling for feature extraction

)

# Build the final classifier model

model = Sequential([

    base\_model,  # Use the pre-trained Xception model for feature extraction

    Flatten(),    # Flatten the extracted features from the base model

    Dropout(rate=0.3),  # Apply dropout with 30% probability to prevent overfitting

    Dense(128, activation='relu'),  # Dense layer with 128 neurons and ReLU activation

    Dropout(rate=0.25),  # Apply dropout with 25% probability to prevent overfitting

    Dense(4, activation='softmax')  # Output layer with 4 neurons and softmax activation for multi-class classification

])

# Compile the model

model.compile(optimizer=Adamax(learning\_rate=0.001),  # Use Adamax optimizer with learning rate 0.001

              loss='categorical\_crossentropy',  # Categorical crossentropy loss for multi-class classification

              metrics=['accuracy',  # Track accuracy during training

                       Precision(),  # Track precision metric

                       Recall(), # Track recall metric

                       AUC()])  # Track AUC metric

# Print a summary of the model architecture

model.summary()

# start model training for 10 epochs

history = model.fit(train\_generator,

                    epochs=10,

                    validation\_data=valid\_generator,

                    shuffle= False)

# Extract training and validation metrics from history object

tr\_acc = history.history['accuracy']

tr\_loss = history.history['loss']

val\_acc = history.history['val\_accuracy']

val\_loss = history.history['val\_loss']

# Create a figure for plotting (size: 12x6 inches)

plt.figure(figsize=(12, 5))

# Set plot style for aesthetics

plt.style.use('fivethirtyeight')

# Create a subplot for loss visualization (occupying left half)

plt.subplot(1, 2, 1)

# Plot training loss (red line, labeled)

plt.plot(tr\_loss, 'r', label='Training loss')

# Plot validation loss (green line, labeled)

plt.plot(val\_loss, 'g', label='Validation loss')

# Add labels and title for loss plot

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

# Add legend to identify lines in the loss plot

plt.legend()

# Create a subplot for accuracy visualization (occupying right half)

plt.subplot(1, 2, 2)

# Plot training accuracy (red line, labeled)

plt.plot(tr\_acc, 'r', label='Training Accuracy')

# Plot validation accuracy (green line, labeled)

plt.plot(val\_acc, 'g', label='Validation Accuracy')

# Add labels and title for accuracy plot

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

# Add legend to identify lines in the accuracy plot

plt.legend()

# Adjust layout to prevent overlapping elements

plt.tight\_layout()

# Display the final visualization

plt.show()

**EfficientNet**

plt.figure(figsize=(14, 6))  # Create a figure with width 14 and height 6

# Define a list of colors for the pie chart slices

colors = ['yellow', 'green', 'red']

# Set the default font size for the plot

plt.rcParams.update({'font.size': 15})

# Get the lengths (number of rows) of the DataFrames

train\_df\_len = len(train\_df)

valid\_df\_len = len(valid\_df)

test\_df\_len = len(test\_df)

# Create a pie chart to visualize dataset distribution

plt.pie([train\_df\_len, valid\_df\_len, test\_df\_len],  # Use lengths for pie slice sizes

        labels=['Train', 'Valid', 'Test'],          # Set slice labels

        colors=colors,                               # Set slice colors

        autopct='%.2f%%',                          # Display percentage with two decimal places

        explode=(0.05, 0.02, 0.07),                 # Separate slices for better visibility

        startangle=30                               # Start the pie at 30 degrees

)

plt.show()  # Display the pie chart

# Define batch size for data loading (common practice is a power of 2)

batch\_size = 32

# Set image size for resizing during data preparation

img\_size = (300, 300)

# Define data augmentation for training data

train\_gen = ImageDataGenerator(rescale=1/255,  # Rescale pixel values to [0, 1]

                               brightness\_range=(0.8, 1.2))  # Randomly adjust brightness

# Define data preparation for test data (no augmentation)

test\_gen = ImageDataGenerator(rescale=1/255)  # Rescale pixel values to [0, 1]

# Create training data generator from the training DataFrame

train\_generator = train\_gen.flow\_from\_dataframe(train\_df,

                                                x\_col='Path',  # Column containing image paths

                                                y\_col='Label',  # Column containing class labels

                                                target\_size=img\_size,  # Resize images to specified size

                                                batch\_size=batch\_size,  # Load data in batches of this size

                                                shuffle=True)  # Shuffle data for training

# Create validation data generator (similar to training, but no shuffling)

valid\_generator = train\_gen.flow\_from\_dataframe(valid\_df,

                                                x\_col='Path',

                                                y\_col='Label',

                                                target\_size=img\_size,

                                                batch\_size=batch\_size,

                                                shuffle=False)

# Create test data generator (no augmentation, smaller batch size, no shuffling)

test\_generator = test\_gen.flow\_from\_dataframe(test\_df,

                                               x\_col='Path',

                                               y\_col='Label',

                                               target\_size=img\_size,

                                               batch\_size=16,  # Smaller batch size for testing

                                               shuffle=False)  # Don't shuffle test data

# Get class indices from training data generator

label\_mappings = train\_generator.class\_indices

print(label\_mappings)  # Print the dictionary mapping class labels to numerical indices

# Create a list of class names from the keys of the label mappings dictionary

label\_names = list(label\_mappings.keys())

# Get a batch of images and labels from the test generator

images, labels = next(test\_generator)

# Create a figure for visualization

plt.figure(figsize=(10, 10))  # Set figure size to 10x10 inches

# Loop through each image and label in the batch

for idx, (image, label) in enumerate(zip(images, labels)):

    # Create a subplot at a specific position in the figure grid

    plt.subplot(4, 4, idx + 1)  # 4 rows, 4 columns, starting from index 1 (top-left)

    # Display the image using imshow

    plt.imshow(image)

    # Get the class name based on the predicted label index

    label\_name = label\_names[np.argmax(label)]  # Find the index with highest value in the label

    # Set the subplot title to the class name

    plt.title(label\_name)

    # Turn off axis labels and ticks for cleaner visualization

    plt.axis('off')

# Display the resulting image grid

plt.show()

# Define image input shape (height, width, color channels)

img\_shape = (300, 300, 3)  # pixels with 3 color channels (RGB)

# Load a pre-trained Xception model

base\_model = tf.keras.applications.EfficientNetB3(

    include\_top=False,  # Exclude the top (classification) layers

    weights="imagenet",  # Load pre-trained weights on ImageNet dataset

    input\_shape=img\_shape,  # Specify the input shape

    pooling='max'       # Use max pooling for feature extraction

)

# Build the final classifier model

model = Sequential([

    base\_model,  # Use the pre-trained Xception model for feature extraction

    Flatten(),    # Flatten the extracted features from the base model

    Dropout(rate=0.3),  # Apply dropout with 30% probability to prevent overfitting

    Dense(128, activation='relu'),  # Dense layer with 128 neurons and ReLU activation

    Dropout(rate=0.25),  # Apply dropout with 25% probability to prevent overfitting

    Dense(4, activation='softmax')  # Output layer with 4 neurons and softmax activation for multi-class classification

])

# Compile the model

model.compile(optimizer=Adamax(learning\_rate=0.0005),  # Use Adamax optimizer

              loss='categorical\_crossentropy',  # Categorical crossentropy loss for multi-class classification

              metrics=['accuracy',  # Track accuracy during training

                       Precision(),  # Track precision metric

                       Recall(), # Track recall metric

                       AUC()])  # Track AUC metric

# Print a summary of the model architecture

model.summary()

# start model training for 10 epochs

history = model.fit(train\_generator,

                    epochs=10,

                    validation\_data=valid\_generator,

                    shuffle= False)

# Extract training and validation metrics from history object

tr\_acc = history.history['accuracy']

tr\_loss = history.history['loss']

val\_acc = history.history['val\_accuracy']

val\_loss = history.history['val\_loss']

# Create a figure for plotting (size: 12x6 inches)

plt.figure(figsize=(12, 5))

# Set plot style for aesthetics

plt.style.use('fivethirtyeight')

# Create a subplot for loss visualization (occupying left half)

plt.subplot(1, 2, 1)

# Plot training loss (red line, labeled)

plt.plot(tr\_loss, 'r', label='Training loss')

# Plot validation loss (green line, labeled)

plt.plot(val\_loss, 'g', label='Validation loss')

# Add labels and title for loss plot

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

# Add legend to identify lines in the loss plot

plt.legend()

# Create a subplot for accuracy visualization (occupying right half)

plt.subplot(1, 2, 2)

# Plot training accuracy (red line, labeled)

plt.plot(tr\_acc, 'r', label='Training Accuracy')

# Plot validation accuracy (green line, labeled)

plt.plot(val\_acc, 'g', label='Validation Accuracy')

# Add labels and title for accuracy plot

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

# Add legend to identify lines in the accuracy plot

plt.legend()

# Adjust layout to prevent overlapping elements

plt.tight\_layout()

# Display the final visualization

plt.show()

**ResNet50**

#Model Traning

# Define image input shape (height, width, color channels)

img\_shape = (224, 224, 3)  # 299x299 pixels with 3 color channels (RGB)

# Load a pre-trained Xception model

base\_model = tf.keras.applications.ResNet50(

    include\_top=False,  # Exclude the top (classification) layers

    weights="imagenet",  # Load pre-trained weights on ImageNet dataset

    input\_shape=img\_shape,  # Specify the input shape

    pooling='max'       # Use max pooling for feature extraction

)

# Build the final classifier model

model = Sequential([

    base\_model,  # Use the pre-trained Xception model for feature extraction

    Flatten(),    # Flatten the extracted features from the base model

    Dropout(rate=0.3),  # Apply dropout with 30% probability to prevent overfitting

    Dense(128, activation='relu'),  # Dense layer with 128 neurons and ReLU activation

    Dropout(rate=0.25),  # Apply dropout with 25% probability to prevent overfitting

    Dense(4, activation='softmax')  # Output layer with 4 neurons and softmax activation for multi-class classification

])

# Compile the model

model.compile(optimizer=Adamax(learning\_rate=0.001),  # Use Adamax optimizer with learning rate 0.001

              loss='categorical\_crossentropy',  # Categorical crossentropy loss for multi-class classification

              metrics=['accuracy',  # Track accuracy during training

                       Precision(),  # Track precision metric

                       Recall(), # Track recall metric

                       AUC()])  # Track AUC metric

# Print a summary of the model architecture

model.summary()

# start model training for 10 epochs

history = model.fit(train\_generator,

                    epochs=10,

                    validation\_data=valid\_generator,

                    shuffle= False)

# Extract training and validation metrics from history object

tr\_acc = history.history['accuracy']

tr\_loss = history.history['loss']

val\_acc = history.history['val\_accuracy']

val\_loss = history.history['val\_loss']

# Create a figure for plotting (size: 12x6 inches)

plt.figure(figsize=(12, 5))

# Set plot style for aesthetics

plt.style.use('fivethirtyeight')

# Create a subplot for loss visualization (occupying left half)

plt.subplot(1, 2, 1)

# Plot training loss (red line, labeled)

plt.plot(tr\_loss, 'r', label='Training loss')

# Plot validation loss (green line, labeled)

plt.plot(val\_loss, 'g', label='Validation loss')

# Add labels and title for loss plot

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

# Add legend to identify lines in the loss plot

plt.legend()

# Create a subplot for accuracy visualization (occupying right half)

plt.subplot(1, 2, 2)

# Plot training accuracy (red line, labeled)

plt.plot(tr\_acc, 'r', label='Training Accuracy')

# Plot validation accuracy (green line, labeled)

plt.plot(val\_acc, 'g', label='Validation Accuracy')

# Add labels and title for accuracy plot

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

# Add legend to identify lines in the accuracy plot

plt.legend()

# Adjust layout to prevent overlapping elements

plt.tight\_layout()

# Display the final visualization

plt.show()

# Extract training and validation precision/recall metrics from history object

tr\_precision = history.history['precision']

tr\_recall = history.history['recall']

val\_precision = history.history['val\_precision']

val\_recall = history.history['val\_recall']

# Create a figure for plotting (size: 20x12 inches)

plt.figure(figsize=(12, 5))

# Set plot style for aesthetics

plt.style.use('fivethirtyeight')

# Create a subplot for precision visualization (occupying left half)

plt.subplot(1, 2, 1)

# Plot training precision (red line, labeled)

plt.plot(tr\_precision, 'r', label='Training Precision')

# Plot validation precision (green line, labeled)

plt.plot(val\_precision, 'g', label='Validation Precision')

# Add labels and title for precision plot

plt.title('Training and Validation Precision')

plt.xlabel('Epochs')

plt.ylabel('Precision')

# Add legend to identify lines in the precision plot

plt.legend()

# Create a subplot for recall visualization (occupying right half)

plt.subplot(1, 2, 2)

# Plot training recall (red line, labeled)

plt.plot(tr\_recall, 'r', label='Training Recall')

# Plot validation recall (green line, labeled)

plt.plot(val\_recall, 'g', label='Validation Recall')

# Add labels and title for recall plot

plt.title('Training and Validation Recall')

plt.xlabel('Epochs')

plt.ylabel('Recall')

# Add legend to identify lines in the recall plot

plt.legend()

# Adjust layout to prevent overlapping elements

plt.tight\_layout()

# Display the final visualization

plt.show()

# Extract training and validation AUC metrics from history object

tr\_auc = history.history['auc']

val\_auc = history.history['val\_auc']

# Set plot style for aesthetics

plt.style.use('fivethirtyeight')

# Plot training AUC (red line, labeled)

plt.plot(tr\_auc, 'r', label='Training AUC')

# Plot validation AUC (green line, labeled)

plt.plot(val\_auc, 'g', label='Validation AUC')

# Add labels and title for AUC plot

plt.title('Training and Validation AUC')

plt.xlabel('Epochs')

plt.ylabel('AUC')

# Add legend to identify lines in the AUC plot

plt.legend()

# Adjust layout to prevent overlapping elements

plt.tight\_layout()

# Display the final visualization

plt.show()

# Evaluate model performance on training, validation, and test sets

# Evaluate the model on the training data

train\_score = model.evaluate(train\_generator, verbose=1)

# Evaluate the model on the validation data

valid\_score = model.evaluate(valid\_generator, verbose=1)

# Evaluate the model on the test data (unseen data for final assessment)

test\_score = model.evaluate(test\_generator, verbose=1)

# Make predictions on the test data

preds = model.predict(test\_generator)

# Convert predicted class probabilities to most likely class labels

y\_pred = np.argmax(preds, axis=1)  # Extract the index of the highest probability class for each sample

# Get the class labels (names) from the test data generator's class indices dictionary

g\_dict = test\_generator.class\_indices

classes = list(g\_dict.keys())

# Calculate the confusion matrix

cm = confusion\_matrix(test\_generator.classes, y\_pred)

# Display the confusion matrix (optional: visualize using heatmaps or other plotting libraries)

print("\nConfusion Matrix:")

print(cm)

print(f"Training Loss: {train\_score[0]:.2} \t Accuracy: {train\_score[1]:.3} \t Precision: {train\_score[2]:.3} \t Recall: {train\_score[3]:.3}")

print(f"Validation Loss: {valid\_score[0]:.2} \t Accuracy: {valid\_score[1]:.3} \t Precision: {valid\_score[2]:.3} \t Recall: {valid\_score[3]:.3}")

print(f"Testing Loss: {test\_score[0]:.2} \t Accuracy: {test\_score[1]:.3} \t Precision: {test\_score[2]:.3} \t Recall: {test\_score[3]:.3}")

import itertools

# Create a visualization for the confusion matrix

# Create a figure with a specified size (10x10 inches)

plt.figure(figsize=(10, 10))

# Display the confusion matrix using imshow

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)  # Use nearest neighbor interpolation and Blues colormap

# Set the title of the plot

plt.title('Confusion Matrix')

# Add a colorbar to visualize the range of values in the matrix

plt.colorbar()

# Define tick marks for the x and y axes based on the number of classes

tick\_marks = np.arange(len(classes))

# Set x-axis ticks and labels with rotation for better readability (45 degrees)

plt.xticks(tick\_marks, classes, rotation=45)

# Set y-axis ticks and labels using the same tick marks and class labels

plt.yticks(tick\_marks, classes)

# Calculate a threshold for text color based on the maximum value in the matrix

thresh = cm.max() / 2.

# Loop through each cell (i, j) of the confusion matrix

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

    # Add text labels to each cell showing the corresponding count

    plt.text(j, i, cm[i, j],  # Text content (count at that position)

             horizontalalignment='center',  # Center the text horizontally

             color='white' if cm[i, j] > thresh else 'black')  # White text for high values, black for low values

# Adjust layout to prevent overlapping elements

plt.tight\_layout()

# Set labels for axes

plt.ylabel('True Label')  # Label for y-axis (ground truth)

plt.xlabel('Predicted Label')  # Label for x-axis (model's prediction)

# Display the final confusion matrix visualization

plt.show()