University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project 7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Project Title:

Enhancing brain tumor diagnosis by elevating MRI image classification with cutting-edge deep learning and advanced data augmentation

Student Name and SRN:
Aasim Ghaffar
21084757

Supervisor: **Gulay Gurkan Uygun**

Date Submitted: 29th, August 2024

Word Count: 6627

DECLARATION STATEMENT

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

I have read the detailed guidance to students on academic integrity, misconduct, and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project 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I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged.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 did not use human participants in my MSc Project.

I now permit the report to be available on module websites, provided the source is acknowledged.

Student Name printed:	Aasim Ghaffar
Student Name signature:	Aasim Ghaffar
Student SRN number:	21084757

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Table of Contents

Contents

Abstract:

Brain tumors are potentially life-threatening diseases that need to be diagnosed as soon as possible to be effectively treated. Enhancing patient results and quality of life requires an accurate and timely classification of brain tumors. This research aims to create an advanced deep-learning model that accurately recognizes brain tumors from MRI images. We increase the performance of the Xception architecture by employing data augmentation and optimisation methodologies that include CNN and transfer learning. The primary goal is to create a robust and reliable model that can distinguish between various types of brain tumours, such as gliomas, meningiomas, pituitary tumours, and no tumours at all. We utilize significant data preprocessing and augmentation techniques to improve model generalization using the Brain Tumor dataset on Kaggle. There are 14133 images in the dataset. To accurately identify the brain tumour MRI images, we used a number of transfer learning-trained CNN models, including the Xception, ResNet101, and EfficientNetB3 models. While all the models performed well on the unseen data and predicted very well on the new data, the exception model outperforms all these and achieves 99% accuracy. This accomplishment could significantly improve patient outcomes because of its astounding 99% accuracy rate. The model can efficiently classify brain tumors from MRI images, leading to earlier diagnosis and better patient treatment decisions.

Introduction:

Brain tumours are the most common type of brain disease affecting the brain's central nervous system[1]. Over the last two decades, radiologists have increasingly depended on computer-aided diagnosis (CAD) techniques to improve their accuracy in diagnosing, segmenting, and categorising brain tumors[2]. MRI, also known as magnetic resonance imaging, is one of the oldest and most often utilised imaging techniques for detecting aberrant brain tissue growth. Radiologists have typically used manual interpretation to diagnose brain cancer [3].

However, other challenges to manual diagnosis include the sheer volume of MRI images that must be evaluated, which can lead to costly errors and laborious work. Furthermore, a higher image volume can cause human visual perception to become less sensitive.[4], making manual diagnosis tedious and susceptible to subjective interpretation. This increases the risk of misclassification.[5]and calls for invasive treatments like spinal taps and biopsies [6].

To address these issues, numerous researchers created CAD systems that use MRI images to automatically recognise and categorise brain tumours. The extraction of relevant properties from images is critical to the effectiveness of these systems, with texture features and the Discrete Wavelet Transform (DWT) emerging as significant methodologies.[2].

Texture features obtained via statistical analysis or Gabor filters are used to identify patterns repeated in images.[7-9]. In the meantime, DWT makes feature extraction more effective by yielding substantial information in less time.[10], which is especially helpful when evaluating brain MRI images to classify tumors[11]. Several categorisation algorithms have been investigated to distinguish various tumour kinds from brain images; artificially generated neural networks and SVMs constitute two of the most commonly used classifiers.

[7, 12-14]. These methods use various kernels and architectures to produce precise categorization outcomes. Alternative approaches have also been studied, including dictionary learning, sparse coding, ensemble learning, neuro-fuzzy systems, and k-nearest neighbor (KNN) [5, 12-14].

The GLCM approach was first presented by Haralicket al. [15] Since 1973, researchers have widely used it.Selvaraj and associates[4] By extracting two statistical characteristics from pictures and five GLCM features from different offsets, Kharrat et al. could distinguish between normal and pathological slices. [16]. Differentiated 44 GLCM characteristics to distinguish between benign and malignant tumors. Zarchari & associates [14]. Brain tumors were classified into multiple classes using statistical techniques, GLCM, intensity, shape, and Gabor filters to extract 100 featuresGeorgiadis et al.[17, 18]. Ten features from run-length matrices, 22 from GLCM, and four from histograms were retrieved. This research considers feature extraction techniques for brain tumors and medical images, which have been widely explored in the literature. We selected features with effective discrimination capabilities for differentiating tumors from normal tissue in images.Pattern recognition techniques are used for the classification of Segmented Regions of Interests . With a least squares support vector machine, Selvaraj et al. were able to classify 1,100 ROIs as normal or aberrant with 98% and 96% clarity, respectively.

Dashan El et al. [19]. Following reducing features by the use of principal component analysis (PCA), 80 ROIs were categorised as either typical or abnormal using an FP-ANN classification algorithm, which achieved 97% and 98% accuracy, respectively.

Brain tumour several classes of models for classification have been proposed by [14, 17, 18]. Zacharaki et al. [14] analysed a dataset of 98 pictures. The overall rate for accuracy was 91.7% for low-quality gliomas, 90.9% for glioma multiforme, and 41.2% for metastases.

Employing an LSFT-PNN algorithm, meningiomas, gliomas, and metastases were categorized by Georgiadis et al. [11, 12]. 87.50%, 95.24%, and 96.67% accuracy for metastases, meningiomas, and gliomas, respectively, were reported in one investigation.[17, 20].In a different study, their accuracy was 88.89% for gliomas and 100% for meningiomas. A non-parametric feed-forward neural network that reduces computation load and trains rapidly is the LSFT-PNN classifier. [21]. It alters patterns of training using an irregular least- squares strategy and clusters every class's designs around chosen places.

Objectives:

- To create a sophisticated deep learning model for precisely categorizing MRI pictures of brain tumors.
- Enhancing performance of the Xception architecture and other CNN models through transfer learning and data augmentation optimization techniques.
- To contribute to early diagnosis and improved treatment outcomes for brain tumor patients by providing reliable and efficient MRI image classification models.

Problem statement:

Brain tumours are one of the most serious and sometimes dangerous medical conditions, and early detection and treatment are critical to enhancing outcomes for patients.
Classifying brain tumors is particularly difficult because of their intricate and variable appearance on MRI images. The prevalence of overlapping characteristics and the modest distinctions between tumor types present challenges for radiologists and can result in misdiagnosis. Furthermore, manual diagnosis takes much time, is prone to human error, and can lead to discrepancies in results between various practitioners. These difficulties can harm the patient's prognosis by causing delays in initiating necessary treatment. In light of these problems, automated diagnostic systems that can reliably and quickly identify brain cancers from MRI images are desperately needed. These technologies would help radiologists by giving consistent and accurate assessments, but they would also speed up the diagnostic process, allowing for speedier treatment decisions and maybe improved patient results. One approach that shows promise in resolving this important problem is creating comprehensive deep learning models that leverage the capabilities of convolutional neural networks (CNNs).Methods like transfer learning and data augmentation further enhance them.

Research Contribution:

- Developing and refining models to categorize brain tumor MRI images is a major contribution of this research to deep learning and medical image analysis. Among the major contributions are:
- Model Development: Implementing and fine-tuning three CNN models—EfficientNetB3, Xception, and ResNet101—using transfer learning techniques to achieve high classification accuracy for brain tumor MRI images.

Achieving impressive classification accuracies with Xception (0.99%), EfficientNetB3 (0.97%), and ResNet101 (0.89%) through the application of advanced data augmentation and fine-tuning methods, demonstrating the effectiveness of these techniques in improving model performance.

Real World Application:

Automated brain tumor classification systems are particularly beneficial in telemedicine and remote diagnostics. They provide initial diagnostics support in regions with limited access to medical specialists, ensuring that patients receive timely care recommendations. Furthermore, these systems enable remote monitoring, allowing patients in distant locations to have their MRI scans analyzed by specialists, thus facilitating continuous care without frequent travel.

Literature Review:

Brain tumours are among the most severe cancers, and precise early-stage diagnosis is critical for effective therapy. This review examines various studies that have explored automated methods to improve diagnostic accuracy and efficiency. This study [22] employed a basic Convolutional neural network, or CNN, to categorise three types of brain cancers: meninges, pituitary tumours, and gliomas. The model obtained 98.51% accuracy for training and 84.19% validation accuracy utilising a simple CNN design consisting of one convolutional layer, max pooling, becoming thinner, and a layer that is completely connected.

These results are on par with those of more advanced methods, showing that even simple CNNs can achieve success. J. Seetha et al.[22]employed deep learning algorithms to distinguish between pituitary cancer, meningioma, and glioma in MRI brain scans; the study concentrated on 989 axial images to avoid confusion from different imaging planes. Image augmentation improved the training process for testing fully connected and convolutional neural networks. The best-performing model's average accuracy was 91.43% using five-fold cross-validation. This suggests that deep learning can outperform specialized methods that necessitate extensive pre-processing. [23]. This study emphasised the need for correct tumour classification by providing new neural network models for the MRI classification of images. Based on these three datasets, the model was examined and shown to outperform earlier techniques.

This indicates that CNNs have great potential for automated brain tumor diagnosis.
Brain tumor classification was accomplished by Wadhah Ayadi et al.[24]using deep transfer learning in conjunction with a pre-trained GoogLeNet model. The system used five-fold cross-validation to achieve 98% validation accuracy on an MRI datasets.It also evaluated the system's performance with fewer training samples and examined examples of misclassification, demonstrating the usefulness of transfer learning in situations with a shortage of medical images. [24]Deep learning and CNN are two AI techniques that are suggested to improve the MRI-based brain tumor classification. A brain tumour dataset was utilised to train five pre-trained CNN models (Xception, ResNet50, InceptionV3, VGG16, and Mobile Net), with F1 scores ranging from 97.25% to 98.75%. These higher accuracies demonstrate how effectively the models function to improve detection and treatment planning.
After analyzing 110 images using histological feature extraction, Caulo et al. [25] suggested an approach to categorizing glioma grades. With 83.9% sensitivity and 96.2% specificity, the maximum accuracy of around 95.5% was attained. Multi-parametric sequences provided useful information for classification. Lin et al. developed a system to grade meningioma tumours. [26] Grade I tumours spread gradually and are not cancer; Grade II tumours are both cancerous and not cancerous; and Grade III tumours are usually malignant and grow quickly.
Features like contextual and radiological properties were retrieved without segmentation or preprocessing. Multiple logistic regression was applied to categorization. Results were adequate for the dataset; however, larger datasets are required for validation. The test used 120 private patient MRI pictures (90 Grades I and 30 Grade II or III).
A method for classifying MRI brain tumors was presented by Deepak et al. [27] It involved extracting MRI characteristics using pre-trained GoogLeNet and deep transfer learning. Performance was evaluated on the Figshare dataset using fivefold cross-validation. Despite having fewer training samples, the classification accuracy was 92.3%. Using SVM and KNN classifiers, respectively, this was further enhanced to 97.8% and 98%.Ahmad et al. [28]Use CNN and the Discrete Wavelet Transform (DWT). Classified MRI brain cancers using 510 images from the Figshare dataset with a 99.3% overall accuracy rate. Paul et al.[23]Classified MRI brain tumors. The three classifiers included a random forest, an entirely linked neural network, and a CNN. CNN had the biggest accuracy percentage (90.26%). The model that was suggested included convolutional, maxpool, and fully connected layers. Using this dataset, Abiwinanda et al. [29] examined CNN for brain tumour classification and created 7 neural networks.The second model, with 84.19% test accuracy and 98.51% training accuracy, was the best. It consisted of one fully connected layer and two convolutional layers.
To boost classification outcomes, Tahir et al. [30]Investigated several preprocessing methods and divided them into edge detection, contrast enhancement, and noise removal. After applying several combinations to various image sets, the results indicated that the combined

preprocessing techniques were superior to the individual methods. On the Figshare dataset, they reported an accuracy of 86% using an SVM classifier.

Introduction to CNN:

A Comprehensive Overview of Convolutional Neural Networks (CNNs) [31]. Recently, most followed the concept of CNN, which invokes heterogeneous layers. The purpose is to utilize massive data and enormous feature extraction. Since 1990, researchers have been struggling to improve the DL system. This area of study eventually became known as computer vision.

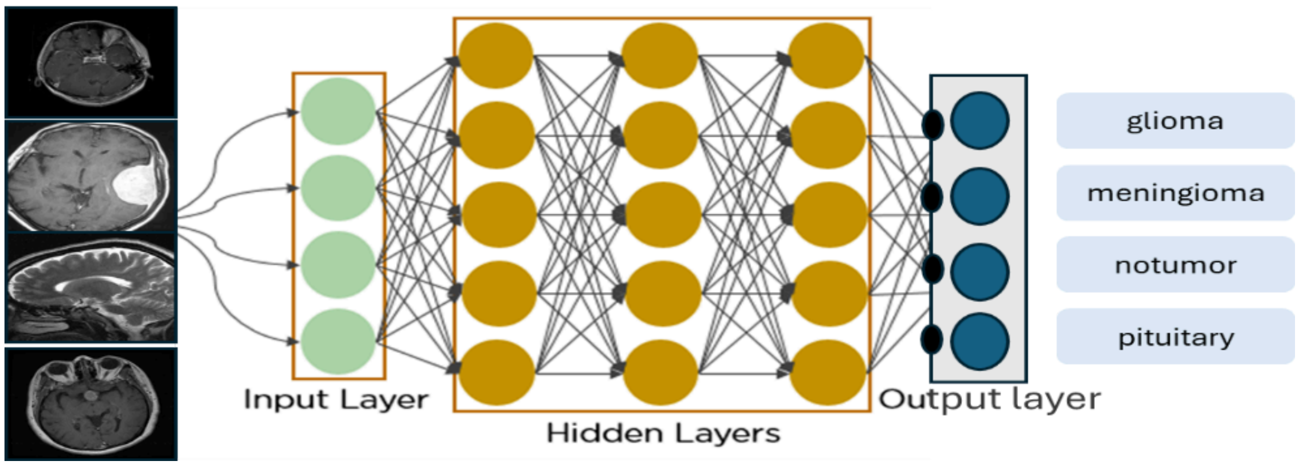


Figure 2.1: Working of Convolutional Neural Network

Background of CNN:

The CNN concept was developed in 1980. It was mainly utilized in the postal industry to read zip codes, PINs, etc. All techniques are limited, so DL requires a high processor to data train or intuition. Thus, it's a big issue. CNNs were restricted to the postal industry and could not enter the ML field.

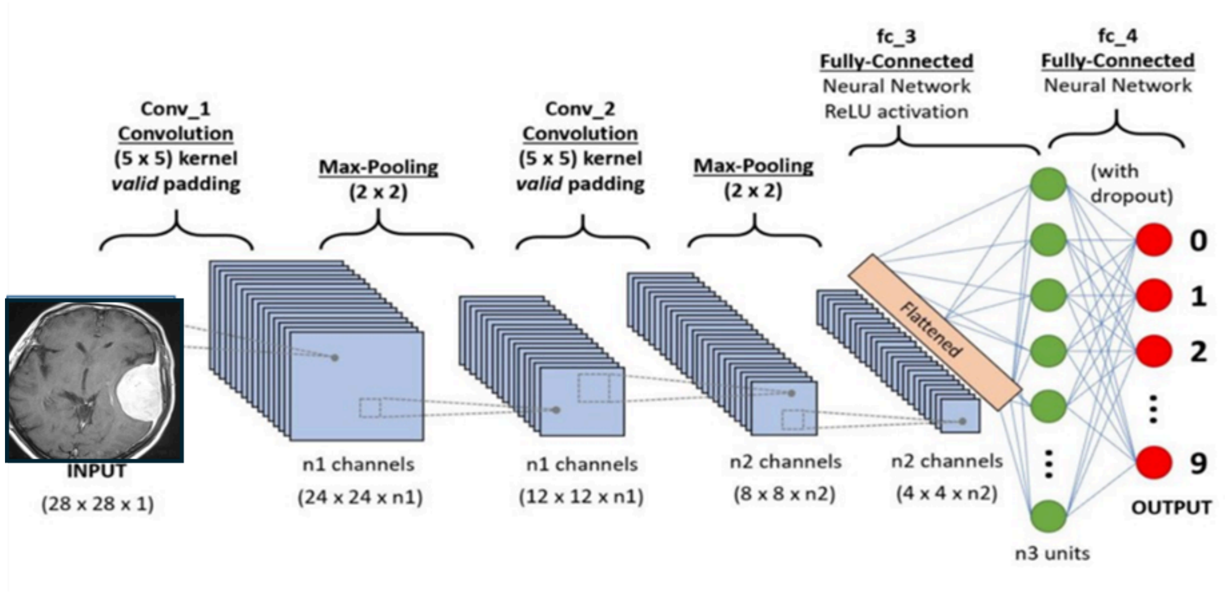


Figure 2.2: Architecture design of CNN[32]

Pooling Layers:

Many of the pooling layers are briefly explained [33]. As the name implies, the pooling layers in CNN combine the features discovered by the convolutional layer feature map. Because they frequently employ the maximum or average values of the input to down-sample the data, pooling layers are relatively straightforward. To reduce the computation process, often using a pooling layer after convolutional layers gives less output. The featured map is consolidated by pooling layers, negating the model's need to be trained on classifying features. As a result, a model becomes more reliable and sturdier.

Max Pooling Layer:

The maximum pooling layer performs its tasks by using the most apparent feature retrieved from the convolution layer's map of features. The element with the greatest value is selected from the regions obtained by the filter's results in every feature map. Low-level features like edges, points, etc., are extracted from the data via the Max pooling layer. Moreover, max-pooling eliminates the most obvious features—the finest lower-level image representations—while discussing image processing.

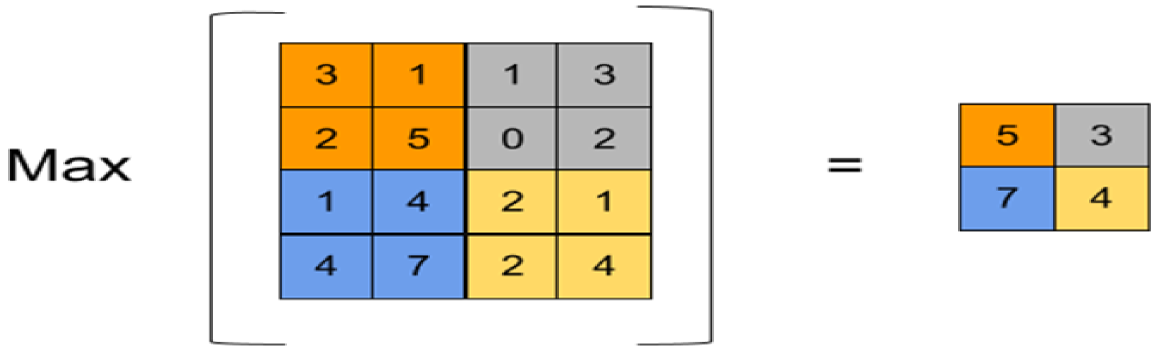


Figure **2.3: MaxPooling Layer with two-dimensional layer map** ***[34]***

F.C Layer:

The F.C layer in [35] Introduces briefly. It’s also called hidden layers. It’s at the last stage of CNN. The structure of the fully connected layer figure [36] It is shown below.

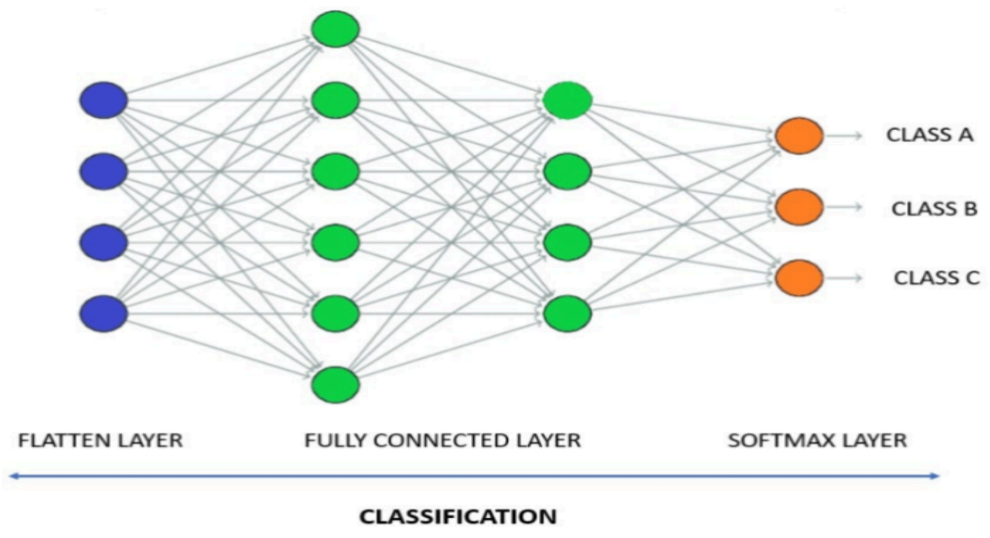


Figure 2.4: Structure of F.C layer

Methodology:

Overview

This chapter describes how to use low-quality MRI images to identify and categorize brain tumors using an efficient and effective methodology for medical image processing.Fig.1 [37]. Displays the suggested model's block diagram. The proposed approach can identify and categorize tumors, locate them from MRI images, and extract strong features. It may also be a useful addition to the current techniques for classifying brain tumors. This applied research project will offer a second viewpoint to assist radiologists in determining the kind, location, intensity, and diameter of the tumor. When brain tumors are detected early, doctors can create treatment plans that are more successful and provide better results for the patient [21]. Fig. 2 [37]. Shows the structure of the suggested brain tumor classifier. Three separate MRI images for each patient were fed into the network individually. Furthermore, each disease is explained in detail.

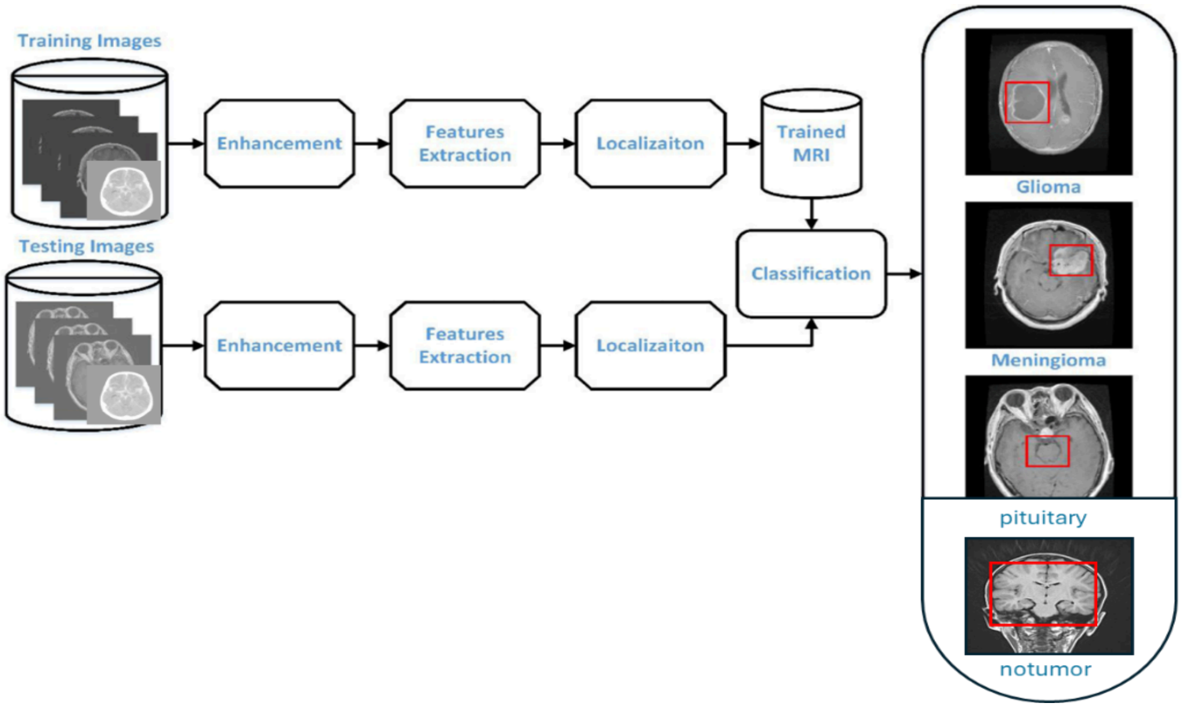


Figure 3.1: The model architectural design with predicted outcomes [37]

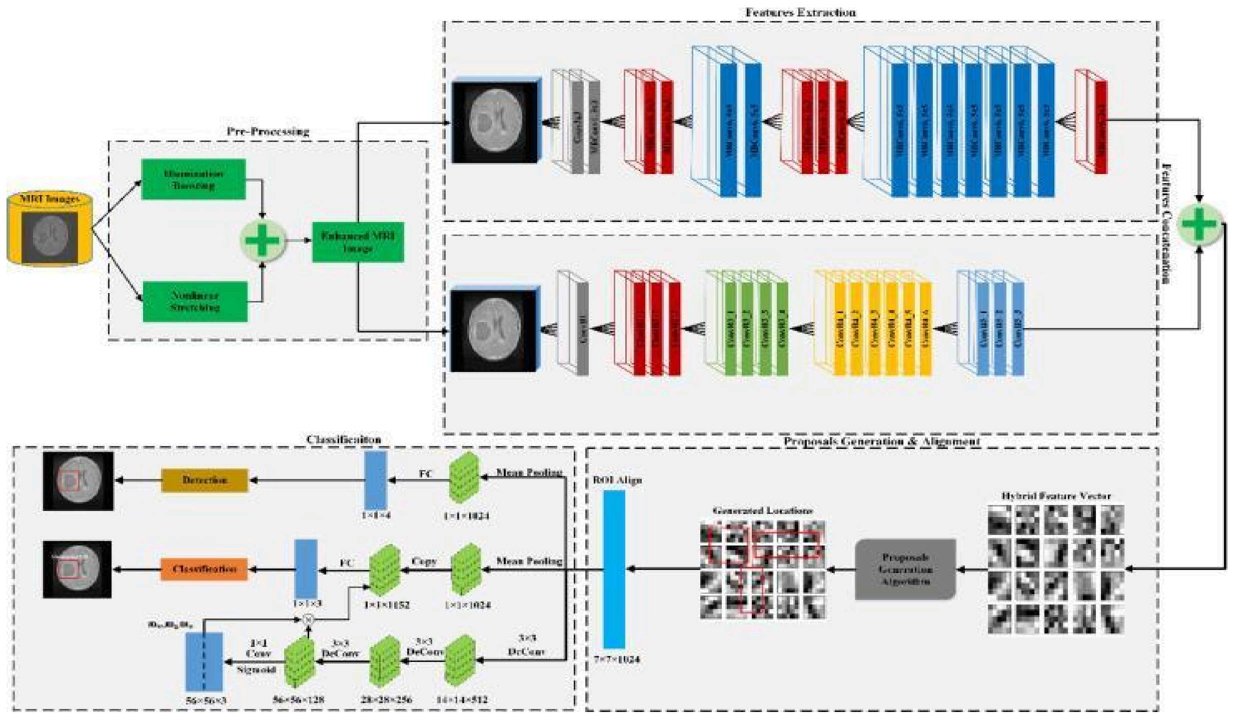


Figure 3.2: The suggested model's general structural layout [37]

Proposed Method:

To maintain consistency, a dataset of brain MRI images is first gathered and preprocessed as part of the suggested method for brain MRI categorization. To take advantage of the ability of preprocessed images to extract features, a pre-trained Xception model is fed into it via transfer learning. The foundation model layers stay frozen while new, fully connected layers are introduced and initially trained. A fine-tuning phase ensues, during which the final layers of the underlying model are unfrozen and optimized for optimal performance. Training and validation enable monitoring of the model's losses and reliability. The efficiency of the model is finally evaluated using confusion matrices and classification reports to guarantee that brain MRI images are correctly identified as glioma, meningitis, no tumour, or hypothalamic. The findings aid in the precise diagnosis and planning of neurological conditions treatment.

Data Collection and Preprocessing:

Dataset:

This work used a collection of 7023 human brain MRI images gathered from three main sources [37]: Br35H, the SARTAJ dataset, and figshare. Gliomas, meningiomas, no tumor, and pituitary are the four separate classifications into which every image in the collection has been carefully classified.This careful categorization ensures the quality and reliability of the data. After applying the augmentation technique, the total number of images is 14245.

1. **Figshare:** This repository provided a significant portion of the dataset and was a reliable source for diverse brain MRI images across various conditions. The images sourced from figshare were crucial in ensuring a comprehensive representation of brain tumors and healthy brain scans.
2. **SARTAJ Dataset:** Initially included in the dataset, the SARTAJ dataset posed challenges due to misclassified glioma images. Upon discovering discrepancies through comparative analysis with other datasets and models, the glioma images from SARTAJ were replaced with more accurately categorized images from figshare. This corrective

action ensured the dataset's integrity and enhanced the accuracy of tumor classification tasks.

3. **Br35H:** This dataset contributed specifically to the 'no tumor' class images. These images are essential controls, allowing for comparative analysis against tumor-present images. Incorporating Br35H images offers a foundation for assessing how well tumor detection and classification models differentiate between tumor-free and abnormal brain scans. A critical component of the model's classification task is distinguishing between healthy and pathological brain images, which requires knowledge of the 'no tumor' class. Pituitary.

Why Choose this dataset?

The choice of this dataset is justified by its diversity and comprehensive representation of various brain conditions. The dataset comprises a large number of brain MRI pictures, guaranteeing that the model is competent for a wide range of brain disorders. Corrective measures are taken to ensure data accuracy and suitability for advanced deep learning techniques. Together, these factors increase the likelihood of discoveries in recognizing and classifying brain tumors, which would enhance medical imaging and patient care.

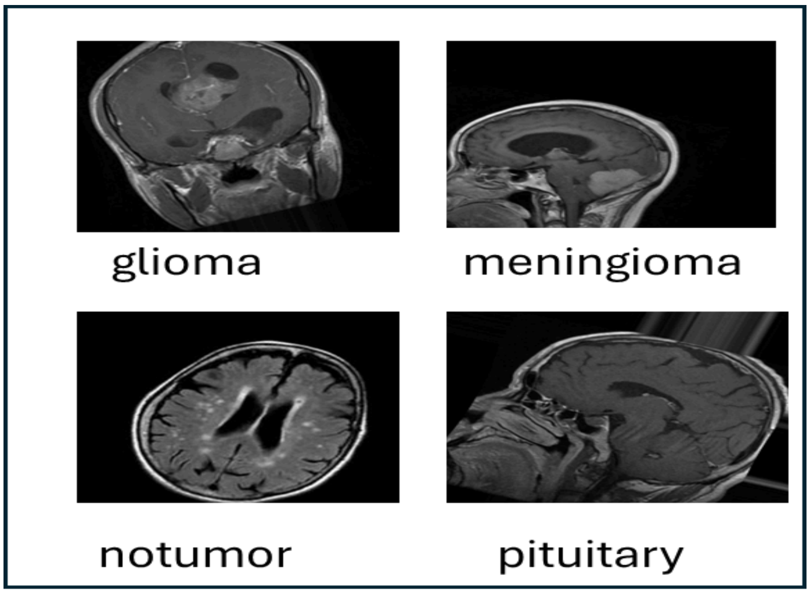


Figure 3.3: Dataset Overviewof Augmented Images

Data Distribution

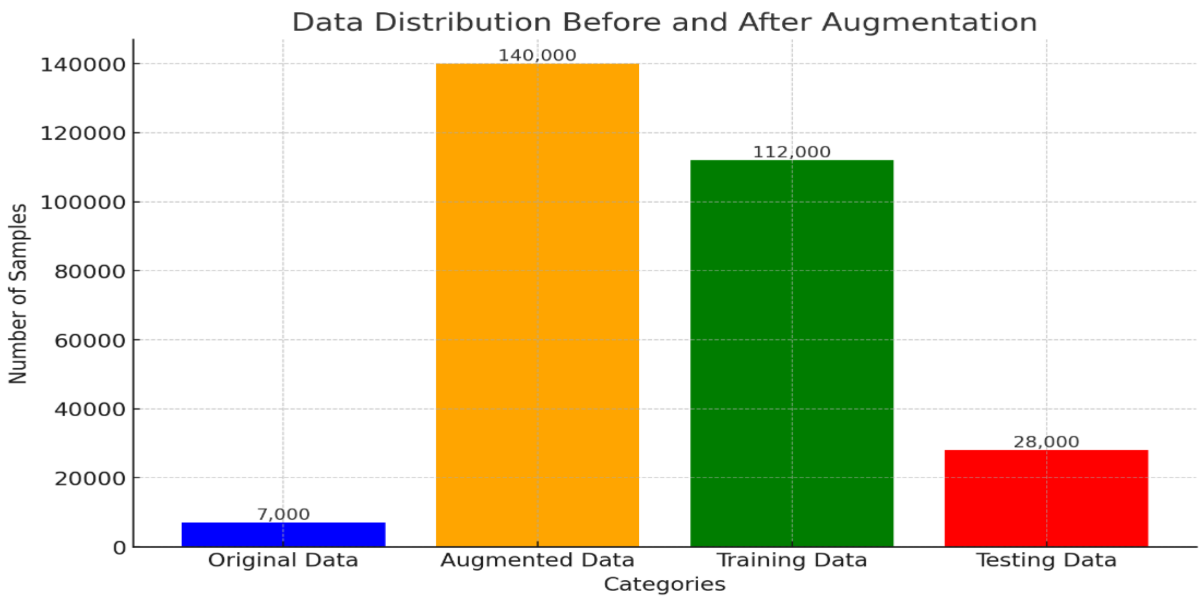


Figure 3.4: Histogram of Data Distribution

Pituitary Brain Tumors:

Pituitary brain tumours are unusual tumours that originate in the pituitary gland. This vital organ governs growth, gland function (such as the thyroid gland and endocrine glands), and the operation of multiple organs (including the kidneys, breasts, and uterus). Certain forms of pituitary tumours can cause hormone overproduction or underproduction. The majority of pituitary tumours, known as benign adenomas, remain within the gland that produces testosterone or its surrounding tissues rather than spreading to other areas of the body.

Glioma Brain Tumor:

Tumors called gliomas are produced by glial cells, which are the cells that envelop and sustain nerve cells. They originate in the spinal cord. The precise glial cell type implicated in these tumors and their genetic makeup, which can assist in forecasting their long-term behavior, are used to classify them. Gliomas can cause headaches, nausea, visual issues, memory loss, behavioral abnormalities, and seizures, especially in people who have never had seizures before.

Meningioma Brain Tumor:

Meningiomas are tumours that develop from the meninges that the coverings that surround the brain and the spinal cord.. Although it isn't technically considered a brain tumor, it fits into this category because it can compress nearby blood vessels, nerves, and brain structures.
The most frequent kind of brain tumor, meningiomas, usually show slow-growing symptoms, such as headaches (which are generally greater in the morning), vision changes, hearing issues, memory loss, seizures, limb weakness, and language difficulty.

Pre-Processing:

Preprocessing is required to improve the reliability and value of data in a deep-learning system. The brain tumor dataset is preprocessed using several methods to maximize data integrity and model performance. The suggested MRI classification model's preprocessing step makes sure that the input images are uniform in size and format, which is necessary for effective training and precise predictions. The original dataset consists of four brain MRI images: pituitary, meningioma, glioma, and no tumor. The photos are stored in a directory structure with one subdirectory for each of the four classes.

Image Rescaling:

The MRI images' pixel values are modified during the preparation stage of image rescaling to provide uniformity and the best possible input for the deep learning model. The original images' pixels are all normalized, with each pixel originally indicating a grayscale intensity range of 0 to 255. Divide the value of a pixel by 255 to normalize it.The input can be standardized for all images by converting the pixel values to a scale of 0 to 1. The model can converge faster during training if the pixel values are normalized because the gradients become more stable, and the model parameter updates become more consistent.

Data Augmentation for Training Data:

During the learning phase, data enhancement is an important technique for artificially increasing the variety of the training dataset. This tactic enhances the model's generalization of new data by avoiding overfitting. Several augmentation methods are frequently applied to brain MRI images:

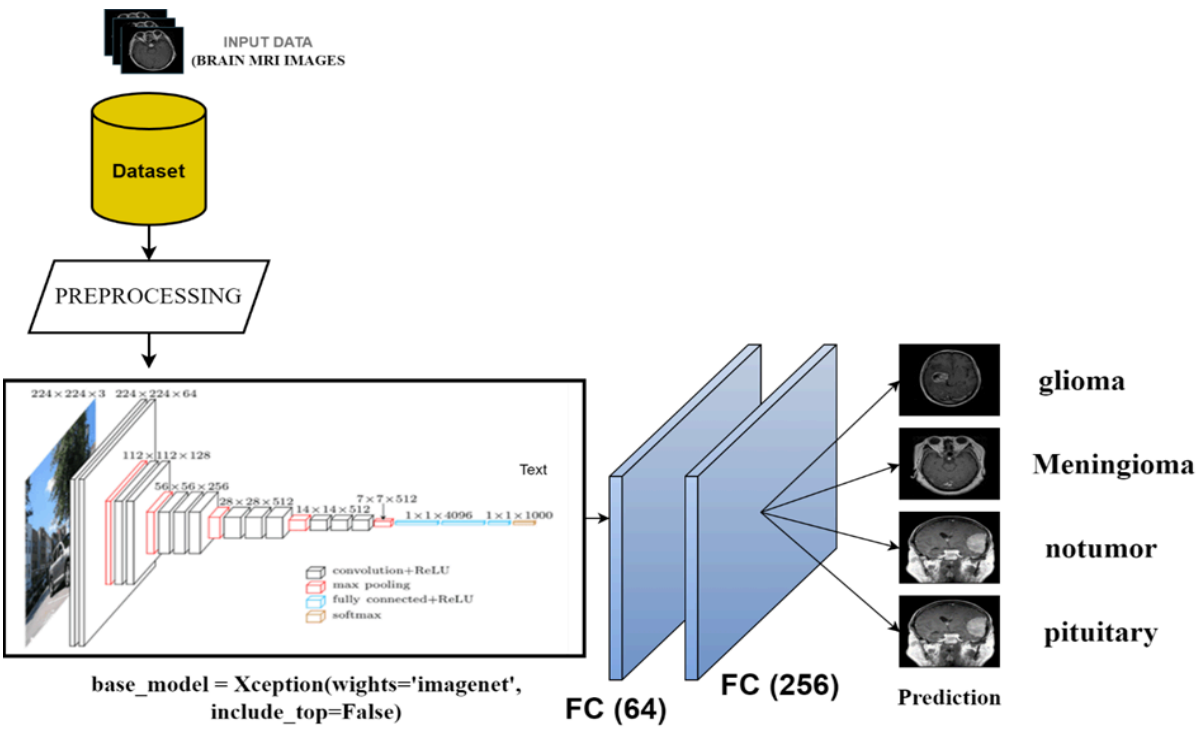


Figure 3.5: Proposed Model Architecture

Workflow of Xception Model

The "Extreme Inception" deep learning model, known as Xception[38], was created by Francois Chollet of Google. It expands on and improves the Inception architecture, increasing its usefulness and renown in the deep learning community. The Inception architecture, especially Inception v3, serves as the model's primary source of inspiration. It seeks to increase deep convolutional neural networks' efficacy and computational efficiency. Xception uses depth-wise separable convolutions to improve computational performance. Depth-wise convolutions in Xception process each input channel independently, saving computational overhead, in contrast to typical convolutions that employ a single filter for all input channels. Point-wise convolutions come next, in which the outputs of depth-wise convolutions are combined into a 1x1 convolution to capture complex patterns while keeping the model lightweight.

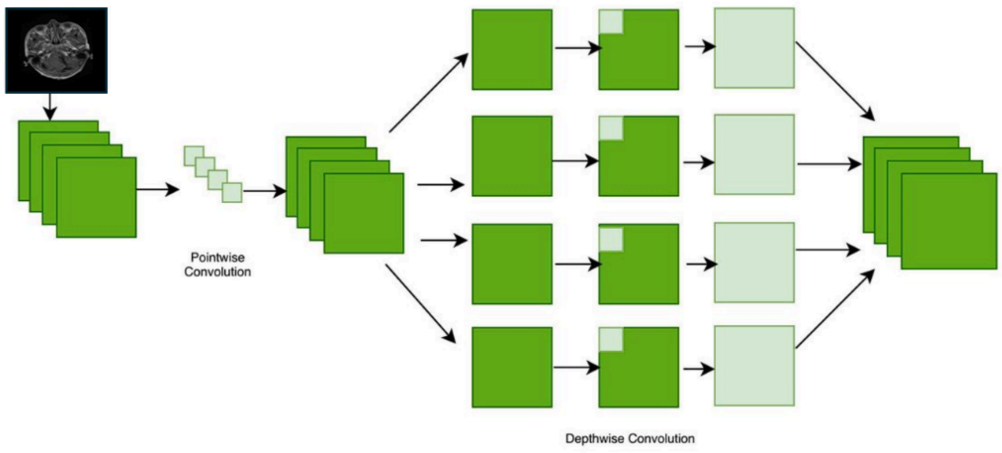


Figure 3.6: Depthwise Convolution [39]

Separable convolution based on depth forms the basis of the Xception model. Therefore, let's examine them before delving into specific Xception architecture components.

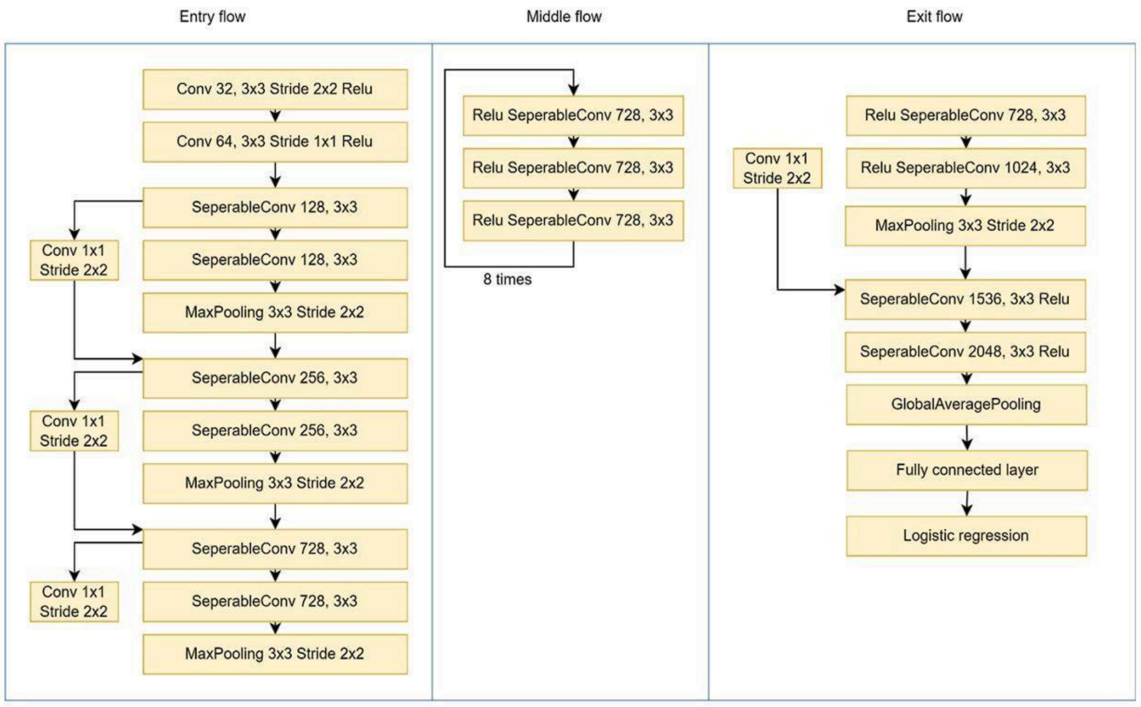
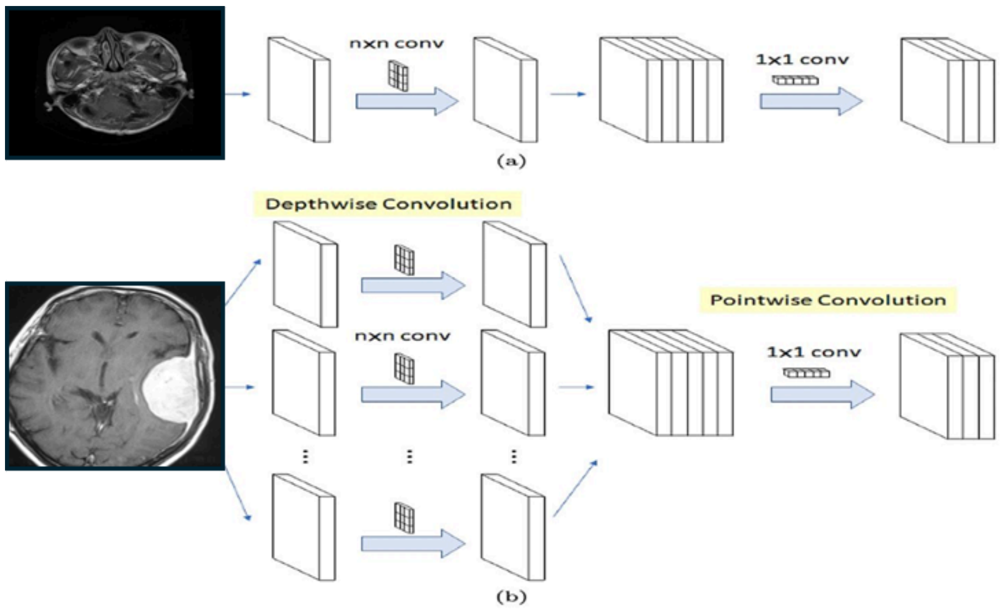

Entry flow
Middle flow
Exit flow
Conv 32, 3x3 Stride 2x2 Relu
Conv 64, 3x3 Stride 1x1 Relu
SeperableConv 128, 3x3
Conv 1x1 Stride 2x2
SeperableConv 128, 3x3
MaxPooling 3x3 Stride 2x2
SeperableConv 256, 3x3
Conv 1x1 Stride 2x2
SeperableConv 256, 3x3
MaxPooling 3x3 Stride 2x2
SeperableConv 728, 3x3
Conv 1x1 Stride 2x2
SeperableConv 728, 3x3
MaxPooling 3x3 Stride 2x2
Relu SeperableConv 728, 3x3
Relu SeperableConv 728, 3x3
Relu SeperableConv 728, 3x3
8 times
Relu SeperableConv 728, 3x3
Conv 1x1 Stride 2x2
Relu SeperableConv 1024, 3x3
MaxPooling 3x3 Stride 2x2
SeperableConv 1536, 3x3 Relu
SeperableConv 2048, 3x3 Relu
GlobalAveragePooling
Fully connected layer
Logistic regression

Figure 3.7: Xception Architecture[40]

Loading and Importing of Data:

As previously stated, the data set was obtained from Kaggle and is essentially a compilation of three distinct data sets: Br35H (Hamada, 2020), SARTAJ (Bhuvaji, 2020), and figshare (Cheng, 2017), which was carefully selected by Nickparvar (2021). 5712 meticulously chosen and arranged photos are included in the training dataset for analysis. These pictures offer numerous brain MRI scans, which I used as the starting point for my project. The MRI scans of 1457 pituitary, 1339 meningioma, 1321 glioma, and 1595 no tumors are included in the training folder. The dataset is separated into testing and training folders.



Figures 3.8: Standard CNN. (b) Depth wise Separable[41]

The Xception model uses this in a somewhat modified form. We execute depthwise convolution first, followed by pointwise convolution in the original depthwise separable convolution. Using different nxn filters, the Xception model performs pointwise convolution (1×1) and depthwise convolution.

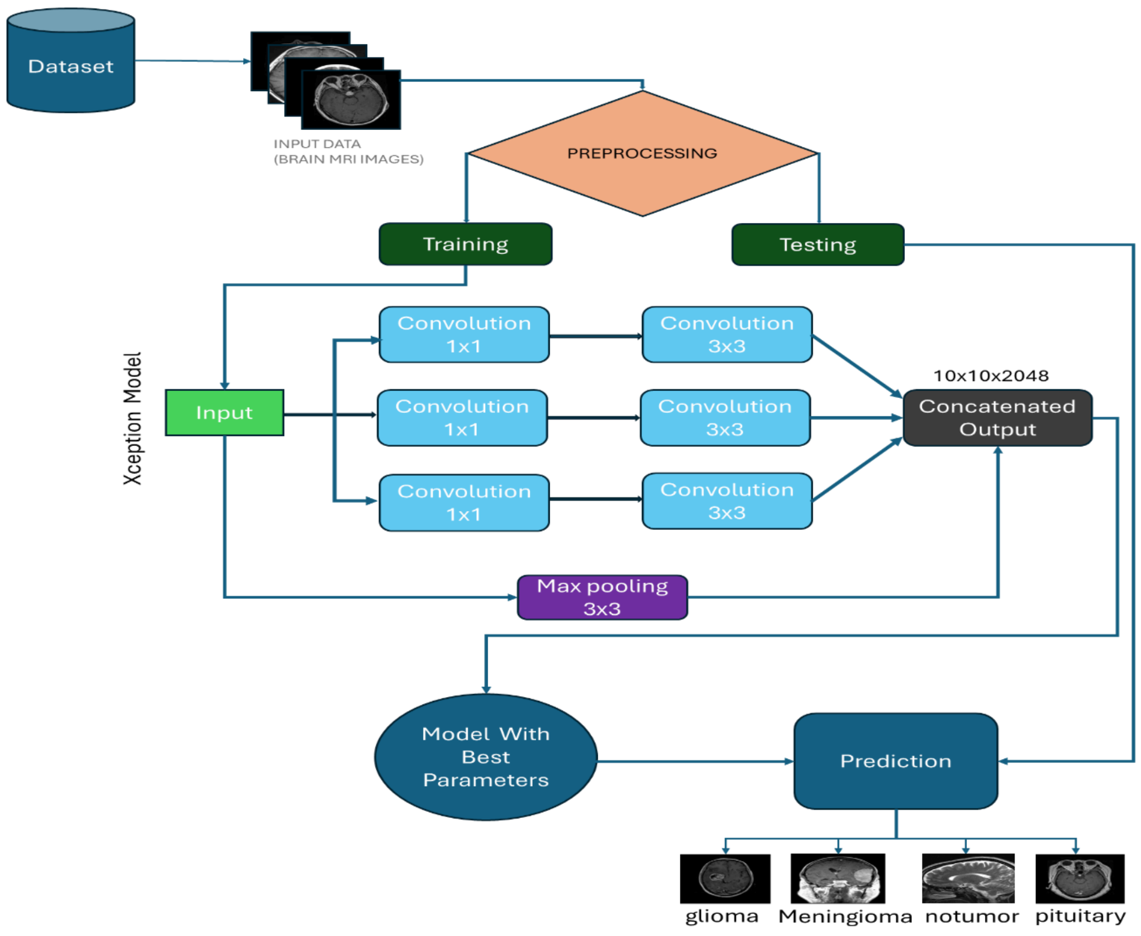


Figure 3.9: The proposed model flow for predicting.

Proposed Model:

In my architecture, I applied transfer learning using the Xception model, well-known for its deep convolutional neural network architecture with depth wise separable convolutions that effectively capture spatial and channel-wise correlations in images. I customized the model for my classification task by removing the top layers and adding custom dense layers. To reduce spatial dimensions, a global average pooling layer was added. Additionally, fully linked layers with 64 and 256 units, respectively, were integrated, and non-linearity was achieved by utilizing ReLU activation. SoftMax activation was employed in the last layer to create probabilities for every class in the dataset. To minimise errors in prediction, the model was developed with the Adam optimiser and the category entropy cross-loss function was used.
Utilizing improved data, the model was trained, its performance was verified on an independent dataset, and its weights were stored for further use. The model's performance across several classes was revealed by the visualization of training parameters, such as accuracy and loss, and analysis using a confusion matrix and classification report. To maximize accuracy in image classification tasks, this strategy made the most of the advanced architecture and transfer learning capabilities of the Xception model. I used Xception, ResNet101, and EfficientNetB3 three pre-trained models to predict brain cancers in four classes. All models did well, but Xception excelled, with 99% validation and training accuracy.

Why Choose the Model?

There are 71 layers in the Xception. [42]TL algorithm. Feature extraction is built on top of the 36 convolutional layers of the Xception network. Each convolutional layer is surrounded by linear residual connections. Furthermore, the core of the Xception, as the network is made up of depth-wise separated layers of convolution. The structure of the Xception model is highly customisable. After training on millions of pictures from the Picture Net dataset, the already trained variant of the Xception TL engine may be able to detect new tasks associated with them. The model utilised in the suggested Network:
In this study, I used three pre-trained models to predict brain cancers across four classes: Xception, ResNet101, and EfficientNetB3. The Xception model, covered in section 3.4, is recommended because of its effective accuracy. We'll talk about the other two models now.

Resnet101:

ResNets, which relies on the VGG-19 model, is one of the most sophisticated architectures suggested for detecting objects and classification of images ImageNet apps. Convolutional neural networks (CNNs) typically consist of successively connected layers trained to learn different features. All the layers in the ResNet architecture can learn various feature levels. In ResNet-101, there are usually 33 filters in each convolutional layer. The number of filters in ResNet doubles when the feature map size is halved. It remains constant for an output feature map size of the same size in order to ensure consistent temporal complexity across layers. Down sampling in ResNet-101 is directly achieved by the use of convolutional layers with a stride of two. SoftMax-enabled fully connected layer and ResNet-101 global average pooling layer round out the system[43] .

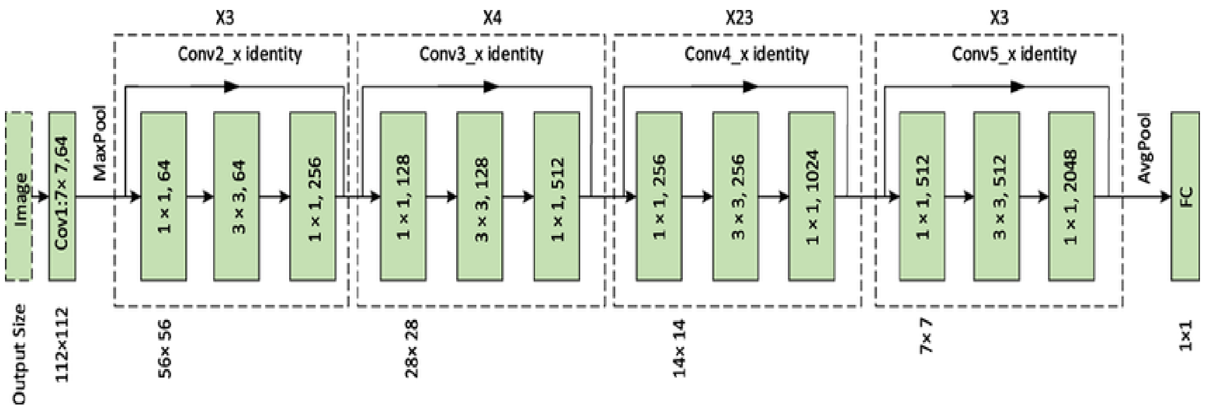


Figure 3.10: Resenet101 architecture[44]

Efficientnetb3:

The architecture of convolutional neural networks the Efficient Net family includes EfficientNetB3. It is renowned for achieving cutting-edge results in image recognition applications by striking a balance between the size of models and processing power. EfficientNetB3, created by Mingxing Tan and Quoc V. Le in 2019, is built on an additive scaling algorithm that grows depth, width, and clarity proportionally. This method finds the best balance between these characteristics, enabling it to outperform other models in accuracy and efficiency. In particular, EfficientNetB3 is deeper and has more parameters than its predecessors, such as EfficientNetB0, which makes it more suited for applications requiring more intricate feature extraction and higher precision.

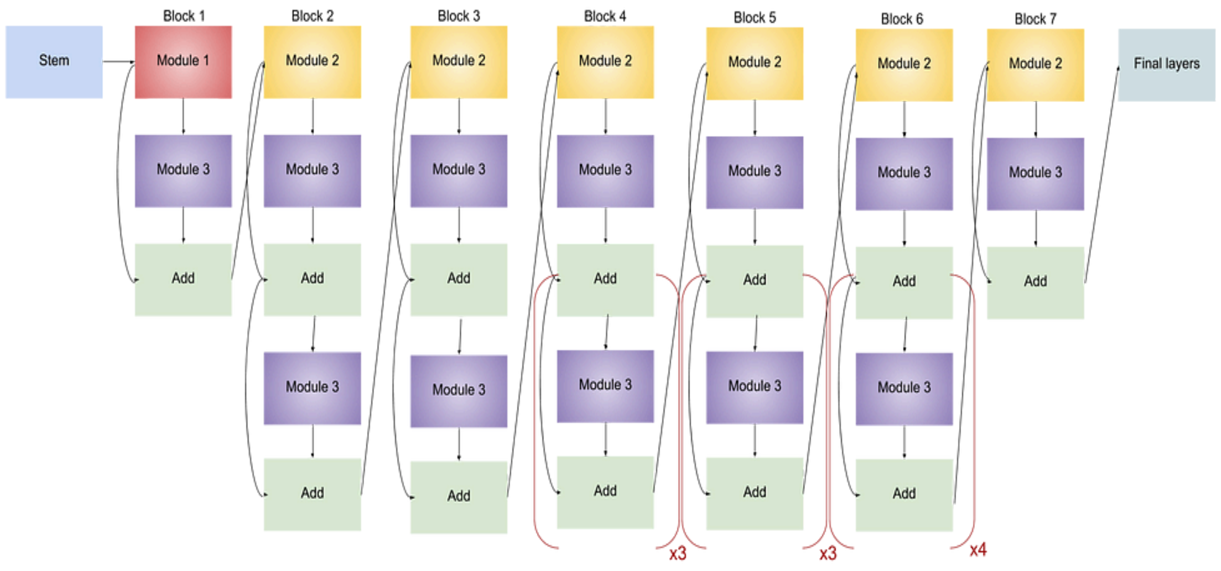


Figure 3.11: EfficientNetB3 architecture [45]

Reason for choice of the models:

I chose EfficientNetB3, Xception, and ResNet101 models to classify MRI images due to their advanced architectures, which have proven performance in image classification tasks. EfficientNEtB3 offers an optimal balance between efficiency and accuracy through compound scaling, making it resource-efficient while maintaining high accuracy. Xception performs remarkably at a lower computational cost and parameter count by utilizing depth-wise separable convolutions. ResNet101, with its deep architecture and residual connections, effectively captures complex features and patterns, which are crucial for the detailed analysis required in medical imaging. These models collectively ensure robust, efficient, and accurately classified MRI images.

Results and Discussion:

Overview:

In the present research, three models that had been trained (Xception, ResNet101, and EfficientNetB3) were utilized to categorise brain tumour images into three categories. Xception demonstrated 99% accuracy on the training and validation datasets, making it the best model. ResNet101 had 99% training accuracy but only 89% validation accuracy, while EfficientNetB3 came in second with 99% and 97% validation accuracy. This research has a substantial potential impact on the classification of medical images. The architecture of Xception, which effectively captures spatial and channel-wise correlations using depth-wise separable convolutions, is the reason for its outstanding performance. As a result, there is less overfitting and improved generalization due to reduced parameters and computational expense. ResNet101's intricate structure led to severe overfitting, even with its conventional convolutions and residual connections. Due to its dependency on typical convolutions, EfficientNetB3, despite its balanced design, did not perform as well as Xception. Overall, Xception outperformed ResNet101 and EfficientNetB3 as the most successful model for classifying brain tumors because of its efficient architecture and effective feature extraction.

Preliminaries:

The investigation made use of brain tumor MRI pictures. The entire system is implemented in Python. This work makes use of a network of deep convolutional neural networks (DCNN). The system has 16 GB of RAM and runs Windows 10. It also has an NVIDIA GeForce RTX 3060 SUPER devoted graphic card. The model is developed using a combination of Python and several deep learning modules listed in Table 1.

Name	Description
Operating System:	**Windows® 10 Education (OS build: 19045.4529)**
Processor:	**12th Gen Intel(R) Core(TM) i5-12400 2.50 GHz**
Graphics:	**NVIDIA GeForce RTX 3060**
Coding	**Python 3.8 for Windows® Operating System**
Libraries	**Opencv, Keras, tensorflow, numpy, sklearn, os, matplotlib.**

Table 1: Software and hardware specifications

Xception Model:

The Xception model is trained on MRI images of brain tumors in order to efficiently classify diseases. After 16 epochs of training, the suggested model produced results with a high level of accuracy. After completing the Xception model training, the model weights are saved with H5. A graph is drawn to visualize the accuracy of the training and validation. Figures 4.1 and 4.2 show the proposed model graphs, showing how the model learned during the training time.

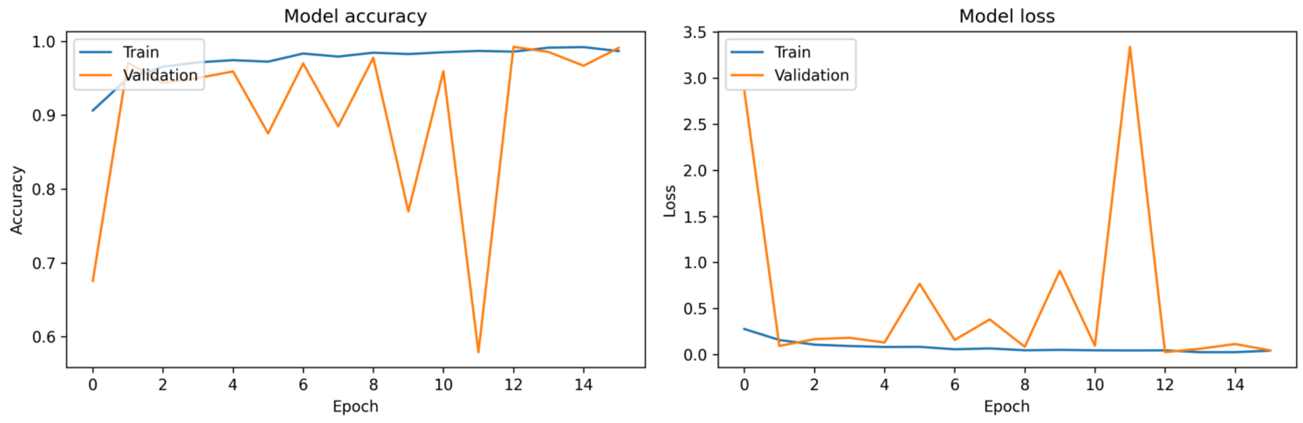


Figure 4.1: Training and Validation Accuracy and Training & validation loss graphs

The red line in Figure 4.1 denotes validation accuracy, while the blue line denotes training accuracy. Both lines show a steady upward trend throughout the training process, improving ongoing training and validation accuracy. After training, the model reaches an optimal accuracy of 99% during both training and validation, demonstrating the Xception model's strong generalization abilities and outstanding performance on unknown data. In Figure 4.2, validation loss is shown by the red line, and the blue line represents training loss. Both lines consistently decrease throughout training, as seen in Figure 4.1, suggesting that the loss decreases as the model gains knowledge. This decrease in loss corresponds with the level of reliability improvement in Figure 4.1, indicating that the model has no overfitting and is capable of learning from the information at hand. When taken as a whole, these visualizations highlight the Xception model's strength and efficacy, demonstrating its capacity to attain high accuracy with little loss and verifying its appropriateness for the given categorization task.

Evaluation Metrics:

Class	*Precision*	*Recall*	*F1-score*	*Support*
Glioma	0.23	0.23	0.23	689
Meningioma	0.24	0.24	0.24	690
No Tumor	0.29	0.28	0.29	756
Pituitary	0.29	0.30	0.29	713
Accuracy	-	-	**0.26**	**2848**
Macro Avg	0.26	0.26	0.26	2848
Weighted Avg	0.26	0.26	0.26	2848

Table 2: Evaluation Metrics of the Xception Model

Table 2 shows the precision, recall, F1-score, as well as support (number of cases) for meningioma, the pituitary, lymphoma and and no tumours. Furthermore, it displays the general accuracy, macro a typical basis, and average weighted for all classes.

Confusion Matrix:

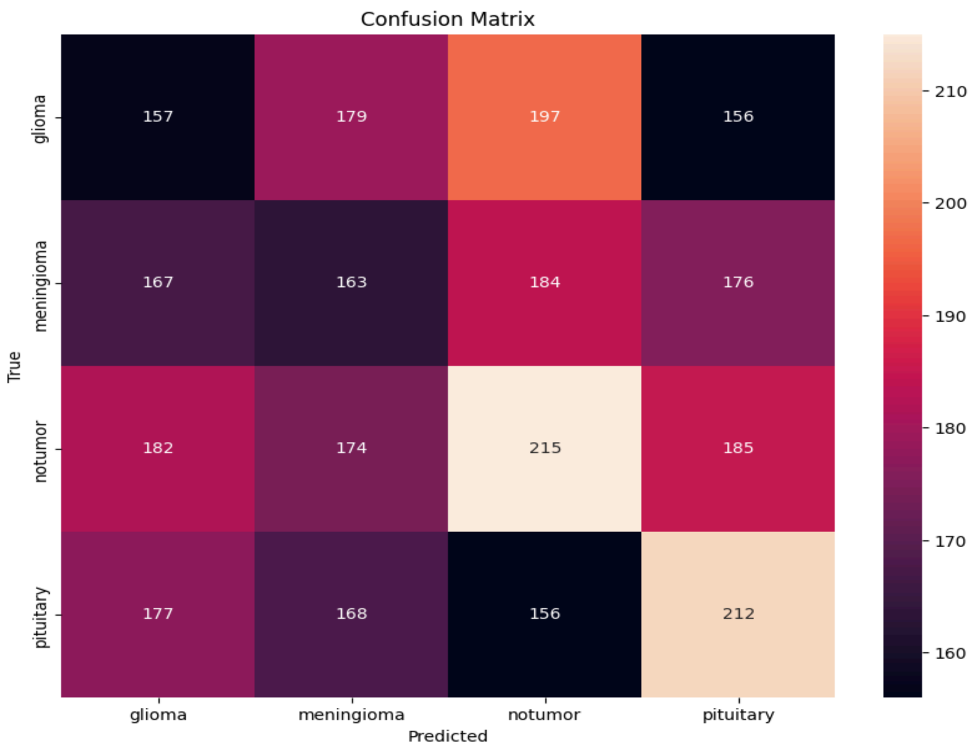


Figure 4.2: Confusion Matrix

An important technique for assessing the effectiveness of classification algorithms is a confusion matrix, especially in multiclass classification scenarios. By carefully comparing the model's predictions to the actual classifications, it assists in differentiating between accurate and inaccurate classifications. This work identifies four types of MRI images glioma, meningioma, no tumor, and pituitary using the confusion matrix. The model successfully predicted the true positives for each tumor type, represented by the matrix's diagonal values (157, 163, 215, and 212). To illustrate the model's great classification capacity, the following categories were identified accurately: glioma was classified 157 times, meningioma 163 times, no tumor 215 times, and pituitary 212 times.
But the matrix also reveals notable misclassifications. A total of 176 times, glioma was mislabeled as meningioma, 197 times as no tumor, and 156 times as pituitary. Similarly, 167 meningioma cases were mistakenly identified as gliomas, 184 were tumorless, and 176 were pituitaries. Incorrect diagnoses for gliomas, meningiomas, and pituitaries were made in 182 cases, 174 of which fell into the no-tumor group. Pituitary tumors were misclassified in 177 cases as meningiomas, 156 as no tumors, and 177 as gliomas. These incorrect classifications point to potential areas for model optimization, addressing any class imbalances in the dataset and utilizing methods like class weighting or resampling to improve the model further. Improving the model to lower these misclassifications is essential to boosting its dependability and assisting in the more precise and effective identification of brain tumors.

Model Type	Models	Epoch	Training Loss	Training Acc	Val Loss	Val Acc
Pre-trained Models	EfficientNetB3	116	0.0305	0.9908	0.0296	0.9733
	Resnet101	116	0.0766	0.9736	0.3433	0.8982
Proposed Model	Xception	116	0.0438	0.9868	0.0466	0.9912

Table 3: Comparison of the pre-trained models with the proposed model

Work	*Method*	*Training data*	*Accuracy*
Jun Cheng[27]	BoW-SVM	80%	91.28%
Ismael[46]	DWT-Gabor-NN	70%	91.90%
Pashaei[47]	CNN-ELM	70%	93.68%
Nyoman[29]	CNN	-	84.19%
Afshar[48]	CapsNet	-	90.89%
Proposed	Xception	80%	99.12%

Table 4: Related works & comparison with Stat-e-of-the art techniques

<u>Comparison of the models:</u>

The classification performance of MRI images of brain tumors over 16 epochs for three models Xception, Resnet101, and EfficientNetB3 is compared in Table 2. With a validation accuracy of 0.9733 and a training accuracy of 0.9908, EfficientNetB3 showed excellent performance. Resnet101 had lowest validation accuracy at 0.8982, suggesting possible overfitting despite having a high training accuracy of 0.9736. With the greatest validation accuracy of 0.9912 with the greatest outcome having a training accuracy of 0.9868, the Xception model specially designed for this task showed to be the most reliable model in this study for accurately identifying brain cancers.

<u>Xception Model Prediction:</u>

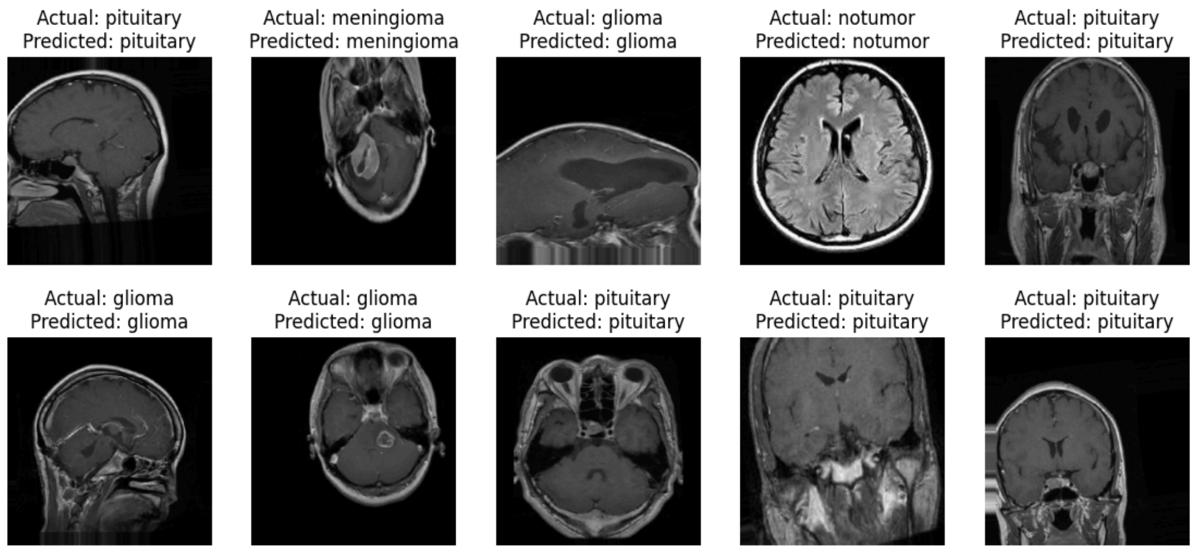


Figure 4.3: Model Prediction

For the Xception model, MRI pictures were classified into four distinct groups: no tumour, meningioma, glioma, and pituitary tumour. According to the findings, the model's accurate forecasts indicate that it learnt well. The model reliably and consistently recognized the pituitary tumors in the first, sixth, seventh, eighth, and ninth images. The second image correctly identified a meningioma, demonstrating its capacity to identify this form of tumor. In the third, fifth, and sixth images, the model correctly predicted the presence of gliomas, providing additional evidence of its accuracy in classifying this condition. In the fourth image, the model also identified a healthy brain scan devoid of a tumor.

Conclusion and Future Work:

Conclusion:

The model has been trained to distinguish between various forms of brain tumors and healthy brain scans, as evidenced by its overall performance in all four categories.
In this work, we have introduced a thorough method using cutting-edge deep-learning models to categorize MRI images of brain tumors. We also presented a supervised learning strategy based on CNN, which uses deep learning algorithms for multiclass categorization. Four classes glioma, meningioma, pituitary tumors, and non-tumorous images were successfully classified using the brain tumor MRI image collection. Several pre-processing methods, including rescaling, resizing, and normalizing, were used to improve performance. Data augmentation methods were also used to prepare the data for training.The dataset was used to test several pre-trained CNN models incorporating transfer learning, including Xception, ResNet101, and EfficientNetB3. Out of all of them, the Xception model outperformed conventional procedures in terms of accuracy and dependability, outperforming the others.

The Xception model outperformed previous models with a validation accuracy of 99% after being trained on multiple datasets and improved through data augmentation. Because of its great accuracy, the model has the potential to help with the rapid and accurate identification of brain tumours, which is critical for improving the treatment of patients. Even with the positive findings, a few issues with this study need to be fixed. The model's generalizability may be impacted by biases towards specific tumor types or imaging situations, as the large dataset may not accurately reflect the variety observed in actual clinical settings. Furthermore, even though the model's accuracy was quite high, there were times when specific tumor kinds with minute variations were incorrectly diagnosed. This underscores the necessity for additional improvement in differentiating across related tumor categories.

Limitation of the work:

This work faces several limitations. The model may overfit the specific dataset, affecting its performance on new, unseen data. The dataset's lack of diversity and imbalances among tumor types might hinder the model's generalizability and accuracy. Additionally, the models' diagnostic capabilities are limited by relying solely on MRI images and not incorporating other clinical information. Moreover, the absence of real-world validation raises concerns about how well the model will perform in diverse settings. These factors suggest that further refinement and broader testing are needed.

Future work:

To improve the model's diagnostic abilities, we will integrate more clinical data, such as patient history and genetic information, and broaden the dataset to include more varied images from different sources in subsequent work. Investigating more complex machine learning approaches, like attention processes and ensemble learning, can further increase accuracy and resilience. Additionally, Practical clinical deployment will require real-world validation and the creation of user-friendly interfaces.

Appendix:

Xception model Code:

```
# Training

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import Xception
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"

# Using adjust, shear range, zoom range, and horizontal flip for training purposes
train_datagen = ImageDataGenerator(
  rescale=1.0 / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.2  # 80-20 split
)

# Only adjust for data validation, (no enhancement).

validation_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)

# Learning to operate Generators
train_generator = train_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
  subset='training'  # Specify this is for training set
)

# Verification of generators
validation_generator = validation_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
  subset='validation'  # Specify this is for validation set
)

num_classes = train_generator.num_classes
num_classes

from tensorflow.keras.applications import Xception
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
# Downloads Xception, as a method, excluding the top layer.
base_model = Xception(weights='imagenet', include_top=False)
# Adds top layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)  # Additional fully-connected layer with 64 units
x = Dense(256, activation='relu')(x)  # Additional fully-connected layer with 256 units
predictions = Dense(num_classes, activation='softmax')(x)
# Generate entire models.
model = Model(inputs=base_model.input, outputs=predictions)

# Built the Xception_model once more (make sure to after adding changes).
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Refine the model design.
history = model.fit(
train_generator,
steps_per_epoch=train_generator.n // batch,
   epochs=16,
validation_data=validation_generator,
validation_steps=validation_generator.n // batch
)

model.save_weights(r"/content/drive/MyDrive/model_weights/Xception_Latest_v1.h5")

# Plot the training and the reliability of validation statistics.
plt.figure(figsize=(12, 4) ,dpi=300)

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot the training and validation error values.
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()

import seaborn as sns
# Plot the training data and error validation metrics.

validation_generator.reset()
```

```
Y_pred = model.predict(validation_generator, validation_generator.n // batch + 1)
y_pred = np.argmax(Y_pred, axis=1)
y_true = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

# Makes a confusion design  matrix.
conf_matrix = confusion_matrix(y_true, y_pred)

# Plots confused matrices.
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d',
xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Print categorisation results.
print('Classification Report')
print(classification_report(y_true, y_pred, target_names=class_labels))

class_indices = train_generator.class_indices
classes = list(class_indices.keys())

images, labels = next(validation_generator)

len(labels)

images[0].shape

num_images = 10

images, labels = next(validation_generator)
images = images[:num_images]
labels = labels[:num_images]

def get_predictions(generator, model, num_images):
   images, labels = next(generator)
   images = images[:num_images]
   labels = labels[:num_images]
   predictions = model.predict(images)
actual_classes = [classes[np.argmax(label)] for label in labels]
predicted_classes = [classes[np.argmax(prediction)] for prediction in predictions]
   return images, actual_classes, predicted_classes

num_images = 10
images, actual_classes, predicted_classes = get_predictions(train_generator, model, num_images)

# Plot of the pictures with the current and expected class labels.

plt.figure(figsize=(14,6))
for i in range(num_images):
plt.subplot(2, 5, i + 1)
plt.imshow(images[i])
plt.title(f'Actual: {actual_classes[i]}\nPredicted: {predicted_classes[i]}')
plt.axis('off')
plt.show()
```

# Xception Model Evaluation

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Imports the Xception_model from TensorFlow Kiras
from tensorflow.keras.applications import Xception
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"

#Using adjust, shear range, zoom range, and horizontal_flip to enrich the data used for
training.
train_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.2  # 80-20 split
)

# Only resizing for data validation, no enhancement.
validation_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)

# Traininng generators
train_generator = train_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='training'
)

# Validity of generators
validation_generator = validation_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='validation'
)

num_classes = train_generator.num_classes
num_classes

from tensorflow.keras.models import load_model

path = r"/content/drive/MyDrive/model_weights/Xception_Latest_v1.h5"
model = load_model(path)

model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

class_indices = train_generator.class_indices
classes = list(class_indices.keys())

images, labels = next(validation_generator)

len(labels)

images[0].shape

num_images =10
images, labels = next(validation_generator)
images = images[:num_images]
labels = labels[:num_images]

def get_predictions(generator, model, num_images):
   images, labels = next(generator)
   images = images[:num_images]
   labels = labels[:num_images]
   predictions = model.predict(images)
actual_classes = [classes[np.argmax(label)] for label in labels]
predicted_classes = [classes[np.argmax(prediction)] for prediction in predictions]
   return images, actual_classes, predicted_classes

num_images = 10
images, actual_classes, predicted_classes = get_predictions(train_generator, model,
num_images)

# Plots a picture with the real and expected class names.
plt.figure(figsize=(20, 10))
for i in range(num_images):
plt.subplot(2, 5, i + 1)
plt.imshow(images[i])
plt.title(f'Actual: {actual_classes[i]}\nPredicted: {predicted_classes[i]}')
plt.axis('off')
plt.show()

images, labels = next(validation_generator)

images= images[1,:,:,:]

images.shape

import tensorflow as tf
import cv2
import numpy as np

# Loads the photos
path = r"/content/drive/MyDrive/Augmentation1/Augmentation/pituitary/Te-piTr_0005.jpg"
img = cv2.imread(path)
img = cv2.resize(img, (224, 224))

# Transforms the images into empty arrays and normalises them.
img = np.array(img) / 255.0

# Grows the dimensions to fit the input shape requested by the model (batch, height, breadth,
channels).
img = np.expand_dims(img, axis=0)

predictions = model.predict(img)

predicted_label = np.argmax(predictions, axis=-1)
predicted_index = np.argmax(predictions, axis=-1)[0]
# Obtain class numbers from the generator.
class_indices = validation_generator.class_indices

class_names = {v: k for k, v in class_indices.items()}

# Find the projected class name.
predicted_class = class_names[predicted_index]

print(f'Predicted Label Index: {predicted_index}')
print(f'Predicted Class Name: {predicted_class}')
```

Resnet101 model code:

```
# Training

!pip install resnet
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet101  # Import ResNet101 model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

img_w, img_h = 224, 224  # Image dimension expected by ResNet50
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"

# Implementing rescales, shear_range, zoom range, and horizontal_flip to enhance data used
for training
train_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.2  # 80-20 split
)

# Just rescale data from validation (no additions).
validation_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)

# Trainings generators
train_generator = train_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size =batch,
class_mode='categorical',
   subset='training'
)

# Validations generators
validation_generator = validation_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size =batch,
class_mode='categorical',
   subset='validation'
)

num_classes = train_generator.num_classes
num_classes

from tensorflow.keras.applications import ResNet101
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
# Loads the ResNet101 model, excluding the tops layer
base_model = ResNet101(weights='imagenet', include_top=False)
```

```
# Adds tops layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

# Build the whole model.
model = Model(inputs=base_model.input, outputs=predictions)

# Gather the models once (make sure to after adding changes).
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Fine-tune the models
history = model.fit(
train_generator,
steps_per_epoch=train_generator.n // batch ,
   epochs=16,
validation_data=validation_generator,
validation_steps=validation_generator.n // batch
)

model.save_weights(r"/content/drive/MyDrive/model_weights/Resnet101_Latest.h5")

# Plot the training and validation reliability values.
plt.figure(figsize=(12, 4) ,dpi=300)

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plots training and validation loss values.
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()

import seaborn as sns
# Obtains accurate label and forecast.
validation_generator.reset()
```

```
Y_pred = model.predict(validation_generator, validation_generator.n // batch + 1)
y_pred = np.argmax(Y_pred, axis=1)
y_true = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

# Makes a confusion the matrix approach.
conf_matrix = confusion_matrix(y_true, y_pred)

#Plots uncertainty matrix.
 plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d',
xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Prints the category report.
print('Classification Report')
print(classification_report(y_true, y_pred, target_names=class_labels))

class_indices = train_generator.class_indices
classes = list(class_indices.keys())

images, labels = next(validation_generator)

len(labels)

images[0].shape

def get_predictions(generator, model, num_images):
    images, labels = next(generator)
    images = images[:num_images]
    labels = labels[:num_images]
    predictions = model.predict(images)
actual_classes = [classes[np.argmax(label)] for label in labels]
predicted_classes = [classes[np.argmax(prediction)] for prediction in predictions]
    return images, actual_classes, predicted_classes

num_images = 10
images, actual_classes, predicted_classes = get_predictions(train_generator, model,
num_images)
# Plots the image with the observed and expected class names.

plt.figure(figsize=(20, 10))
for i in range(num_images):
plt.subplot(2, 5, i + 1)
plt.imshow(images[i])
plt.title(f'Actual: {actual_classes[i]}\nPredicted: {predicted_classes[i]}')
plt.axis('off')
plt.show()
```

# Resnet101 Model evaluation

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet101  # Import ResNet101 model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

img_w, img_h = 224, 224  # Image dimensions expected by ResNet50
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"

# Using changes, shear_range, zoom_range, and horizontal_flip to enhance training data
train_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.2  # 80-20 split
)

# Only rescales for validations data (no augmentations)
validation_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)

# Trainings generators
train_generator = train_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='training'  # Specifys this is for trainings set
)

# Validations generators
validation_generator = validation_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
    subset='validation'  # Specify this is for validation set
)

num_classes = train_generator.num_classes
num_classes

from tensorflow.keras.models import load_model

path = r"/content/drive/MyDrive/model_weights/Resnet101_Latest.h5"
model = load_model(path)

# Recompile the models (always after doing changes).
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

class_indices = train_generator.class_indices
classes = list(class_indices.keys())

images, labels = next(validation_generator)

len(labels)

images[0].shape

num_images =10
images, labels = next(validation_generator)
images = images[:num_images]
labels = labels[:num_images]

def get_predictions(generator, model, num_images):
    images, labels = next(generator)
    images = images[:num_images]
    labels = labels[:num_images]
    predictions = model.predict(images)
actual_classes = [classes[np.argmax(label)] for label in labels]
predicted_classes = [classes[np.argmax(prediction)] for prediction in predictions]
    return images, actual_classes, predicted_classes

num_images = 10
images, actual_classes, predicted_classes = get_predictions(train_generator, model,
num_images)

# Plots the image with the actually and expected class names.
plt.figure(figsize=(20, 10))
for i in range(num_images):
plt.subplot(2, 5, i + 1)
plt.imshow(images[i])
plt.title(f'Actual: {actual_classes[i]}\nPredicted: {predicted_classes[i]}')
plt.axis('off')
plt.show()
```

```
images, labels = next(validation_generator)

images= images[1,:,:,:]

images.shape

import tensorflow as tf
import cv2
import numpy as np

# Loads the images
path = r"/content/drive/MyDrive/Augmentation1/Augmentation/glioma/Te-glTr_0000.jpg"
img = cv2.imread(path)
img = cv2.resize(img, (224, 224))

# Converts the pictures into the numpy file arrays and normalises them.

img = np.array(img) / 255.0

# Expands the dimensions to match the given input shape anticipated by the model (batch,
height, breadth, channels).
img = np.expand_dims(img, axis=0)

predictions = model.predict(img)

predicted_label = np.argmax(predictions, axis=-1)
predicted_index = np.argmax(predictions, axis=-1)[0]
# Obtains class indices from the generators
class_indices = validation_generator.class_indices

class_names = {v: k for k, v in class_indices.items()}

# Find the expected class name.
predicted_class = class_names[predicted_index]

print('Predicted Label Index: {predicted_index}')
print('Predicted Class Name: {predicted_class}')
```

EfficientNetB3 Model Code:

```
# training

!pip install efficientnet
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from efficientnet.tfkeras import EfficientNetB3  # Import EfficientNetB3 model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"
dataset_path

# Employing changes, shear_range, zoom_ranges, and horizontal_flip to augment training
data.
train_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.2  # 80-20 split
)

# Only changes for data validation, no modifications.
validation_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)

# for trainings
train_generator = train_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='training'  # Specifys this is for trainings set
)

# for validations
validation_generator = validation_datagen.flow_from_directory(
dataset_path,
target_size=(imge_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='validation'  # Specifys this is for validations set
)

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"
dataset_path

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"
dataset_path

img_w, img_h = 224, 224
```

```
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"
dataset_path

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"
dataset_path

num_classes = train_generator.num_classes
num_classes

from tensorflow.keras.layers import Dense, Dropout
base_model = EfficientNetB3(weights='imagenet', include_top=False)

# Add tops layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)a

# Creates models
model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Fine-tunes
history = model.fit(
train_generator,
steps_per_epoch=train_generator.n // batch,
   epochs=16,
validation_data=validation_generator,
validation_steps=validation_generator.n // batch
)

model.save_weights(r"/content/drive/MyDrive/model_weights/efficientnetb3_Latest.h5")

# Display figures and graphs for training as well as validation accuracy values.
plt.figure(figsize=(12, 4) ,dpi=300)

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Displays graphs for validation and training loss values.
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()

import seaborn as sns
# Gets the correct label and forecast
validation_generator.reset()
Y_pred = model.predict(validation_generator, validation_generator.n // batch + 1)
y_pred = np.argmax(Y_pred, axis=1)
y_true = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

# Creates confusions matrix
conf_matrix = confusion_matrix(y_true, y_pred)

# Plots confusions matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d',
xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

print('Classification Report')
print(classification_report(y_true, y_pred, target_names=class_labels))

class_indices = train_generator.class_indices
classes = list(class_indices.keys())

images, labels = next(validation_generator)
len(labels)
images[0].shape

images, labels = next(validation_generator)
images = images[:num_images]
labels = labels[:num_images]

def get_predictions(generator, model, num_images):
   images, labels = next(generator)
   images = images[:num_images]
   labels = labels[:num_images]
   predictions = model.predict(images)
actual_classes = [classes[np.argmax(label)] for label in labels]
```

```
predicted_classes = [classes[np.argmax(prediction)] for prediction in predictions]
   return images, actual_classes, predicted_classes

num_images = 10
images, actual_classes, predicted_classes = get_predictions(train_generator, model,
num_images)

# displays the image with their real and expected class names.
plt.figure(figsize=(20, 10))
for i in range(num_images):
plt.subplot(2, 5, i + 1)
plt.imshow(images[i])
plt.title(f'Actual: {actual_classes[i]}\nPredicted: {predicted_classes[i]}')
plt.axis('off')
plt.show()
```

# EfficientNetB3 Evaluation

```
!pip install efficientnet
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from efficientnet.tfkeras import EfficientNetB3
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')

img_w, img_h = 224, 224
batch = 32
dataset_path = "/content/drive/MyDrive/Augmentation1/Augmentation"

# Applying alters, shear_ranges, zoom_ranges, and horizontal_flip to augment data used for
training.
train_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.2  # 80-20 split
)

# Only changes data for validation (no augmentations).
```

```
validation_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.2)

# trainings Trainings
train_generator = train_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='training'  # Specify this is for training set
)

#for  Validations
validation_generator = validation_datagen.flow_from_directory(
dataset_path,
target_size=(img_w, img_h),
batch_size=batch,
class_mode='categorical',
   subset='validation'  # Specify this is for validation set
)

num_classes = train_generator.num_classes
num_classes

from tensorflow.keras.models import load_model

path = r"/content/drive/MyDrive/model_weights/efficientnetb3_Latest.h5"
model = load_model(path)

model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

class_indices = train_generator.class_indices
classes = list(class_indices.keys())

images, labels = next(validation_generator)

len(labels)
images[0].shape

num_images =10
images, labels = next(validation_generator)
images = images[:num_images]
labels = labels[:num_images]

def get_predictions(generator, model, num_images):
   images, labels = next(generator)
   images = images[:num_images]
   labels = labels[:num_images]
   predictions = model.predict(images)
actual_classes = [classes[np.argmax(label)] for label in labels]
predicted_classes = [classes[np.argmax(prediction)] for prediction in predictions]
```

```
    return images, actual_classes, predicted_classes

num_images = 10
images, actual_classes, predicted_classes = get_predictions(train_generator, model,
num_images)

# displays the image with their real and expected class names.
plt.figure(figsize=(20, 10))
for i in range(num_images):
plt.subplot(2, 5, i + 1)
plt.imshow(images[i])
plt.title(f'Actual: {actual_classes[i]}\nPredicted: {predicted_classes[i]}')
plt.axis('off')
plt.show()

images, labels = next(validation_generator)
images= images[1,:,:,:]
images.shape

import tensorflow as tf
import cv2
import numpy as np

# Loads the images
path =
r"/content/drive/MyDrive/Augmentation1/Augmentation/meningioma/Te-meTr_0004.jpg"
img = cv2.imread(path)
img = cv2.resize(img, (224, 224))

# Converts images to numpy allows you arrays and normalises them.
img = np.array(img) / 255.0
img = np.expand_dims(img, axis=0)

predictions = model.predict(img)

# Assuming that the model's output is a distribution of probability over classes.
predicted_label = np.argmax(predictions, axis=-1)
predicted_index = np.argmax(predictions, axis=-1)[0]
# Get class indice from the generators
class_indices = validation_generator.class_indices

# Invert the class_indicesdictionarys to map indice to class name
class_names = {v: k for k, v in class_indices.items()}

# Get the predicted class name
predicted_class = class_names[predicted_index]

print('Predicted Label Index: {predicted_index}')
print('Predicted Class Name: {predicted_class}')
```