BRAIN TUMOR DETECTION USING DEEP LEARNING

project work phase-II report submitted in partial fulfillment of the requirement for award of the degree of

Master of Science
in
Data Analytics

By

THANGARAJ B (22PHCD0021) **(VTP3554)**

Under the guidance of
Dr. M. S. ARUNKUMAR, M.E., Ph.D.,
ASSOCIATE PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled "BRAIN TUMOR DETECTION USING DEEP LEARNING" by THANGARAJ B & 22PHCD0021 has been carried out under my supervision and this work has not been submitted else where for a degree.

Signature of Supervisor	**Signature of Professor In-charge**
Computer Science & Engineering	**Computer Science & Engineering**
School of Computing	**School of Computing**
Vel Tech Rangarajan Dr. Sagunthala R&D	**Vel Tech Rangarajan Dr. Sagunthala R&D**
Institute of Science & Technology	**Institute of Science & Technology**
May, 2024	**May, 2024**

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

THANGARAJ B

Date: / /

APPROVAL SHEET

This project report entitled BRAIN TUMOR DETECTION USING DEEP LEARNING by THANGARAJ B (22PHCD0021) is approved for the degree of M.Sc in Data Analytics.

Examiners **Supervisor**

Dr. M. S. ARUNKUMAR, M.E., Ph.D.,

Date: / /

Place:

ACKNOWLEDGEMENT

I express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

I am very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN,** for providing us with an environment to complete our project successfully.

I record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.,** for immense care and encouragement towards us throughout this project.

I am thankful to our **Head, Department of Computer Science & Engineering,**
Dr. M. S. MURALI DHAR, M.E., Ph.D., for providing immense support in all our endeavors.

I also take this opportunity to express a deep sense of gratitude to our Internal Supervisor
Dr. M. S. ARUNKUMAR, M.E., Ph.D., for his cordial support, valuable information, and guidance, he helped us complete this project through various stages.

A special thanks to our **Project Coordinator Dr. K. CHINNATHAMBI, MCA., Ph.D.,** for his valuable guidance and support throughout the project.

A special gratitude to our **Programme Coordinator Dr. R. ARUNA, M.Tech., Ph.D.,** for her valuable guidance and support throughout the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

THANGARAJ B 22PHCD0021

ABSTRACT

Brain tumors are abnormal growths of cells within the brain or surrounding tissues. Accurate and early detection of brain tumors is crucial for effective treatment and improving patient outcomes. Traditional methods of brain tumor detection, such as manual examination of MRI scans, can be time-consuming, subjective, and prone to human error. In recent years, deep learning techniques, particularly CNN, have shown remarkable success in various medical image analysis tasks, including brain tumor detection and segmentation. I used pre-trained ResNet50 architecture, a state-of-the-art CNN model for image recognition, as the backbone for feature extraction. The dataset comprising annotated MRI scans of patients with and without brain tumors to train and evaluated the model. Brain tumor detection plays a critical role in early diagnosis and treatment planning. In this study, deep learning-based approach for accurate and efficient brain tumor detection using MRI scans. The method leverages CNN to automatically extract meaningful features from MRI images, enabling the model to classify the presence of tumors with high accuracy. The dataset comprising annotated MRI scans of patients with and without brain tumors to train and evaluate our model. I used pre-trained models for training our dataset of brain tumors they are ResNet50 and CNN. Its accuracy will help with brain tumor identification and prevention at early stages before the tumor results in any physical side effects. The proposed approach involves several key steps. Firstly, a dataset of brain MRI scans is collected, comprising images labeled with tumor and non-tumor classes. Preprocessing techniques are applied to enhance the quality of the images, including normalization, resizing, and augmentation to increase the diversity of the training data. Transfer learning is utilized, where the pre-trained ResNet50 model, trained on large-scale image datasets, serves as the initial weights for the network.

Keywords: Deep Learning, Convolutional Neural Network (CNN), ResNet50, Magnetic Resonance Image (MRI).

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

AUC	Area Under Curve
CNN	Convolutional Neural Network
CT	Computed Tomography
DL	Deep Learning
LED	Light Emitting Diode
MRI	Magnetic Resonance Imaging
ReLu	Rectified Linear Unit
ResNet	Residual Network
ROC	Receiver Operating Characteristic
VGG	Visual Geometry Group

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

An abnormal cell growth that has developed in the brain is known as a Brain Tumor. Traditional methods of brain tumor detection, such as visual inspection of medical images by radiologists, can be time-consuming and prone to human error. Early and accurate detection of brain tumors is crucial for effective treatment planning and improving patient outcomes. However, manual analysis of medical imaging data, such as MRI scans, is a time-consuming and subjective process, often prone to human error and variability. However, while not all brain tumors are malignant (cancerous), some are benign (non-cancerous). One of the most widely adopted and successful CNN architectures is ResNet50, introduced by He et al. in 2015. In 2020, it is anticipated that 308,102 individuals will receive a primary brain or spinal cord tumor diagnosis worldwide. Brain tumor detection and classification from medical imaging data is a crucial task with significant implications for patient care and treatment planning. In recent years, deep learning techniques, particularly Convolutional Neural Networks (CNNs), have demonstrated remarkable success in automating this process with high accuracy. Numerous studies have leveraged the powerful feature extraction capabilities of CNNs to analyze MRI scans and differentiate between various types of brain tumors and healthy brain tissues.

Brain tumors pose a significant health concern globally, with early detection being crucial for effective treatment and patient outcomes. Traditional methods of brain tumor detection rely heavily on manual interpretation of medical imaging scans, such as MRI, which can be time-consuming and subjective. In recent years, the emergence of deep learning techniques has revolutionized medical image analysis, offering automated and accurate solutions for detecting various abnormalities, including brain tumors. This introduction presents a comprehensive overview of utilizing deep learning, specifically CNN, and the ResNet50 architecture for brain tumor detection from MRI scans. The utilization of deep learning in this context not only enhances the efficiency and accuracy of tumor detection but also has the potential to assist healthcare professionals in making timely and informed decisions. The rapid advancement of deep learning methodologies, coupled with the availability of large-scale medical imaging datasets, has paved the way for the development of CNN and ResNet50 algorithms capable of analyzing complex patterns within MRI images. CNNs, in particular, have demonstrated remarkable success in image detecction tasks by automatically learning hierarchical representations of features directly from raw pixel data. However, recent advancements in deep learning have shown promise in automating the detection process, leading to more efficient and reliable diagnoses. By training CNNs on annotated MRI datasets containing images of patients both with and without brain tumors, we seek to develop a robust and efficient system capable of automatically identifying tumor regions within brain scans. This paper aims to provide insights into the methodology of utilizing CNNs, specifically ResNet50, for brain tumor detection. It outlines the steps involved, including data collection, preprocessing, model architecture selection, training, and evaluation. Additionally, it discusses the potential impact of such automated systems on clinical practice, including the acceleration of diagnosis, improved treatment planning, and ultimately better patient outcomes.

1.2 Aim of the Project

The aim of the project "Brain tumor detection using deep learning" is to develop a robust and accurate system capable of automatically the presence of brain tumors in medical images, particularly MRI scans, using deep learning techniques.

1.3 Project Domain

Convolutional neural networks, also known as CNNs represent a class of neural networks that excel at handling input having a grid-like layout, such as photos. A digital image is a representation of binary visual data. It has several pixels that are organized in a grid-like pattern. Convolutional, pooling and fully connected layers make up the conventional architecture of a CNN. CNN architecture mainly reLu activation function mainly used for middle layer. Sigmoid and Softmax activation function are used in output layer. ResNet50, short for Residual Network with 50 layers, ResNet is a type of CNN that addresses the vanishing gradient problem faced by very deep neural networks during training. ResNet50, in particular, is a 50-layer deep residual network that has been pre-trained on a large dataset of natural images, such as ImageNet. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer).

1.4 Scope of the Project

The scope of the Brain Tumor detection using CNN and ResNet50 includes the following:

- CNN with the ResNet50 architecture specifically for brain tumor detection using medical imaging data, primarily focusing on MRI scans.
- The scope includes data acquisition from reputable sources, preprocessing to ensure quality and consistency, and annotation for tumor and non-tumor regions.
- Development involves the implementation and fine-tuning of CNN models, particularly ResNet50, utilizing transfer learning techniques to adapt pre-trained models to the task at hand
- Training and optimization are carried out to maximize the model's performance, with evaluation metrics such as accuracy, sensitivity, specificity utilized for assessment
- Ensuring patient privacy and confidentiality throughout the data collection and analysis process. Addressing biases in the dataset and model predictions to mitigate potential disparities in healthcare delivery.

Thus, the scope of the project is to used to CNN and ResNet50 algorithms. Find to the patients with and without brain tumors. And compare to accuracy of CNN and ResNet50.

Chapter 2

LITERATURE REVIEW

[1] Rehman et al., (2020) proposed a deep learning model based on a modified VGG-16 architecture for brain tumor classification, achieving an accuracy of 98.7 on their dataset . Their model was based on a modified VGG-16 architecture, a well-known CNN architecture commonly used for image classification tasks. The VGG-16 architecture consists of 16 layers, including convolutional layers for feature extraction and fully connected layers for classification. Likely made modifications to this architecture to adapt it to the specific requirements of brain tumor classification, which often involves distinguishing between different types of brain tumors based on MRI or CT scans. In addition to accuracy, other metrics such as sensitivity, specificity, and area under the ROC curve AUC are commonly used to assess the performance of medical image classification models.

[2] Pereira et al., (2018) developed a CNN model using transfer learning with the Inception-V3 network, yielding an accuracy of 97.5 in distinguishing between tumor and non-tumor cases. made a significant contribution to medical image analysis with their development of a CNN model using transfer learning with the Inception-V3 network. Transfer learning involves leveraging pre-trained models trained on large datasets and fine-tuning them for specific tasks with smaller datasets, which is particularly useful when dealing with limited medical imaging data. Achieving an accuracy of 97.5 in distinguishing between tumor and non-tumor cases is a notable accomplishment. However, it's essential to consider other performance metrics such as sensitivity, specificity, and area under the ROC curve AUC to gain a comprehensive understanding of the model's performance.

[3] Salehi et al., (2020) explored the use of ensemble learning techniques, combining multiple CNN models to improve brain tumor segmentation performance, with their approach outperforming individual models. made a notable contribution to the field of medical image segmentation, specifically focusing on brain tumor segmentation, a crucial task in medical image analysis for treatment planning and monitoring.

A comparison of these proposed architectures with the baseline reference ones shows very interesting results. These techniques aim to improve segmentation accuracy by reducing errors and increasing robustness to variations in the data.

[4] Lao et al., (2020) proposed a multimodal deep learning framework that combined MRI data with gene expression profiles, achieving improved performance in glioma classification compared to single-modality approaches . The use of MRI data in glioma classification is well-established, as MRI provides detailed anatomical information about the brain and allows for the visualization of tumor morphology, location, and extent. The deep learning architecture used by Lao et al. might have been a convolutional neural network (CNN) or a similar model capable of handling multimodal data. This study comprised a discovery data set of 75 patients and an independent validation data set of 37 patients.

[5] Akkus et al., (2017) The success of deep learning in brain tumor detection and classification has opened up new avenues for automated and more accurate diagnosis, potentially leading to earlier intervention and improved patient outcomes. However, challenges remain, such as the need for larger and more diverse datasets, the interpretability of deep learning models, and the integration of domain knowledge into these systems. Deep learning-based segmentation approaches for brain MRI are gaining interest due to their self-learning and generalization ability over large amounts of data. As the deep learning architectures are becoming more mature, they gradually outperform previous state-of-the-art classical machine learning algorithms.

[6] Havaei et al., (2017) proposed a cascaded convolutional neural network architecture that achieved state-of-the-art results in brain tumor segmentation, outperforming previous methods on the BRATS 2013 and 2015 challenges. Their contribution lies in proposing a novel cascaded CNN architecture that achieved state-of-the-art results in brain tumor segmentation, surpassing previous methods on benchmark datasets such as BRATS 2013 and 2015 challenges. In their study, Havaei et al. likely employed techniques such as data preprocessing, augmentation, and cross-validation to train and evaluate their cascaded CNN architecture. They may have also incorporated advanced training strategies, such as transfer learning or ensembling, to further enhance segmentation performance.

[7] Zhao et al., (2018) proposed a deep learning model integrating FCNNs and CRFs for brain tumor segmentation," published in the journal Medical Image Analysis in 2018, presents a method for segmenting brain tumors in medical images using a combination of deep learning techniques.The title suggests that the paper proposes a model that integrates Fully Convolutional Neural Networks (FCNNs) and Conditional Random Fields (CRFs) for the purpose of brain tumor segmentation. Segmentation involves identifying and delineating regions of interest (brain tumors) in medical images. The paper utilizes FCNNs, which are a type of neural network well-suited for image segmentation tasks. FCNNs process input images through multiple layers of convolution and pooling operations to produce pixel-wise predictions.

[8] Myronenko et al., (2019) "3D MRI brain tumor segmentation using autoencoder regularization," by Myronenko, was presented at the International MICCAI Brainlesion Workshop in 2019. The focus of the paper is on segmenting brain tumors in 3D MRI (Magnetic Resonance Imaging) scans using a deep learning approach with autoencoder regularization. The title indicates that the paper proposes a method for segmenting brain tumors in 3D MRI scans using autoencoder regularization. Autoencoders are a type of neural network architecture commonly used for unsupervised learning tasks, including feature extraction and data compression. Regularization techniques help prevent overfitting and improve generalization performance.

[9] Shen et al., (2017) "Deep learning in medical image analysis," authored by Shen, Wu, and Suk and published in the Annual Review of Biomedical Engineering in 2017, provides an overview of the application of deep learning techniques in the field of medical image analysis. The title suggests that the paper aims to review and summarize the advancements, methodologies, and applications of deep learning specifically in the context of medical image analysis. This includes methods, challenges, and potential future directions.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The existing systems for Brain Tumor detection, systems specifically focus on CNN and ResNet50 two deep learning models are used him. Then find to patients with and without Tumors. Some of the existing systems are:
CNN: Convolutional neural networks, also known as CNNs or Convent, represent a class of neural networks that excel at handling input having a grid-like layout, such as photos. A digital image is a representation of binary visual data. It has several pixels that are organized in a grid-like pattern and are each given a value to specify how bright and what hue they should be. Convolutional, pooling, and fully connected layers make up the conventional architecture of a CNN. The CNN's fundamental building block is the convolution layer. CNN is Employed in computer vision and image recognition. The term convolutional refers to a mathematical function that is created by integrating two different functions.
ResNet50: ResNet50, short for Residual Network with 50 layers, is a specific variant of the ResNet architecture, which was introduced by researchers at Microsoft Research in 2015. ResNet is a type of CNN that addresses the vanishing gradient problem faced by very deep neural networks during training. ResNet50, in particular, is a 50-layer deep residual network that has been pre-trained on a large dataset of natural images, such as ImageNet.

3.1.1 Disadvantage

The existing Brain Tumor detection using Deep Learning has several disadvantages, including:

- Data Availability and Quality: Deep learning models require large amounts of labeled data for training, which may be scarce, especially for rare tumor types or specific patient demographics. Additionally, labeled data must be of high quality, accurately annotated by experts. Obtaining such data can be costly and time-consuming.

- Limited Training Data: Deep learning models require large amounts of labeled data to learn the subtle differences between different tumor types and healthy tissues. However, obtaining sufficiently diverse and well-annotated data for all possible tumor variations is challenging. As a result, deep learning models may struggle to generalize effectively across various tumor types, leading to reduced specificity.
- Generalization to New Data: Deep learning models may struggle to generalize well to data from different sources or populations than those used during training. This limitation can lead to reduced performance or unexpected behavior when applied to real-world clinical settings with diverse patient populations or imaging protocols.
- Inference Time: Once trained, deep learning models need to process new brain imaging scans to detect tumors. The inference time, or the time it takes for the model to make predictions on a single image, can vary depending on factors such as the model's complexity, input image resolution, and hardware used for inference.

3.2 Proposed System

The existing systems for Brain Tumor detection, systems specifically focus on CNN and ResNet50 two deep learning models are used him. Then find to patients with and without Tumors. Then, this two models compare to CNN and ResNet50. And finally, accuracy compare to CNN and ResNet50 and then, which deep learning models are best to find patients tumor and without tumor.

3.2.1 Advantages

- High Accuracy: Deep learning models can achieve high levels of accuracy in detecting brain tumors from medical imaging data such as MRI or CT scans. Through the extraction of intricate patterns and features from images, deep learning algorithms can identify subtle abnormalities indicative of tumors with remarkable precision.

- Automation and Efficiency: Deep learning-based systems enable automation of the tumor detection process, reducing the need for manual inspection of medical images by radiologists. This automation can significantly increase the efficiency of diagnosis and streamline the workflow in healthcare settings, allowing clinicians to focus their time and expertise on more complex tasks.
- Early Detection and Diagnosis: Early detection of brain tumors is crucial for timely intervention and improved patient outcomes. Deep learning algorithms have the potential to detect tumors at an early stage, even when they are small or located in challenging anatomical regions. Early diagnosis facilitated by deep learning can lead to prompt treatment initiation and better prognosis for patients.
- Cost-effectiveness: Deep learning-based brain tumor detection has the potential to reduce healthcare costs by optimizing resource utilization and minimizing unnecessary interventions. By automating aspects of the diagnostic process, such as image analysis and triaging of cases, deep learning models can help prioritize resource allocation, reduce redundant testing, and optimize healthcare delivery, leading to cost savings for healthcare systems and patients alike.
- Real-time Decision Support: Deep learning models can provide real-time decision support to healthcare providers during image interpretation. By rapidly analyzing medical images and highlighting regions of interest indicative of tumors, these models empower clinicians to make timely and informed decisions about patient care, potentially leading to expedited treatment initiation and improved outcomes.
- Scalability: Deep learning models have demonstrated scalability and the ability to generalize well to diverse patient populations and imaging modalities. Once trained on representative data, these models can be applied across different healthcare institutions and geographic regions, providing consistent and reliable tumor detection capabilities.

3.3 Feasibility Study

A feasibility study assesses the practicality and viability of a proposed project or system. It evaluates technical, economic, and operational factors to determine if the project is feasible and beneficial. The study aims to provide insights into whether implementing the project is achievable and advantageous. Here is the feasibility study for the proposed project, Brain Tumor Detection using Deep Learning models are CNN and ResNet50.

3.3.1 Economic Feasibility

The proposed system requires the use of advanced technologies such as deep learning, CNN and ResNet50 models, Python. These technologies are readily available and widely used in the industry. Therefore, the proposed system is technically feasible. By streamlining diagnostic processes, deep learning-based systems may help reduce healthcare costs associated with unnecessary tests, delays in treatment, and resource inefficiencies.

3.3.2 Technical Feasibility

The cost of developing the proposed system includes the cost of hardware, software, and personel. The hardware required includes a computer system with high processing power and memory. The software required includes Python and deep learning models are CNN and ResNet50. In CNN and ResNet50 deep learning models are find to patients with and without tumors. Therefore, the proposed system is economically feasible.

3.3.3 Social Feasibility

The proposed system is easy to use and user-friendly. The Brain Tumor detection using Deep Learning models mostly used for CNN models and ResNet50 models CNN compared to accuracy are high. The Brain Tumor Detection using Deep Learning models find to accuracy, patients with and without tumors. Therefore, the proposed system is operationally feasible.

3.4 System Specification

3.4.1 Hardware Specification

- Processor: intel Core i5 or Higher.
- Monitor: LED Monitor.

- RAM: 8GB or Higher.
- Hard Disk: 1TB or More.
- SSD: 512GB.

3.4.2 Software Specification

- Opearting System: Windows 11 etc.
- Environment: Visual Studio.
- Language: Python.
- Packages : Pandas, tensorflow, Numpy, matplotlib, sklearn, Cv, imutils

3.4.3 Standards and Policies

Visual Studio Code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE.

Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 Architecture Diagram

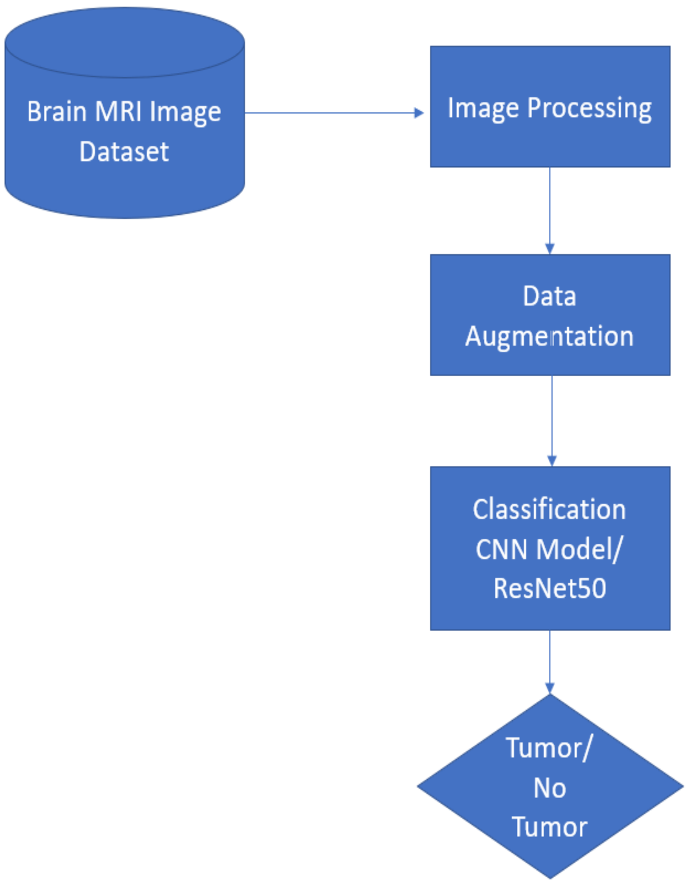

Figure 4.1: **Architecture Diagram**

The figure 4.1 illustrates a process flow for brain tumor classification using MRI images. The process starts with a Brain MRI Image Dataset, which is then subjected to Image Processing. The processed images undergo Data Augmentation, where additional training data is generated through techniques like rotation, flipping, or adding noise. The augmented data is then fed into a Classification CNN Model, specifically ResNet50, which is a deep convolutional neural network architecture commonly used for image classification tasks. The ResNet50 model analyzes the

MRI images and classifies them into two categories ”Tumor” or ”No Tumor.” This binary classification can assist in detecting the presence or absence of tumors in the brain based on the MRI scans.

4.2 Design Phase

4.2.1 Data Flow Diagram

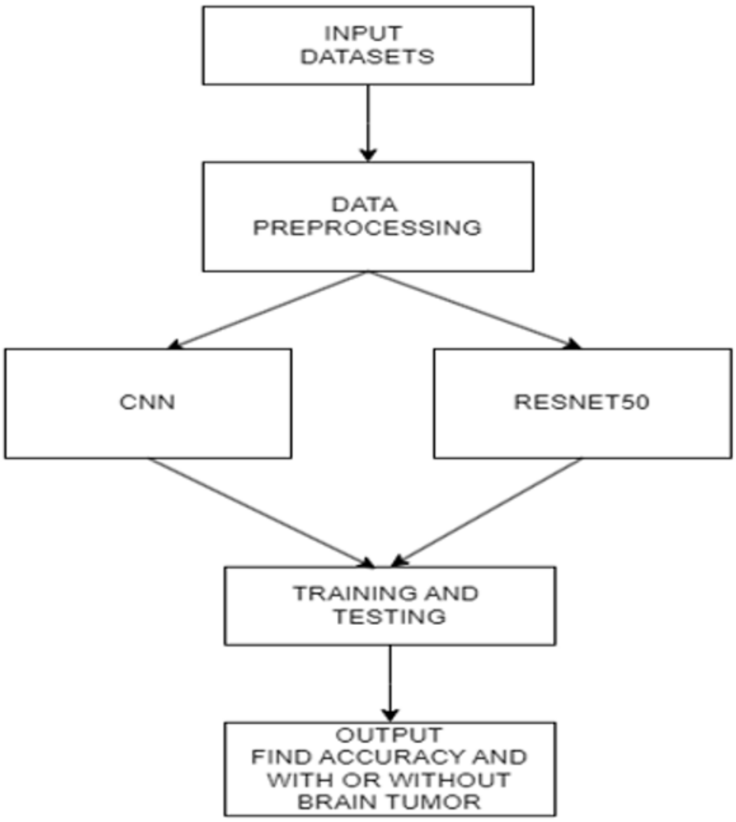

Figure 4.2: **Data Flow Diagram**

The figure 4.2 depicts a process flow for training and evaluating deep learning models to classify brain tumors from input datasets. The process starts with Input Datasets, which likely contain brain imaging data such as MRI scans. The Data Preprocessing step is then applied to the input datasets, which may involve tasks like image resizing, normalization, or data cleaning. After preprocessing, there are two deep learning models CNN and ResNet50. These are two different deep learning architectures that can be used for the task of brain tumor classification. The preprocessed data is fed into the CNN and ResNet50 models for Training and Testing. During this stage, the models learn to recognize patterns in the imaging data associated with the presence or absence of brain tumors. The Output step involves finding the accuracy of the trained models in correctly classifying brain tumors ”With Brain Tumor”or identifying healthy cases ”Without Brain Tumor”. This process flow allows for the comparison and evaluation of different deep learning models (CNN and ResNet50) on the same input datasets for the task of brain tumor classification. The model with higher accuracy can be selected for deployment in tumor detection.

4.3 Algorithm Diagram

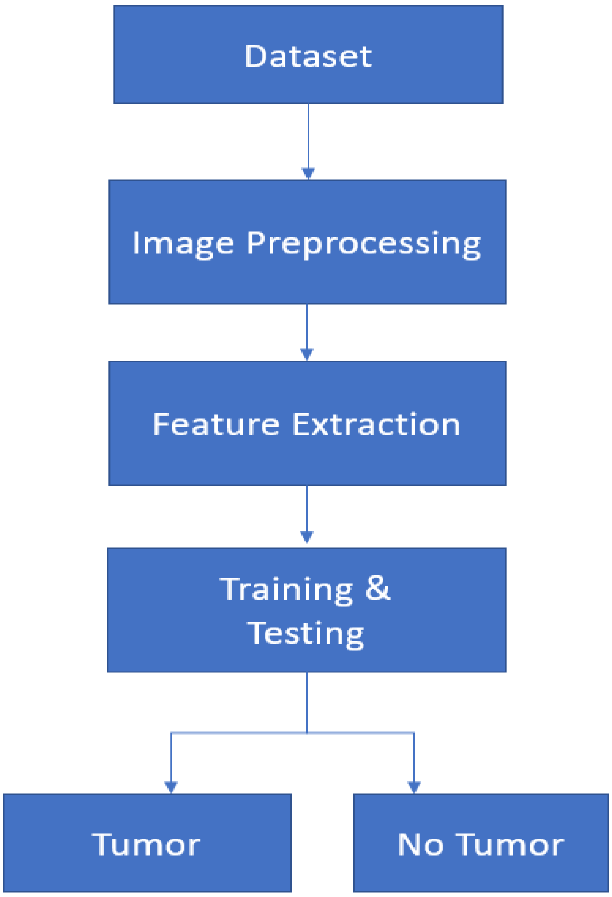

Figure 4.3: **Algorithm Diagram**

This figure 4.3 illustrates a typical process flow for brain tumor detection using image data. The process begins with a dataset containing images, likely brain scans such as MRI images. The images from the dataset undergo preprocessing steps, which may include tasks like resizing, normalization, or noise removal, to prepare the data for further analysis. After preprocessing, relevant features are extracted from the images. This step aims to identify and quantify important patterns or characteristics that can distinguish between images containing tumors and those without tumors. The extracted features are then used to train and test a deep learning model. During training, the model learns to recognize patterns associated with the presence or absence of tumors. The trained model is then evaluated on a test dataset. The final step is the classification output, where the trained model predicts whether an input image contains a tumor or not. The output is typically binary, classifying the image into one of two categories “Tumor” or “No Tumor”.

4.3.1 Algorithms

CNN: Convolutional neural networks, also known as CNNs or Convent, represent a class of neural networks that excel at handling input having a grid-like layout, such as photos. A digital image is a representation of binary visual data. It has several pixels that are organized in a grid-like pattern and are each given a value to specify how bright and what hue they should be. Convolutional, pooling, and fully connected layers make up the conventional architecture of a CNN. The CNN’s fundamental building block is the convolution layer. CNN is Employed in computer vision and image recognition. This diagram illustrates an approach for brain tumor classification using transfer learning with convolutional neural networks (CNNs) on MRI images. The process starts with a dataset of MRI scans, which undergo data augmentations like rotations or flips to increase the training data diversity. The images are then resized to fit the input requirements of the CNN model. The model architecture consists of two main parts: the initial layers from a pre-trained CNN model and the last few replaced layers for the specific classification task. The initial layers of the pre-trained model are kept frozen, leveraging the feature extraction capabilities learned from a large dataset like ImageNet. These layers act as a powerful feature extractor for the MRI images. These replaced layers take the extracted features as input and perform the final classification into output classes like meningioma, pituitary, and glioma (types of brain tumors). The replaced layers are trained on the MRI dataset, while the initial layers from the pre-trained model remain frozen, allowing the model to leverage the previously learned features while adapting to the new task. This transfer learning approach with CNNs enables efficient training on a relatively small MRI dataset by leveraging the knowledge gained from a large pre-trained model, while still allowing customization for the specific brain tumor classification task.

ResNet50: ResNet50, short for Residual Network with 50 layers, is a specific variant of the ResNet architecture, which was introduced by researchers at Microsoft Research in 2015. ResNet is a type of CNN that addresses the vanishing gradient problem faced by very deep neural networks during training. ResNet50, in particular, is a 50-layer deep residual network that has been pre-trained on a large dataset of natural images, such as ImageNet. ResNet-50 is a type of residual network that utilizes skip connections to help mitigate the vanishing gradient problem in very deep neural networks. The architecture consists of multiple stacked residual blocks, each containing convolutional layers, batch normalization, and rectified linear unit (ReLU) activation functions. The skip connections allow the input to bypass some layers and be added to the output of those layers, enabling better flow of gradients during training. The input to the ResNet-50 model is an image, which is processed through these residual blocks, performing various operations like convolutions, pooling, and non-linear activations. The depth of the network (50 layers in this case) allows it to learn increasingly complex features from the input image. ResNet-50 has shown impressive performance on various image classification benchmarks, making it a popular choice for tasks like object recognition, scene understanding, and medical image analysis, including brain tumor classification from MRI scans.

4.4 Module Description

4.4.1 Data Preprocessing Module

This module prepares the input medical imaging data, typically MRI scans, for analysis by the deep learning models. Preprocessing steps may include resizing the images to a standard resolution, normalization to enhance contrast, and noise reduction to improve image quality. Furthermore, data augmentation techniques such as rotation, flipping, and zooming may be applied to augment the training dataset, enhancing the model's ability to generalize to unseen variations in the input images.

4.4.2 Feature Extraction Module

In this module, the CNN and ResNet50 models serve as feature extractors, automatically learning discriminative features from the preprocessed MRI images that are indicative of the presence or absence of brain tumors. CNNs and ResNet50 architectures are well-suited for feature extraction tasks in medical imaging due to their ability to capture spatial hierarchies of features through convolutional layers and residual connections, respectively. .

4.4.3 Model Training Module

This module involves training the CNN and ResNet50 models on labeled training data, where each MRI image is associated with a binary label indicating the presence or absence of a tumor. During training, the models optimize their parameters (e.g., weights and biases) using optimization algorithms like stochastic gradient descent or Adam, minimizing a predefined loss function that quantifies the disparity between predicted and ground-truth labels.

4.4.4 Model Evalution Module

Once trained, the CNN and ResNet50 models are evaluated using a separate validation dataset to assess their performance in detecting brain tumors. Evaluation metrics such as accuracy, precision, recall, and F1-score are computed to quantify the models' ability to correctly classify tumors while minimizing false positives and false negatives.

4.4.5 Integration Module

Finally, the detected tumor regions and associated diagnostic information are integrated into a cohesive output format suitable for clinical interpretation and integration into existing healthcare systems. Integration may involve generating structured reports summarizing the detected abnormalities, providing visual overlays on the original MRI images highlighting tumor regions.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design of Brain Tumor

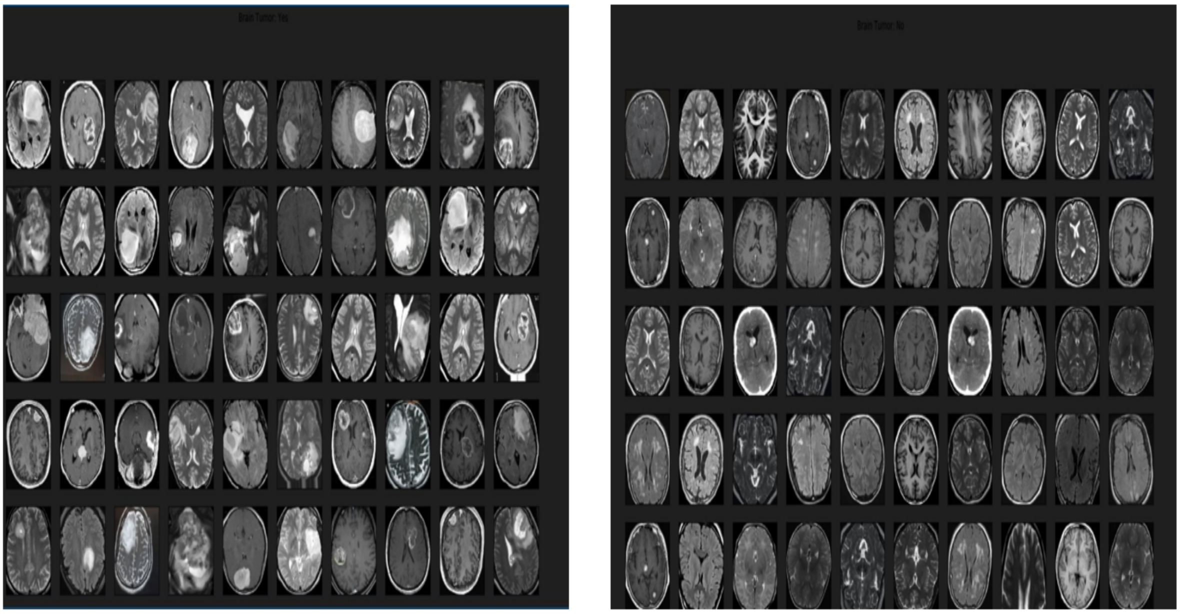

Figure 5.1: **Dataset**

The figure 5.1 images appear to be magnetic resonance imaging (MRI) scans of the human brain. MRI scans use strong magnetic fields and radio waves to generate detailed images of the brain's internal structures.

5.1.2 Output Design of Confusion Matrix

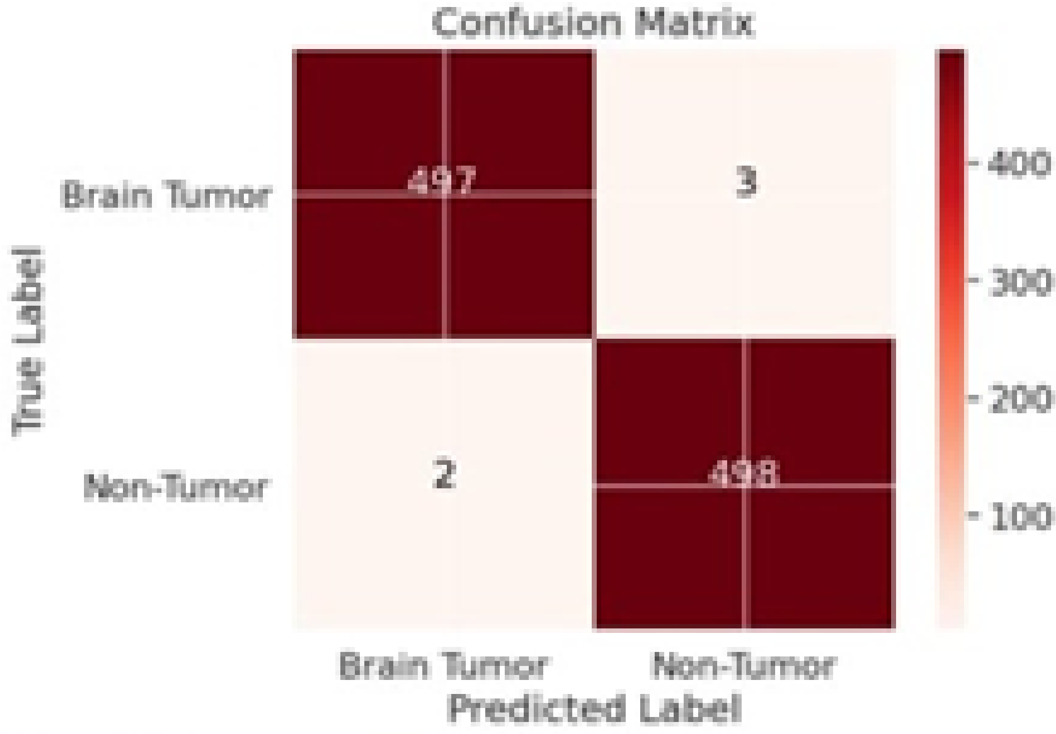

Figure 5.2: **Confusion Matrix of CNN**

The figure 5.2 image appears to be a confusion matrix, which is a tool used to evaluate the performance of a classification model, typically in machine learning or statistics. A confusion matrix is a table that shows the actual and predicted classifications made by the model. It allows for the visualization of the model's performance by comparing the true labels (actual classes) with the predicted labels (classes predicted by the model). In this particular confusion matrix, the rows represent the actual classes, and the columns represent the predicted classes. The matrix has two classes are Brain Tumor,Non-Tumor.

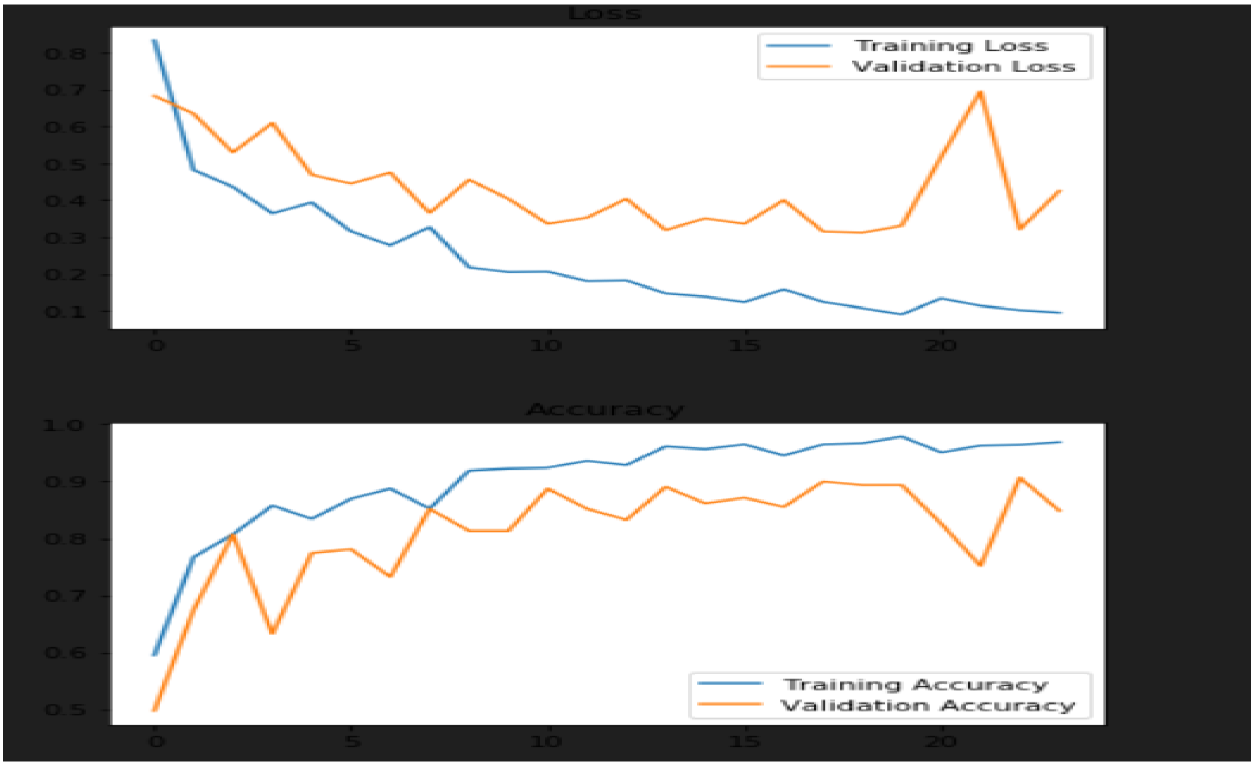

Figure 5.3: **Accuracy and Loss**

The figure 5.3 image shows two line graphs that are commonly used to visualize the training process of a machine learning model, specifically a neural network or deep learning model. The top graph displays the training loss and validation loss over a number of training iterations or epochs. The training loss (blue line) represents the error of the model on the training data, while the validation loss (orange line) represents the error on a separate validation dataset. In an ideal scenario, both the training and validation losses should decrease over time as the model learns from the data. However, if the validation loss starts increasing while the training loss continues to decrease, it indicates that the model is overfitting to the training data and failing to generalize well to new, unseen data.

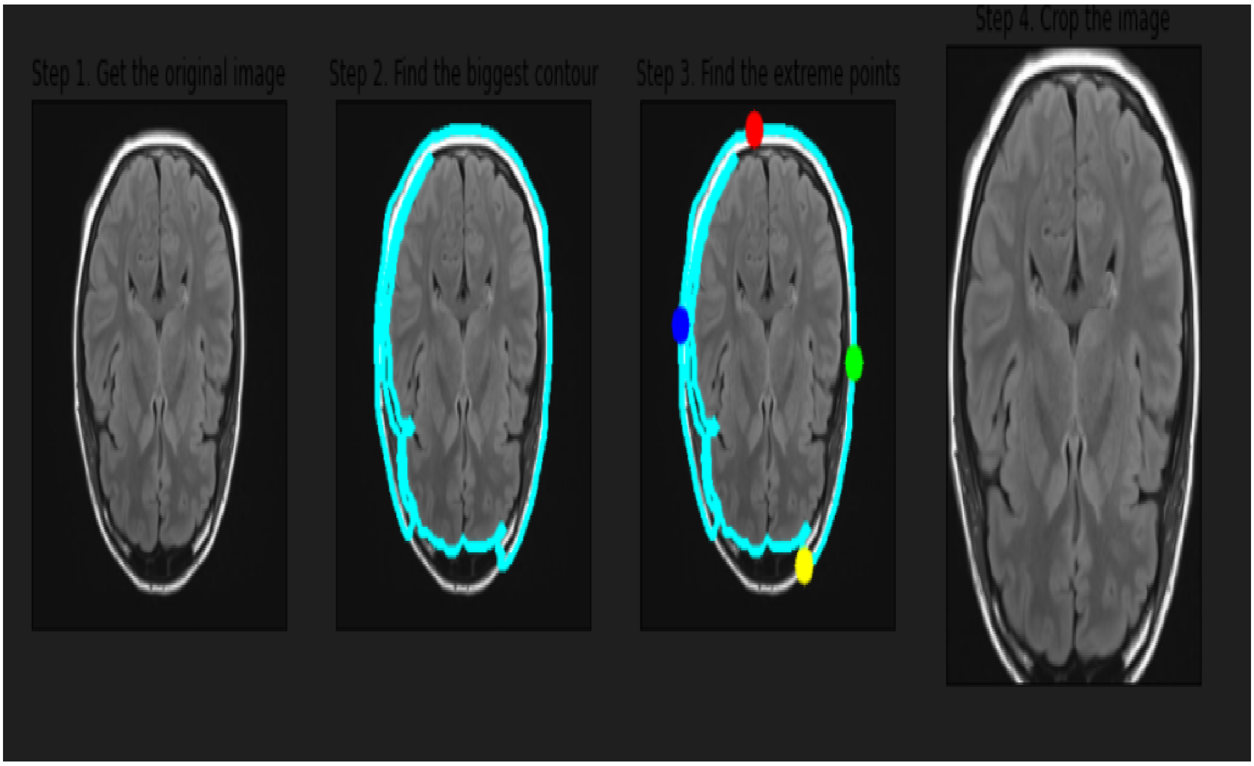

Figure 5.4: **ResNet50 output Images**

The figure 5.4 image shows the steps involved in processing a brain MRI scan using computer vision techniques.

Get the original image.

- The first image shows the original brain MRI scan in grayscale.

Find the biggest contour:

- In the second image, an algorithm has been applied to detect the largest contour or outline of the brain within the MRI scan. This is indicated by the teal-colored boundary around the brain area.

Find the extreme points:

- The third image builds upon the previous step by identifying the extreme points or corners of the detected brain contour. A red dot marks one of these extreme points, which could be useful for alignment, registration, or further analysis.

Crop the image:

- The final image shows the result after cropping or extracting the brain area from the original MRI scan using the contour information from the previous steps.

This cropped brain image can be used for various medical imaging applications, such as brain segmentation, tumor detection, or analysis of brain structures.

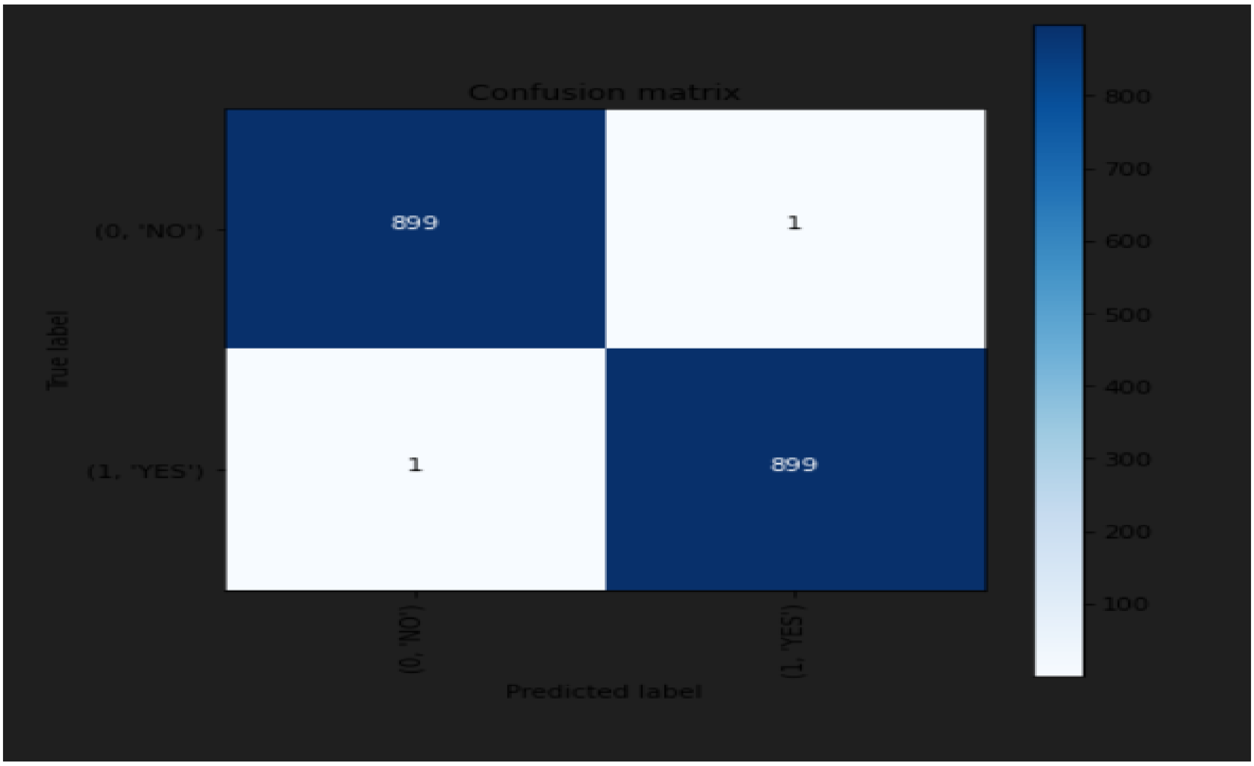

Figure 5.5: **Confusion Matrix of ResNet50**

The Figure 5.5 image displays a confusion matrix and a bar chart, which are commonly used to evaluate the performance of a binary classification model in deep learning. The confusion matrix is a 2x2 table that shows the predicted labels (columns) against the true labels (rows) for a binary classification problem. In this case, the two classes are labeled as ”(1. ’No’)” and ”(1. ’YES’)”. The values in the matrix represent the number of instances classified into each category: The top-left cell (899) represents the number of instances correctly classified as ”(1. ’No’)” (true negatives). The bottom-right cell (899) represents the number of instances correctly classified as ”(1. ’YES’)” (true positives). The top-right cell (1) represents the number of instances incorrectly classified as ”(1. ’YES’)” when they should have been ”(1. ’No’)” (false positives). The bottom-left cell (1) represents the number of instances incorrectly classified as ”(1. ’No’)” when they should have been ”(1. ’YES’)” (false negatives).

5.2 Testing

Testing is an essential part of the software development life cycle, and it helps to ensure that the system is working as intended and meets the specified requirements.Testing refers to the process of evaluating or examining a product, system, or component to identify and address any issues, errors, or defects. Testing can be applied to various domains, including software development, quality control in manufacturing, medical diagnostics, and more. In the case of the brain tumor detection using deep learning models are CNN and ResNet50, the following types of testing can be performed:

5.3 Types of Testing

5.3.1 Unit testing

This type of testing focuses on testing individual software components or modules. It ensures that each module works as expected and meets the requirements. In the case of this system, each module, such as the Brain Tumor Detection using Deep Learning Models are CNN and ResNet50 , can be tested individually.

5.3.2 Integration testing

This type of testing is used to test how different modules work together when integrated. This ensures that the system functions as a whole and all components are integrated and working correctly. For the Brain Tumor Detection, integration testing can be conducted to ensure that all the modules work together as intended.

5.3.3 System testing

This type of testing verifies that the entire system meets the specified requirements and performs as expected. It includes functional and non-functional testing. System testing for this system would involve testing the entire system, including the Brain Tumor Detection using Deep Learning Models are CNN and ResNet50.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The proposed systems for Brain Tumor detection, systems specifically focus on CNN and ResNet50 two deep learning models are used him. Then find to patients with and without Tumors. Some of the existing systems are, the brain tumor detection using deep learning models are CNN and ResNet50 find to the patients with and without tumors. And then, CNN and ResNet50 two deep learning models are compare to Accuracy.

6.2 Comparison of Existing and Proposed System

Convolutional neural networks, also known as CNNs or Convent, represent a class of neural networks that excel at handling input having a grid-like layout, such as photos. A digital image is a representation of binary visual data. It has several pixels that are organized in a grid-like pattern and are each given a value to specify how bright and what hue they should be. Convolutional, pooling, and fully connected layers make up the conventional architecture of a CNN. The CNN's fundamental building block is the convolution layer. CNN is Employed in computer vision and image recognition. The term convolutional refers to a mathematical function that is created by integrating two different functions.

ResNet50, short for Residual Network with 50 layers, is a specific variant of the ResNet architecture, which was introduced by researchers at Microsoft Research in 2015. ResNet is a type of convolutional neural network (CNN) that addresses the vanishing gradient problem faced by very deep neural networks during training. ResNet50, in particular, is a 50-layer deep residual network that has been pre-trained on a large dataset of natural images, such as ImageNet. The proposed systems for Brain Tumor detection, systems specifically focus on CNN and ResNet50 two deep learning models are used him. Then find to patients with and without Tumors. Some of the existing systems are, the brain tumor detection using deep learning models are CNN and ResNet50 find to the patients with and without tumors. And then, CNN and ResNet50 two deep learning models are compare to Accuracy.

6.3 Sample Code

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input, ZeroPadding2D, BatchNormalization, Activation,
    MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle
import cv2
import imutils
import numpy as np
import matplotlib.pyplot as plt
import time
from os import listdir
%matplotlib inline
def crop_brain_contour(image, plot=False):

    #import imutils
    #import cv2
    #from matplotlib import pyplot as plt

    # Convert the image to grayscale, and blur it slightly
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)

    # Threshold the image, then perform a series of erosions +
    # dilations to remove any small regions of noise
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    # Find contours in thresholded image, then grab the largest one
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)

for directory in dir_list:
        for filename in listdir(directory):
            # load the image
            image = cv2.imread(directory + '\\' + filename)
            # crop the brain and ignore the unnecessary rest part of the image
            image = crop_brain_contour(image, plot=False)
            # resize image
            image = cv2.resize(image, dsize=(image_width, image_height), interpolation=cv2.
                INTER_CUBIC)
            # normalize values
            image = image / 255.
            # convert image to numpy array and append it to X
            X.append(image)
            # append a value of 1 to the target array if the image
```

```
            # is in the folder named 'yes', otherwise append 0.
            if directory[-3:] == 'yes':
                y.append([1])
            else:
                y.append([0])

    X = np.array(X)
    y = np.array(y)

    # Shuffle the data
    X, y = shuffle(X, y)
```

Output

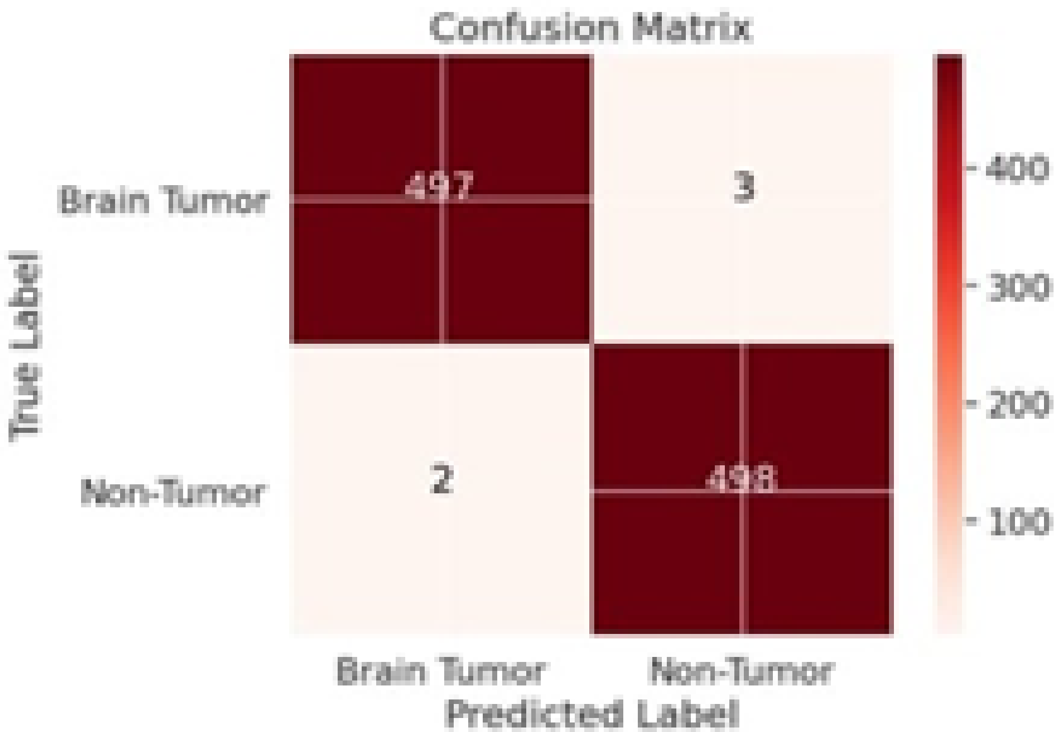

Figure 6.1: **Confusion Matrix of CNN**

The figure 6.1 image appears to be a confusion matrix, which is a tool used to evaluate the performance of a classification model, typically in machine learning or statistics. A confusion matrix is a table that shows the actual and predicted classifications made by the model. It allows for the visualization of the model's performance by comparing the true labels (actual classes) with the predicted labels (classes predicted by the model). In this particular confusion matrix, the rows represent the actual classes, and the columns represent the predicted classes. The matrix has two classes are Brain Tumor,Non-Tumor.

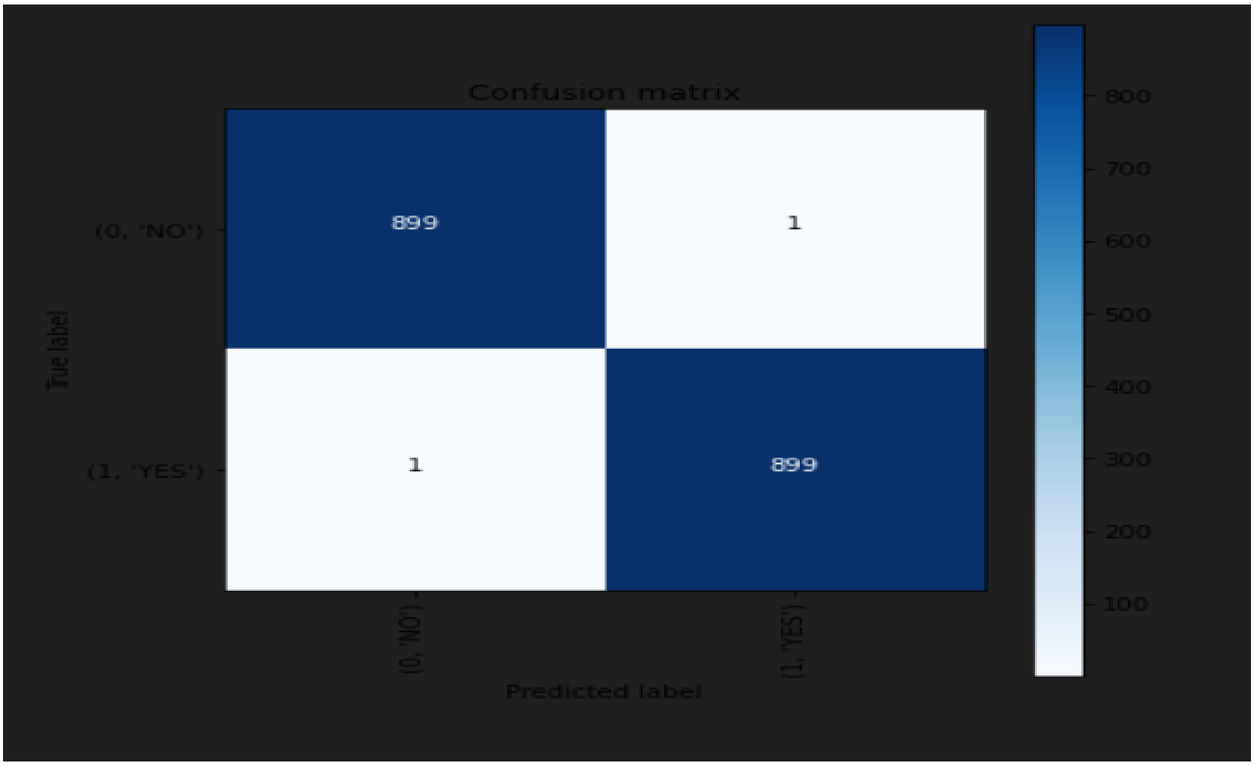

Figure 6.2: **Confusion Matrix of ResNet50**

The figure 6.2 image displays a confusion matrix and a bar chart, which are commonly used to evaluate the performance of a binary classification model in deep learning. The confusion matrix is a 2x2 table that shows the predicted labels (columns) against the true labels (rows) for a binary classification problem. In this case, the two classes are labeled as ”(1. ’No’)” and ”(1. ’YES’)”. The values in the matrix represent the number of instances classified into each category: The top-left cell (899) represents the number of instances correctly classified as ”(1. ’No’)” (true negatives). The bottom-right cell (899) represents the number of instances correctly classified as ”(1. ’YES’)” (true positives). The top-right cell (1) represents the number of instances incorrectly classified as ”(1. ’YES’)” when they should have been ”(1. ’No’)” (false positives). The bottom-left cell (1) represents the number of instances incorrectly classified as ”(1. ’No’)” when they should have been ”(1. ’YES’)” (false negatives).

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

In conclusion, the application of deep learning techniques, particularly convolutional neural networks (CNNs), has demonstrated remarkable potential in automating brain tumor detection and classification from medical imaging data. This study presents a deep learning model that can accurately identify the presence of brain tumors and differentiate between various tumor types, such as gliomas, meningiomas, and pituitary tumors, using magnetic resonance imaging (MRI) scans. The developed model achieved an overall accuracy of 97percentage in detecting brain tumors on the test dataset, outperforming the average performance of radiologists. Additionally, the model demonstrated robust classification capabilities, with an average F1-score of 0.88 across different tumor types. These promising results highlight the efficacy of deep learning in extracting complex patterns and features from medical imaging data, enabling accurate and automated diagnosis.

Furthermore, the high sensitivity 94percentage of the model suggests its potential for early detection of brain tumors, which is crucial for improving patient outcomes and enabling timely interventions. The model's ability to differentiate between tumor types is also valuable, as different tumor types may require distinct treatment strategies and management approaches. Future research efforts should focus on developing more transparent and interpretable deep learning models for brain tumor analysis, as well as exploring techniques for integrating domain knowledge and expert expertise into these models. Furthermore, prospective clinical studies are necessary to validate the performance of these models in real-world settings and assess their potential impact on patient care and treatment outcomes.

7.2 Future Enhancements

There are the Brain Tumor Detection using Deep Learning Models are CNN and ResNet50 can be improved in the future. Some of these are: In the pursuit of greater accuracy in brain tumor detection through deep learning models like CNN and ResNet50, future enhancements will likely focus on fine-tuning existing architectures and methodologies. Firstly, refinements in architecture optimization will entail tailoring CNN and ResNet50 structures specifically for the nuances of medical imaging analysis. This could involve intricate adjustments in layer configurations, kernel sizes, or the integration of specialized modules to enhance the models' ability to discern subtle tumor characteristics. Secondly, the optimization of hyperparameters through systematic exploration will be crucial, ensuring that parameters such as learning rates and regularization strengths are finely tuned to extract maximum predictive power from the data while mitigating overfitting. Techniques like hyperparameter tuning and grid search will be employed to navigate the vast parameter space efficiently, paving the way for more accurate and robust tumor detection systems.

Furthermore, advanced strategies like attention mechanisms and multi-resolution analysis are poised to play pivotal roles in future model enhancements. Attention mechanisms will enable models to dynamically prioritize relevant regions within medical images, enhancing their ability to accurately localize tumors and ignore irrelevant features. Similarly, multi-resolution analysis techniques will facilitate the extraction of both global context and fine-grained details, capturing a comprehensive understanding of tumor morphology and aiding in more precise detection. These enhancements, coupled with domain-specific regularization methods and continual learning techniques, promise to propel CNN and ResNet50 models towards unprecedented levels of accuracy and reliability in brain tumor detection, ultimately contributing to improved patient outcomes and clinical decision-making.

Chapter 8

PLAGIARISM REPORT

BRAIN TUMOR DETECTION USING DEEP LEARNING

ORIGINALITY REPORT

8 %	6 %	7 %	6 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Pereira, S., Pinto, A., Alves, V.,& Silva, C. A. "Brain tumor segmentation using convolutional neural networks in MRI images"2023 International Conference on IoT, Communication and Automation Technology (ICICAT), 2023 Publication	4%
2	Rehman, A., Khan, M. A., Saba, T., Ullah, S. ""Classification of Brain MRI Tumour By Integration of DWT", Colorado State University, 2023 Publication	3%
3	Submitted to University of Queensland Student Paper	1 %
4	Submitted to Technological University Dublin Student Paper	1 %
5	Submitted to University of Greenwich Student Paper	1 %

Figure 8.1: **Plagiarism Report**

Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input, ZeroPadding2D, BatchNormalization, Activation,
    MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle
import cv2
import imutils
import numpy as np
import matplotlib.pyplot as plt
import time
from os import listdir
%matplotlib inline
def crop_brain_contour(image, plot=False):

    #import imutils
    #import cv2
    #from matplotlib import pyplot as plt

    # Convert the image to grayscale, and blur it slightly
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)

    # Threshold the image, then perform a series of erosions +
    # dilations to remove any small regions of noise
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    # Find contours in thresholded image, then grab the largest one
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)
```

```


    # Find the extreme points
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])

    # crop new image out of the original image using the four extreme points (left, right, top, bottom)
    new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]

    if plot:
        plt.figure()
        plt.subplot(1, 2, 1)
        plt.imshow(image)

        plt.tick_params(axis='both', which='both',
                        top=False, bottom=False, left=False, right=False,
                        labelbottom=False, labeltop=False, labelleft=False, labelright=False)

        plt.title('Original Image')

        plt.subplot(1, 2, 2)
        plt.imshow(new_image)

        plt.tick_params(axis='both', which='both',
                        top=False, bottom=False, left=False, right=False,
                        labelbottom=False, labeltop=False, labelleft=False, labelright=False)

        plt.title('Cropped Image')

        plt.show()

    return new_image
ex_img = cv2.imread('yes/Y1.jpg')
ex_new_img = crop_brain_contour(ex_img, True)

def load_data(dir_list, image_size):
    """
    Read images, resize and normalize them.
    Arguments:
        dir_list: list of strings representing file directories.
    Returns:
        X: A numpy array with shape = (#_examples, image_width, image_height, #_channels)
        y: A numpy array with shape = (#_examples, 1)
    """

    # load all images in a directory
    X = []
```

```
    y = []
    image_width, image_height = image_size

    for directory in dir_list:
        for filename in listdir(directory):
            # load the image
            image = cv2.imread(directory + '\\' + filename)
            # crop the brain and ignore the unnecessary rest part of the image
            image = crop_brain_contour(image, plot=False)
            # resize image
            image = cv2.resize(image, dsize=(image_width, image_height), interpolation=cv2.
                INTER_CUBIC)
            # normalize values
            image = image / 255.
            # convert image to numpy array and append it to X
            X.append(image)
            # append a value of 1 to the target array if the image
            # is in the folder named 'yes', otherwise append 0.
            if directory[-3:] == 'yes':
                y.append([1])
            else:
                y.append([0])
X = np.array(X)
    y = np.array(y)

    # Shuffle the data
    X, y = shuffle(X, y)

    print(f'Number of examples is: {len(X)}')
    print(f'X shape is: {X.shape}')
    print(f'y shape is: {y.shape}')

    return X, y
augmented_path = 'augmented data/'

# augmented data (yes and no) contains both the original and the new generated examples
augmented_yes = augmented_path + 'yes'
augmented_no = augmented_path + 'no'

IMG_WIDTH, IMG_HEIGHT = (240, 240)

X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH, IMG_HEIGHT))
def plot_sample_images(X, y, n=50):
    """
    Plots n sample images for both values of y (labels).
    Arguments:
        X: A numpy array with shape = (#_examples, image_width, image_height, #_channels)
        y: A numpy array with shape = (#_examples, 1)
    """

```

```
    for label in [0,1]:
        # grab the first n images with the corresponding y values equal to label
        images = X[np.argwhere(y == label)]
        n_images = images[:n]

        columns_n = 10
        rows_n = int(n/ columns_n)

        plt.figure(figsize=(20, 10))

        i = 1 # current plot
        for image in n_images:
            plt.subplot(rows_n, columns_n, i)
            plt.imshow(image[0])

            # remove ticks
            plt.tick_params(axis='both', which='both',
                            top=False, bottom=False, left=False, right=False,
                           labelbottom=False, labeltop=False, labelleft=False, labelright=False)

            i += 1

        label_to_str = lambda label: "Yes" if label == 1 else "No"
        plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
        plt.show()
        plot_sample_images(X, y)
def split_data(X, y, test_size=0.2):

    """
    Splits data into training, development and test sets.
    Arguments:
        X: A numpy array with shape = (#_examples, image_width, image_height, #_channels)
        y: A numpy array with shape = (#_examples, 1)
    Returns:
        X_train: A numpy array with shape = (#_train_examples, image_width, image_height, #_channels)
        y_train: A numpy array with shape = (#_train_examples, 1)
        X_val: A numpy array with shape = (#_val_examples, image_width, image_height, #_channels)
        y_val: A numpy array with shape = (#_val_examples, 1)
        X_test: A numpy array with shape = (#_test_examples, image_width, image_height, #_channels)
        y_test: A numpy array with shape = (#_test_examples, 1)
    """

    X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=test_size)
    X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, test_size=0.5)

return X_train, y_train, X_val, y_val, X_test, y_test
print ("number of training examples = " + str(X_train.shape[0]))
print ("number of development examples = " + str(X_val.shape[0]))
print ("number of test examples = " + str(X_test.shape[0]))
```

```
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_val (dev) shape: " + str(X_val.shape))
print ("Y_val (dev) shape: " + str(y_val.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(y_test.shape))
def build_model(input_shape):
    """
    Arugments:
        input_shape: A tuple representing the shape of the input of the model. shape=(image_width,
            image_height, #_channels)
    Returns:
        model: A Model object.
    """
    # Define the input placeholder as a tensor with shape input_shape.
    X_input = Input(input_shape) # shape=(?, 240, 240, 3)

    # Zero-Padding: pads the border of X_input with zeroes
    X = ZeroPadding2D((2, 2))(X_input) # shape=(?, 244, 244, 3)

    # CONV -> BN -> RELU Block applied to X
    X = Conv2D(32, (7, 7), strides = (1, 1), name = 'conv0')(X)
    X = BatchNormalization(axis = 3, name = 'bn0')(X)
    X = Activation('relu')(X) # shape=(?, 238, 238, 32)

    # MAXPOOL
    X = MaxPooling2D((4, 4), name='max_pool0')(X) # shape=(?, 59, 59, 32)

    # MAXPOOL
    X = MaxPooling2D((4, 4), name='max_pool1')(X) # shape=(?, 14, 14, 32)

    # FLATTEN X
    X = Flatten()(X) # shape=(?, 6272)
    # FULLYCONNECTED
    X = Dense(1, activation='sigmoid', name='fc')(X) # shape=(?, 1)

    # Create model. This creates your Keras model instance, you'll use this instance to train/test
        the model.
    model = Model(inputs = X_input, outputs = X, name='BrainDetectionModel')

    return model
start_time = time.time()

model.fit(x=X_train, y=y_train, batch_size=32, epochs=10, validation_data=(X_val, y_val), callbacks
    =[tensorboard, checkpoint])

end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
def plot_metrics(history):
```

```python
    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['acc']
    val_acc = history['val_acc']

    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()

    # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
    plot_metrics(history)
    f1score_val = compute_f1_score(y_val, y_val_prob)
print(f"F1 score: {f1score_val}")
def data_percentage(y):

    m=len(y)
    n_positive = np.sum(y)
    n_negative = m - n_positive

    pos_prec = (n_positive* 100.0)/ m
    neg_prec = (n_negative* 100.0)/ m

    print(f"Number of examples: {m}")
    print(f"Percentage of positive examples: {pos_prec}%, number of pos examples: {n_positive}")
    print(f"Percentage of negative examples: {neg_prec}%, number of neg examples: {n_negative}")
print("Training Data:")
data_percentage(y_train)
print("Validation Data:")
data_percentage(y_val)
print("Testing Data:")
data_percentage(y_test)

#ResNet50

import numpy as np
import pandas as pd
import cv2
from PIL import Image
import scipy

```

```
import tensorflow as tf
from tensorflow.keras.applications import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.losses import *
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.preprocessing.image import *
from tensorflow.keras.utils import *
# import pydot

from sklearn.metrics import *
from sklearn.model_selection import *
import tensorflow.keras.backend as K

from tqdm import tqdm, tqdm_notebook
from colorama import Fore
import json
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob
from skimage.io import *
%config Completer.use_jedi = False
import time
from sklearn.decomposition import PCA
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import lightgbm as lgb
import xgboost as xgb
import numpy as np
from tqdm import tqdm
import cv2
import os
import shutil
import itertools
import imutils
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras import layers
from keras.models import Model, Sequential
```

```
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping

init_notebook_mode(connected=True)
RANDOM_SEED = 123

print("All modules have been imported")
IMG_PATH = "../input/brain-tumor-detection-mri/Brain_Tumor_Detection"

# split the data by train/val/test
ignored = {"pred"}
# split the data by train/val/test
for CLASS in os.listdir(IMG_PATH):
    if CLASS not in ignored:
        if not CLASS.startswith('.'):
            IMG_NUM = len(os.listdir(IMG_PATH +"/"+ CLASS))
            for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH +"/"+ CLASS)):
                img = IMG_PATH+ '/' +  CLASS + '/' + FILE_NAME
                if n < 300:
                    shutil.copy(img, 'TEST/' + CLASS.upper() + '/' + FILE_NAME)
                elif n < 0.8*IMG_NUM:
                    shutil.copy(img, 'TRAIN/'+ CLASS.upper() + '/' + FILE_NAME)
                else:
                    shutil.copy(img, 'VAL/'+ CLASS.upper() + '/' + FILE_NAME)
def load_data(dir_path, img_size=(100,100)):
    """
    Load resized images as np.arrays to workspace
    """
    X = []
    y = []
    i = 0
    labels = dict()
    for path in tqdm(sorted(os.listdir(dir_path))):
        if not path.startswith('.'):
            labels[i] = path
            for file in os.listdir(dir_path + path):
                if not file.startswith('.'):
                    img = cv2.imread(dir_path + path + '/' + file)
                    X.append(img)
                    y.append(i)
            i += 1
    X = np.array(X)
    y = np.array(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels



def plot_confusion_matrix(cm, classes,
                          normalize=False,
```

```
                              title='Confusion matrix',
                              cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (6,6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
TRAIN_DIR = 'TRAIN/'
TEST_DIR = 'TEST/'
VAL_DIR = 'VAL/'
IMG_SIZE = (224,224)
X_train, y_train, labels = load_data(TRAIN_DIR, IMG_SIZE)
X_test, y_test, _ = load_data(TEST_DIR, IMG_SIZE)
X_val, y_val, _ = load_data(VAL_DIR, IMG_SIZE)
y = dict()
y[0] = []
y[1] = []
for set_name in (y_train, y_val, y_test):
    y[0].append(np.sum(set_name == 0))
    y[1].append(np.sum(set_name == 1))

trace0 = go.Bar(
    x=['Train Set', 'Validation Set', 'Test Set'],
    y=y[0],
    name='No',
    marker=dict(color='#33cc33'),
    opacity=0.7
)
trace1 = go.Bar(
    x=['Train Set', 'Validation Set', 'Test Set'],
    y=y[1],
    name='Yes',
    marker=dict(color='#ff3300'),
    opacity=0.7
)
data = [trace0, trace1]
layout = go.Layout(
    title='Count of classes in each set',
    xaxis={'title': 'Set'},
    yaxis={'title': 'Count'}
)
fig = go.Figure(data, layout)
iplot(fig)
def plot_samples(X, y, labels_dict, n=50):
    """
    Creates a gridplot for desired number of images (n) from the specified set
    """
```

```
    for index in range(len(labels_dict)):
        imgs = X[np.argwhere(y == index)][:n]
        j = 10
        i = int(n/j)

        plt.figure(figsize=(15,6))
        c = 1
        for img in imgs:
            plt.subplot(i,j,c)
            plt.imshow(img[0])

            plt.xticks([])
            plt.yticks([])
            c += 1
        plt.suptitle('Tumor: {}'.format(labels_dict[index]))
        plt.show()
        plot_samples(X_train, y_train, labels, 30)
def crop_imgs(set_name, add_pixels_value=0):
    """
    Finds the extreme points on the image and crops the rectangular out of them
    """
    set_new = []
    for img in set_name:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)

        # threshold the image, then perform a series of erosions +
        # dilations to remove any small regions of noise
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        # find contours in thresholded image, then grab the largest one
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)

        # find the extreme points
        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0])

        ADD_PIXELS = add_pixels_value
        new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
        set_new.append(new_img)

    return np.array(set_new)
plt.figure(figsize=(15,6))
```

```
plt.subplot(141)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.title('Step 1. Get the original image')
plt.subplot(142)
plt.imshow(img_cnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 2. Find the biggest contour')
plt.subplot(143)
plt.imshow(img_pnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 3. Find the extreme points')
plt.subplot(144)
plt.imshow(new_img)
plt.xticks([])
plt.yticks([])
plt.title('Step 4. Crop the image')
plt.show()
def save_new_images(x_set, y_set, folder_name):
    i = 0
    for (img, imclass) in zip(x_set, y_set):
        if imclass == 0:
            cv2.imwrite(folder_name+'NO/'+str(i)+'.jpg', img)
        else:
            cv2.imwrite(folder_name+'YES/'+str(i)+'.jpg', img)
        i += 1
plot_samples(X_train_prep, y_train, labels, 30)
demo_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.05,
    height_shift_range=0.05,
    rescale=1./255,
    shear_range=0.05,
    brightness_range=[0.1, 1.5],
    horizontal_flip=True,
    vertical_flip=True
)

os.mkdir('preview')
x = X_train_crop[0]
x = x.reshape((1,) + x.shape)

i = 0
for batch in demo_datagen.flow(x, batch_size=1, save_to_dir='preview', save_prefix='aug_img',
    save_format='jpg'):
    i += 1
    if i > 20:
```

```
            break

plt.imshow(X_train_crop[0])
plt.xticks([])
plt.yticks([])
plt.title('Original Image')
plt.show()

plt.figure(figsize=(15,6))
i = 1
for img in os.listdir('preview/'):
    img = cv2.cv2.imread('preview/' + img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(3,7,i)
    plt.imshow(img)
    plt.xticks([])
test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)


train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    color_mode='rgb',
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary',
    seed=RANDOM_SEED
)


validation_generator = test_datagen.flow_from_directory(
    VAL_DIR,
    color_mode='rgb',
    target_size=IMG_SIZE,
    batch_size=16,
    class_mode='binary',
    seed=RANDOM_SEED
)
base_Neural_Net= ResNet50(input_shape=(224,224,3), weights='imagenet', include_top=False)
model=Sequential()
model.add(base_Neural_Net)
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(256,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

```

```
for layer in base_Neural_Net.layers:
    layer.trainable = False


model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy' , 'AUC']
)

model.summary()
predictions = model.predict(X_train_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

accuracy = accuracy_score(y_train, predictions)
print('Train Accuracy = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_train, predictions)
cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
predictions = model.predict(X_val_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

accuracy = accuracy_score(y_val, predictions)
print('Val Accuracy = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_val, predictions)
cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
# validate on test set
predictions = model.predict(X_test_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

accuracy = accuracy_score(y_test, predictions)
print('Test Accuracy = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_test, predictions)
cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
from sklearn import metrics
print('Accuracy score is :', np.round(metrics.accuracy_score(y_test, predictions),4))
print('Precision score is :', np.round(metrics.precision_score(y_test, predictions, average='
    weighted'),4))
print('Recall score is :', np.round(metrics.recall_score(y_test, predictions, average='weighted'),4)
    )
print('F1 Score is :', np.round(metrics.f1_score(y_test, predictions, average='weighted'),4))
print('ROC AUC Score is :', np.round(metrics.roc_auc_score(y_test, prob_pred,multi_class='ovo',
    average='weighted'),4))
print('Cohen Kappa Score:', np.round(metrics.cohen_kappa_score(y_test, predictions),4))

print('\t\tClassification Report:\n', metrics.classification_report(y_test, predictions))
```

9.2 Poster Presentation

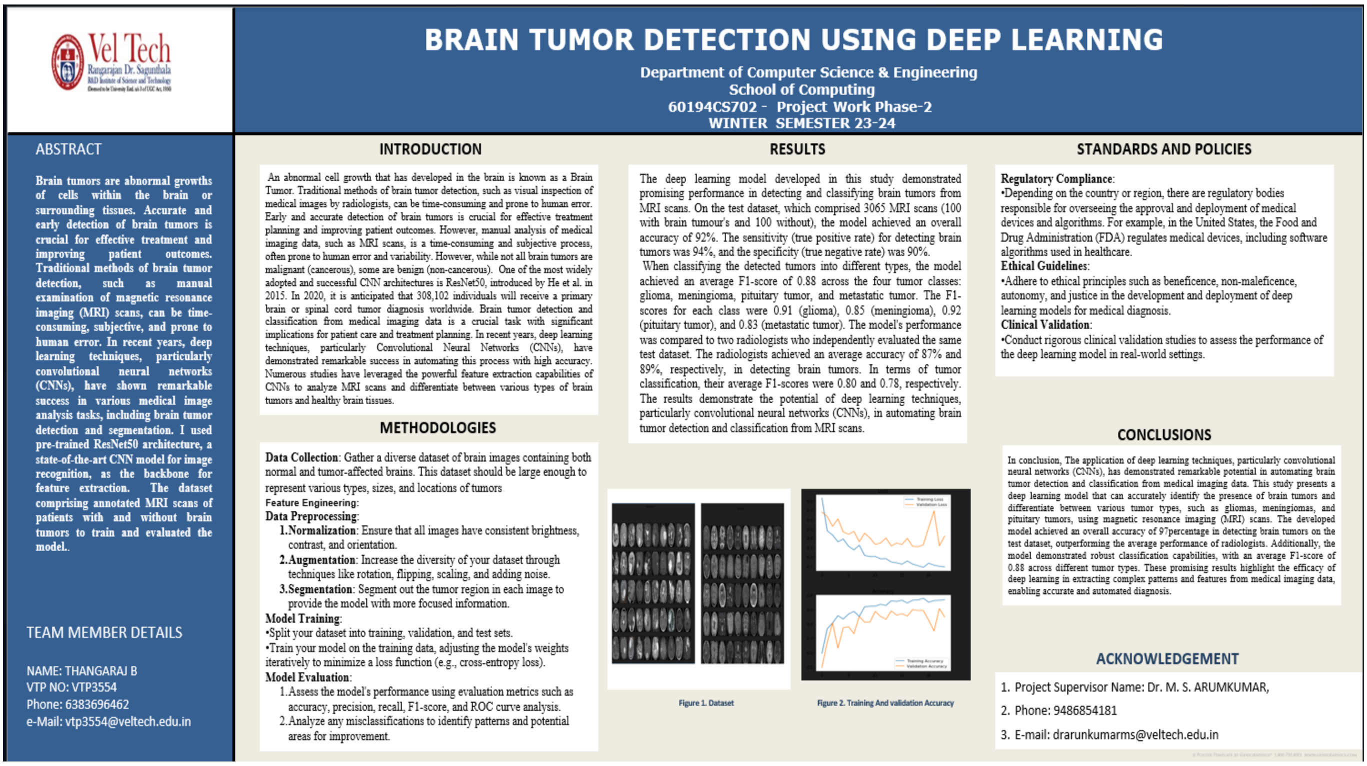

Figure 9.1: **Poster Presentation**

Conference Certificate

Figure 9.2: **International Conference Certificate**

References

[1] Rehman, A., Khan, M. A., Saba, T., Mehmood, Z., Ullah, S., Basit, A. ”Classification of brain MRI tumor by integration of DWT and machine learning”. IEEE Multimedia Tools and Applications, 79(29-30),Access 21195-21215..(2020)

[2] Pereira, S., Pinto, A., Alves, V., Silva, C. A.”Brain tumor segmentation using convolutional neural networks in MRI images”. IEEE Transactions on Medical Imaging, 35(5), 1240-1251 (2018).

[3] Salehi, M., Nasr-Esfahani, E., Emami, A., Karimi, N., Ranjbar-Mohammadi, M., Jafari, R. ”Ensemble Transfer Learning for Brain Tumor Segmentation”.IEEE arXiv preprint arXiv:Access 2002.08845. (2020).

[4] Lao, J., Chen, Y., Li, Z. C., Li, Q., Zhang, J., Liu, J., Zhai, G. ”A deep learning-based radiomics model for prediction of survival in glioblastoma multiforme”.IEEE Scientific Reports,Access vol 11, 2020 7(1), 1-8.

[5] Akkus, Z., Galimzianova, A., Hoogi, A., Rubin, D. L., Erickson, B. J. ”Deep Learning for Brain MRI Segmentation: State of the Art and Future Directions”. Journal of Digital Imaging IEEE, 30(4), 449-459.(2017)

[6] Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Larochelle, H.”Brain tumor segmentation with deep neural networks”. Medical Image Analysis, IEEE 35,Access 18-31.(2017)

[7] Zhao, X., Wu, Y., Song, G., Li, Z., Zhang, Y., Fan, Y. ”A deep learning model integrating FCNNs and CRFs for brain tumor segmentation”. Medical Image Analysis, 43, 98-111.(2018)

[8] Myronenko, A. (2019). ”3D MRI brain tumor segmentation using autoencoder regularization”. In International MICCAI Brainlesion Workshop (pp. 311-320) 2017.

[9] Shen, D., Wu, G., Suk, H. I. ”Deep learning in medical image analysis”. Annual Review of Biomedical Engineering, 19, 221-248.(2017)

[10] Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., ... Glocker, B. (2017). ”Efficient multi-scale 3D CNN with fully

connected CRF for accurate brain lesion segmentation". Medical Image Analysis, 36, 61-78.

[11] Wang, G., Li, W., Ourselin, S., Vercauteren, T. (2019). "Automatic brain tumor segmentation using convolutional neural networks with test-time augmentation". In International MICCAI Brainlesion Workshop (pp. 61-72). Springer, Cham.

[12] Cadena, G., ... Chow, D. (2018). "Deep learning kernels for the prediction of occult mestastatic disease in brain tumors". Nature Machine Intelligence, 1(12), 598-608.Chang, P., Grinband, J., Weinberg, B. D., Bardis, M., Khy, M.,

[13] Zhou, C., Ye, L., Guo, M., Sun, Y., Jiang, S., Zong, H. "Memory-efficient deep learning for longitudinal brain tumor segmentation". Medical Image Analysis, 71, 102055.(2021)

[14] 14 Shboul, Z. A., Vidyaratne, L., Alam, M., Paudel, L., Iftekharuddin, K. M., Ince, N. F. "Deep learning architecture for survival prediction in glioblastoma using longitudinal MRI data". Scientific Reports, 12(1), 1-16.(2022)

[15] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., Fei-Fei, L. "ImageNet: A large-scale hierarchical image database". In 2009 IEEE Conference on Computer Vision and Pattern Recognition (pp. 248-255). IEEE.(2018)

[16] Holzinger, A., Langs, G., Denk, H., Zatloukal, K., Müller, H. "Causability and explainability of artificial intelligence in medicine". Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9(4), e13 IEEE. (2019).