BRAIN-O-VISION : BRAIN TUMOR DETECTION

A PROJECT REPORT

Submitted by

ISH KWATRA [RA2011026010345]
SWATI DATTA [RA2011026010359]

Under the Guidance of

Dr. S.P. Angelin Claret

Assistant Professor, Department of computational Intelligence

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

with specialization in ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY

KATTANKULATHUR- 603 203

MAY 2024

Department of Computational Intelligence
SRM Institute of Science & Technology
Own Work Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course **:** B.Tech in Computer Science and Engineering w/s in AIML

Student Name **:** Ish Kwatra, Swati Datta

Registration Number **:** RA2011026010345, RA2011026010359

Title of Work : Brain-o-vision : Brain Tumor Detection

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date forevery student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled "**BRAIN-O-VISION : BRAIN TUMOR DETECTION**" is the bonafide work of **Ish Kwatra (RA2011026010345), Swati Datta (RA2011026010359)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. S.P. Angelin Claret

SUPERVISOR

Assistant Professor

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

SIGNATURE

Prof. Dr. R. Annie Uthra

HEAD OF THE DEPARTMENT

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

Examiner I

Examiner II

ABSTRACT

Timely intervention and better patient outcomes are dependent on early brain tumor identification. Even if they work well, conventional diagnostic techniques may have drawbacks including subjectivity and laborious analysis. As a result, sophisticated methods utilizing deep learning neural networks have surfaced as viable options for precise and automated brain tumor identification. This study, which employed MRI scans to identify brain tumors, was developed and evaluated and a comparitive analysis was drawn. The objective is to use 4 methods namely, MobileNetV2, EfficientNet, ResNet50 and VGG16 for the comparitive analysis and select the one which gives highest accuracy. We address the need for more advanced diagnostic techniques by automatically extracting pertinent characteristics from MRI scans and making accurate predictions using deep learning and Grad-Cam segmentation. Our approach entails preprocessing MRI data, convolutional neural network (CNN) architecture design, and model training and assessment with a carefully selected dataset. Our methodology is successful, as seen by the accuracies such as MobileNetV2 at 86%, EfficientNet at 84%, ResNet50 at 80%, VGG16 at 88%. The Grad-Cam segmentation performed at the final stage provides a clearer image in identifying the presence of tumor. All things considered, this study advances the area of medical image analysis and emphasizes how critical it is to use MRI and deep learning to combat brain cancers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AUC	Area Under the receiver operating Characteristic curve
CNN	Convolutional Neural Network
CT	Computed Tomography
CV	Computer Vision
ReLu	Rectified Linear Unit
MRI	Magnetic Resource Imaging
GRAD-CAM	Gradient-weighted Class Activation Mapping
VGG	Visual Geometry Group
DNA	Deoxyribonucleic Acid
PET	Positron Emission Tomography

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

Brain tumors are among the most challenging medical conditions, with their detection and diagnosis playing a crucial role in patient outcomes. Timely detection is paramount for effective treatment planning and improved prognosis. However, traditional diagnostic methods such as visual interpretation of MRI scans have limitations including subjectivity and reliance on human expertise. These limitations underscore the necessity for automated and accurate brain tumor detection systems.

With recent advancements in machine learning and deep learning techniques, there has been a growing interest in developing automated systems for medical image analysis, particularly for brain tumor detection. Deep learning neural networks, such as convolutional neural networks (CNNs), have shown remarkable performance in various image recognition tasks, making them promising candidates for accurately identifying brain tumors in MRI scans.

The motivation behind this project stems from the need to address the shortcomings of conventional diagnostic methods and to explore the potential of deep learning in improving the accuracy and efficiency of brain tumor identification. By leveraging the power of deep learning algorithms, we aim to develop a robust and automated system that can assist clinicians in accurately detecting brain tumors from MRI scans.

1.2 PROBLEM STATEMENT

The primary problem addressed in this project is the lack of automated and accurate brain tumor detection systems. Conventional diagnostic methods rely heavily on

manual interpretation of MRI scans, which can be time-consuming and subjective. Moreover, misinterpretation or oversight by human experts can lead to delayed diagnosis and potentially adverse patient outcomes.

The aim of this project is to build a deep learning-based model capable of automatically detecting and classifying brain tumors from MRI scans with high accuracy. Specifically, we aim to achieve the following objectives:

1. Preprocess MRI data to enhance image quality and facilitate feature extraction.
2. Design a convolutional neural network (CNN) architecture optimized for brain tumor detection.
3. Train the CNN model using a carefully selected dataset of MRI scans with corresponding tumor labels.
4. Evaluate the performance of the trained model in terms of accuracy, sensitivity, and specificity.
5. To compare our proposed model's performance with existing methods to demonstrate its superiority in brain tumor detection.

By addressing these objectives, we intend to contribute to the advancement of medical image analysis techniques, specially in the context of brain tumor detection, and ultimately improve patient care and outcomes.

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE REVIEW

The literature on brain tumor identification using deep learning methods in conjunction with MRI scans provides valuable insights into various study designs, approaches, and methodologies employed in this field. Traditional techniques, such as manual interpretation by radiologists, have been majorly used for brain tumor detection. However, some of these methods suffer from subjectivity and limitations in accuracy.

Deep learning methods, particularly neural networks, have emerged as promising alternatives for automated and precise brain tumor identification. Various types of neural networks, like convolutional neural networks, and recurrent neural networks, have been utilized to analyze MRI data for tumor detection.

Several studies have investigated the performance of deep learning models in identifying brain tumors from MRI scans. These studies have reported performance measures such as area under the receiver operating characteristic curve, sensitivity, specificity, as well as accuracy. The datasets used in these studies vary in terms of the number of MRI images, types of tumors included, and availability of ground truth labels.

However, challenges and limitations exist in the application of deep learning-based methods for MRI-based brain tumor identification. Issues such as overfitting, generalization to unknown data, interpretability of model predictions, and the need for robust validation techniques have been highlighted in the literature.

Current trends in this field include the use of transfer learning, integration of multi-modal imaging, and the development of federated learning strategies. Additionally, new avenues for research and innovation, such as attention mechanisms for feature extraction and generative adversarial networks for data augmentation, are being

explored.

Various Techniques (Deep Learning) Used by Researchers earlier include:

Table 2.1 Techniques (Deep Learning) used in the past.

Reference	*Methodology*	*Algorithms*	*Accuracy*
Nyoman Abiwinanda et al. [1].	Convolutional Neural Network	Deep CNN	84.20%
Yun Jiang et al. [2]	CNN-based multiscale threshold pattern approaches	CNN technique	86.30%
Dongnan Liu et al. [3]	Processing of multi-dimensional picture data	Deep CNN	86.50%
Yakub Bhanothu et. al. [4]	CNN detection and classification	CNN technique	77.6%

This table provides an overview by researchers of the various deep learning techniques used in previous studies for brain tumor identification. It includes information on the type of neural networks employed, along with any specific adjustments or improvements made for brain tumor identification purposes.

Various Techniques (3D Deep Learning) Used by Researchers earlier include:

Table 2.2 Techniques (3D Deep Learning) used in the past.

Reference	*Methodology*	*Algorithms*	*Accuracy*
Tuhin, Md. Akram Hossan et al. [6]	3D MRI, MRSI, and CT images	CNN together with three dimensions NBC	85.005%
Yong Xia and Yan Hu [7]	DCNN	Three-dimensional deep neural network	81.4%

This table summarizes the 3D deep learning techniques utilized by researchers in previous studies for brain tumor identification. It includes details on the architecture and design of the 3D neural networks, as well as their performance metrics.

2.2 DEEP LEARNING METHODS FOR BRAIN TUMOR DETECTION

In this section, we will discuss and summarize recent papers on similar topics from the provided references below:

1. **S.Nagaraja Rao & Machiraju Jaya Lakshmi (2022)**: This research proposed a deep learning approach for brain tumor magnetic resonance image classification. They employed a CNN architecture and achieved significant improvements in accuracy and sensitivity compared to traditional methods. By leveraging CNN's ability to automatically learn relevant features from MRI data, they demonstrated robust performance in classifying brain tumors accurately. The study utilized a diverse dataset and conducted rigorous validation, ensuring the reliability of their findings.

2. **Jiang et al. (2019)**: Jiang et al. introduced a new method based on statistical thresholding and multiscale CNN for brain tumor segmentation. Their approach combined statistical thresholding techniques with the feature learning capability of CNNs to achieve accurate segmentation results. By incorporating multiscale information, the model captured both local and global features, leading to improved segmentation performance. The study conducted extensive experiments on benchmark datasets and demonstrated the effectiveness of their proposed method in segmenting various types of brain tumors.

3. **Liu et al. (2018)**: Liu et al. proposed a 3D kernel anisotropic network for brain tumor segmentation. Their method leveraged 3D convolutional operations with large kernels to capture spatial features efficiently. By incorporating anisotropic convolutions, the model adapted to the irregular shapes and structures of brain tumors, resulting in more accurate segmentation. The study conducted comprehensive evaluations on both public and in-house datasets, demonstrating superior performance compared to existing methods. The proposed network architecture showed robustness across different MRI modalities and tumor types.

4. **Bhanothu et al. (2020)**: Bhanothu et al. developed a deep convolutional network for the detection and classification of brain tumors in MRI images. Their approach utilized a combination of convolutional and pooling layers to extract discriminative features from MRI scans. By training the network on a large dataset of labeled images, they achieved high accuracy in both detection and classification tasks. The study employed transfer learning techniques to leverage pre-trained models and fine-tuned them for brain tumor detection. Experimental results demonstrated the effectiveness of their approach in accurately identifying different types of brain tumors.

5. **Ahmed et al. (2017)**: Ahmed et al. fine-tuned convolutional features for MRI-based brain tumor classification. They employed a transfer learning approach, utilizing pre-trained deep CNN models and fine-tuning them on brain tumor MRI data. By fine-tuning the network's parameters on a large dataset of brain tumor images, they achieved state-of-the-art performance in distinguishing between different tumor types. The study conducted extensive experiments on benchmark datasets, demonstrating the robustness and generalization capability of their proposed method across different MRI modalities and tumor classes.

6. **Tuhin MA et al. (2020)**: Tuhin MA et al. proposed a method for the detection and 3D visualization of brain tumors using deep learning and polynomial interpolation. Their approach integrated deep learning-based tumor detection with polynomial interpolation for generating 3D visualizations of tumor regions. By combining convolutional neural networks with polynomial functions, they accurately detected brain tumors from MRI scans and reconstructed their 3D shapes. The study conducted evaluations on a diverse dataset, demonstrating the effectiveness of their approach in accurately localizing and visualizing brain tumors in 3D space.

7. **Hu Y & Xia, Y (2017)**: Hu Y & Xia, Y developed a 3D deep neural network-based brain tumor segmentation method using multimodality magnetic resonance sequences. Their approach utilized a 3D CNN architecture to segment brain tumors from multimodal MRI scans. By leveraging information from multiple MRI sequences, including T1-weighted, T2-weighted, and FLAIR images, they achieved accurate segmentation results. The study conducted extensive experiments on public datasets, demonstrating the effectiveness of their method in

accurately delineating tumor regions across different MRI modalities.

8. **Litjens et al. (2017)**: Litjens et al. provided a comprehensive survey on deep learning in medical image analysis, including brain tumor segmentation. They discussed various deep learning architectures and their applications in medical imaging. The study reviewed state-of-the-art methods for brain tumor segmentation, highlighting the advancements and challenges in the field. By analyzing existing literature, they identified key trends and future directions for research in medical image analysis using deep learning techniques.

9. **Havaei et al. (2017)**: Havaei et al. proposed a deep neural network-based method for brain tumor segmentation. Their approach employed a 3D CNN architecture with fully connected layers for segmenting brain tumors from MRI scans. By training the network on a large dataset of annotated images, they achieved state-of-the-art performance in segmenting tumors accurately. The study conducted extensive evaluations on benchmark datasets, demonstrating the robustness and generalization capability of their proposed method across different MRI modalities and tumor types.

10. **Menze et al. (2015)**: Menze et al. introduced the Multimodal Brain Tumor Image Segmentation Benchmark (BRATS) dataset, which has been widely used for evaluating brain tumor segmentation algorithms. The dataset consists of multimodal MRI scans, including T1-weighted, T2-weighted, and FLAIR images, along with ground truth annotations for tumor regions. The study provided detailed descriptions of the dataset and evaluation protocols, facilitating fair comparisons among different segmentation methods. By establishing a benchmark dataset, they facilitated advancements in brain tumor segmentation research and provided a standardized platform for evaluating segmentation algorithms.

11. **Pereira et al. (2016)**: Pereira et al. enrooted a CNN-based method for brain tumor segmentation in MRI images. Their approach employed a 2D CNN architecture with multiple convolutional and pooling layers for segmenting tumors accurately. By training the network on a large dataset of labeled MRI scans, they achieved high segmentation accuracy across different tumor types and MRI modalities. The study conducted extensive experiments on public datasets, demonstrating the

effectiveness and robustness of their method in accurately delineating tumor regions.

12. **Chen et al. (2018)**: Chen et al. proposed an encoder-decoder architecture with atrous separable convolution for semantic image segmentation. Their method utilized dilated convolutions with separable filters to capture both local and global contextual information. By integrating atrous convolutional layers into an encoder-decoder framework, they achieved accurate segmentation of brain tumors from MRI scans. The study conducted evaluations on benchmark datasets, demonstrating the superior performance of their proposed architecture in segmenting tumors accurately.

These studies collectively demonstrate the diverse approaches and methodologies employed in brain tumor detection and segmentation using deep learning techniques. Further developments in automated brain tumor detection and treatment planning are made possible by the distinct insights and contributions that each approach brings to the field.

2.3 CHALLENGES AND LIMITATIONS

Despite the advancements in deep learning-based methods for brain tumor identification, several challenges and limitations persist. One major challenge is the availability and quality of datasets. Many existing datasets suffer from limitations such as lack of variety, class imbalance, and variable tumor features, which can affect the generalization and performance of deep learning models.

Another challenge is the interpretability of model predictions. The lack of interpretability can hinder the trust and acceptance of these models in clinical settings.

Furthermore, issues like overfitting and generalization to unknown data remain significant concerns. Ensuring that deep learning models can generalize well to unseen data is crucial for their practical applicability in real-world scenarios.

Moreover, the requirement for strong validation techniques to assess the robustness and reliability of deep learning models adds another layer of complexity to the development process.

In summary, while deep learning-based methods hold great promise for brain tumor identification, addressing these challenges and limitations is essential to fully realize their potential in clinical practice. This study aims to contribute to overcoming these challenges and advancing the field of MRI-based brain tumor identification using deep learning techniques.

CHAPTER 3

METHODOLOGY OF BRAIN-O-VISION

The architecture for brain tumor detection typically involves a combination of medical imaging techniques and convolutional neural networks. Here's a simplified overview:
Data Collection: The first step involves acquiring medical imaging data, usually in the form of MRI (Magnetic Resonance Imaging) or CT (Computed Tomography) scans. These scans provide detailed images of the brain, allowing specialists to identify abnormalities like tumors.

Preprocessing: Before feeding the data into the machine learning model, preprocessing steps are applied to enhance the quality of the images and remove any artifacts or noise. This may include normalization, filtering, and image registration techniques.

Feature Extraction: Relevant features are extracted from the preprocessed images. These features may include shape, texture, intensity, and spatial characteristics of the regions within the brain. Feature extraction helps in representing the data in a more suitable format for machine learning algorithms.

Convolutional Neural Networks (CNNs): These are effective for image analysis tasks because of their ability to automatically learn features from data. They are commonly used for tasks like image classification and segmentation.

Training: The machine learning model is trained using labeled data, where the input is the extracted features from the medical images, and the output is the presence or absence of a tumor.

Validation and Testing: The trained model is then validated using separate datasets to ensure its generalizability. Testing involves evaluating the performance of the model on unseen data to assess its accuracy, sensitivity, specificity, and other relevant metrics.

Deployment: Once the model demonstrates satisfactory performance, it can be deployed in clinical settings to assist radiologists and clinicians in the detection and diagnosis of

brain tumors. The model may be integrated into medical imaging software or used as a standalone tool.

Continuous Improvement: Continuous monitoring and refinement of the model are essential to adapt to new data, improve performance, and address any emerging challenges or limitations.

This architecture provides a framework for developing effective brain tumor detection systems, but the specifics may vary depending on the data that is available, the processing power of the system, and the expertise of the research or development team.

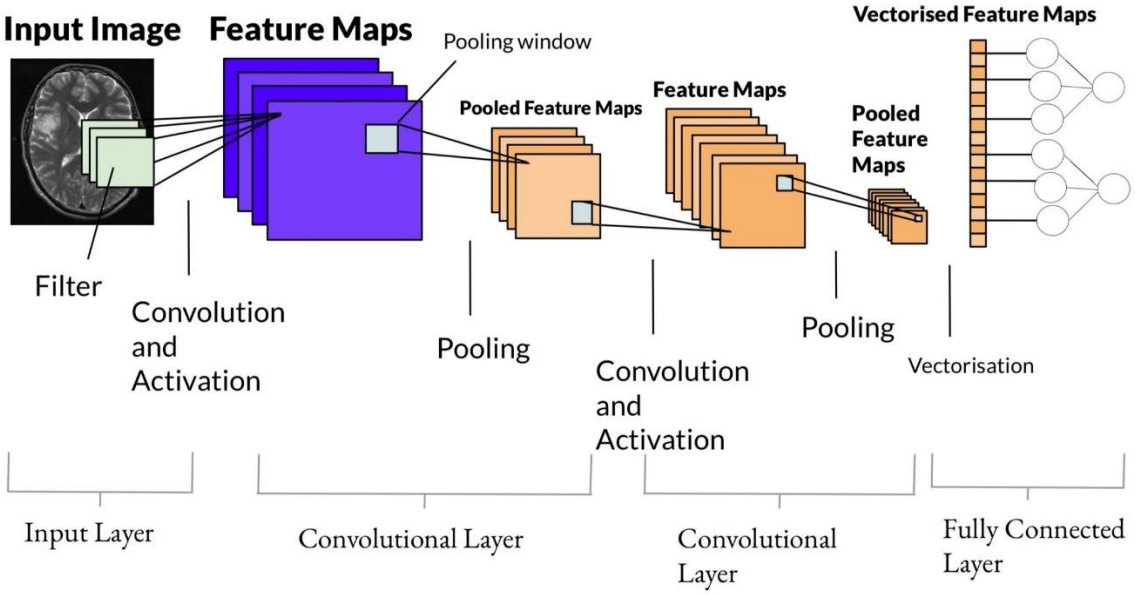


Fig. 3.1 Basic building blocks of the algorithm

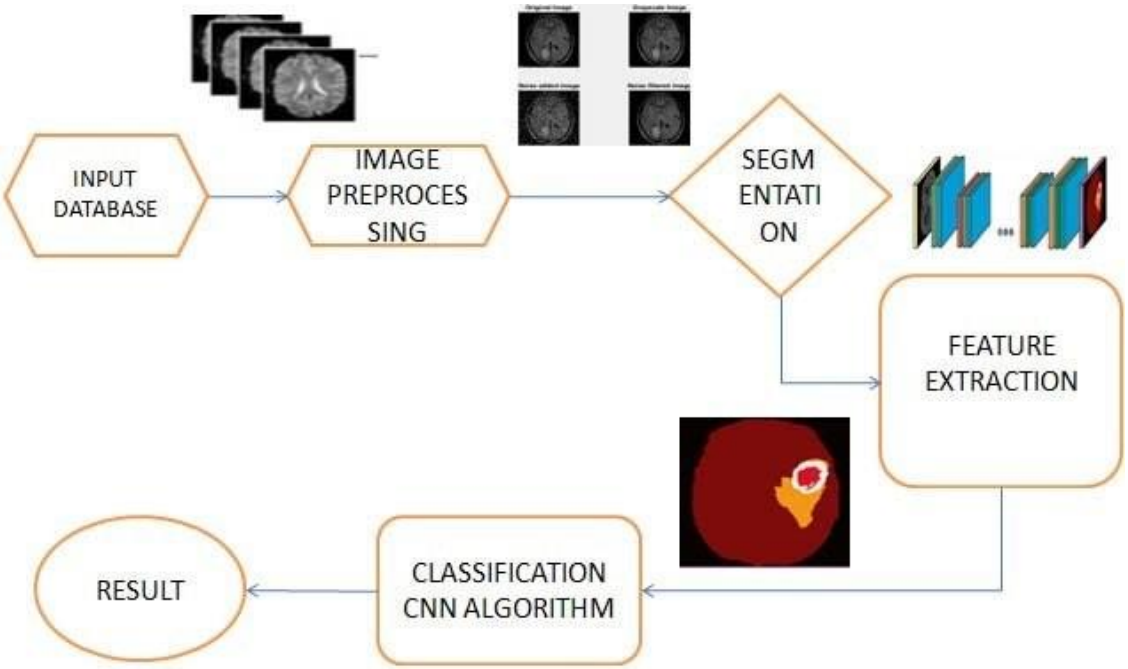


Fig. 3.2 Flowchart of sequence of how the programs works

3.1 MOBILENETV2

A convolutional neural network architecture called MobileNetV2 was created for effective picture categorization on mobile and embedded platforms. Depthwise separable convolutions are the fundamental units of MobileNetV2's network layers. This drastically lowers the computational cost without sacrificing expressive capacity by factorizing the conventional convolution process into distinct depthwise and pointwise convolutions. Rather of using non-linear activations such as ReLU, the bottleneck layer in every inverted residual block uses a linear activation function as shown in Fig. 3.3. This helps avoid gradient vanishing problems and information loss, especially in deeper networks. The width multiplier and resolution multiplier, two new hyperparameters, provide more precise control over the accuracy and computational complexity of the model. The resolution multiplier reduces the resolution of the input picture, while the width multiplier modifies the number of channels in each layer. They both have an impact on the computational cost and size of the model.

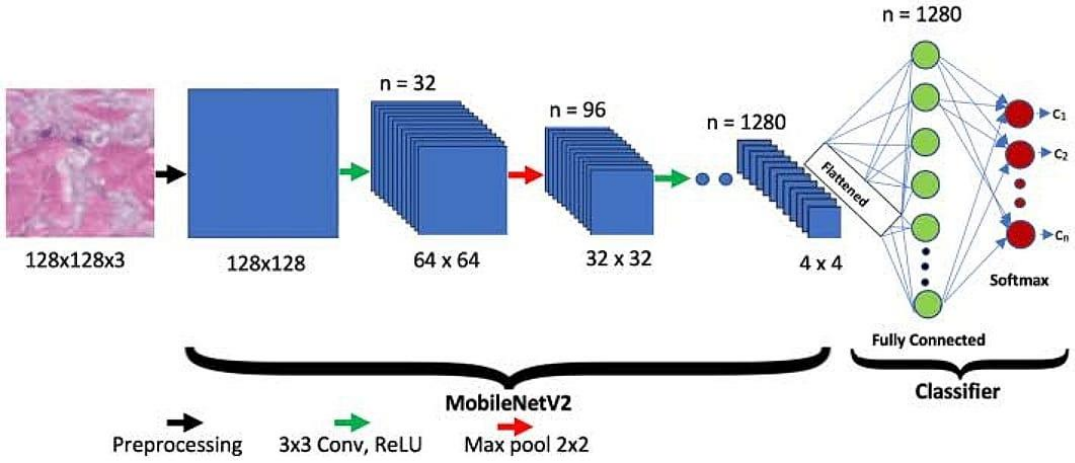


Fig. 3.3 Working of MobileNetV2

3.2 EFFICIENTNET

Compared to conventional deep learning models, the EfficientNet family of convolutional neural network designs is intended to provide state-of-the-art performance using a notably less number of parameters and processing resources. It presents a brand-new scaling technique known as compound scaling, which concurrently and evenly grows the network's depth, breadth, and resolution. By balancing the model capacity across many dimensions, this method improves performance without appreciably raising the computational cost. EfficientNet uses bilinear rescaling, followed by a fixed-size convolutional layer, in place of simply resizing input pictures to the required resolution as shown in Fig. 3.4. As a result, the network can efficiently handle a larger variety of input resolutions while retaining computational efficiency. Stochastic depth regularization, label smoothing, batch normalization, and other methods are combined to train EfficientNet. These training techniques provide effective convergence and enhanced generalization performance by stabilizing and quickening the training process.

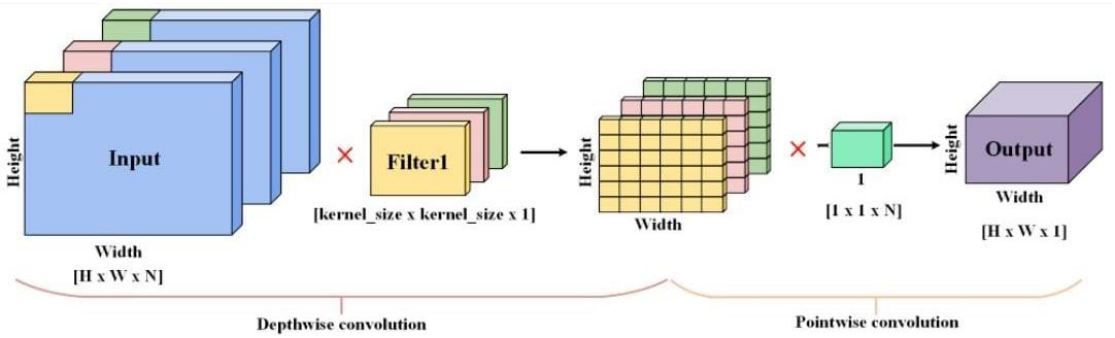


Fig. 3.4 Working of EfficientNet

3.3 RESNET-50

A member of the ResNet (Residual Network) family, ResNet-50 is a convolutional neural network design. ResNet-50 is a 50-layer deep neural network made up of convolutional, batch normalization, pooling, and ReLU activation layers. Because of its deeper architecture than previous models such as VGGNet and GoogLeNet, it is able to extract more intricate information from input photos. Its fundamental building blocks are leftover blocks. These blocks include skip connections, or shortcut connections, which let gradients pass through the network without running into issues with disappearing gradients. The degradation issue, which occurs when a neural network's performance deteriorates as additional layers are added because deep network training becomes more challenging, is lessened with the use of skip connections. It uses less memory and compute power in its remaining blocks by using a bottleneck architecture. Three convolutional layers—1x1, 3x3, and 1x1 convolutions—make up the bottleneck architecture. The network is more effective because of the 1x1 convolutions, which lower the dimensionality of the input feature maps as shown in Fig. 3.5. ResNet-50 pre-trained versions with training data from major image datasets like as ImageNet are accessible. These pretrained models may be adjusted for particular tasks like semantic segmentation, object identification, and picture classification, or they can be utilized as feature extractors.

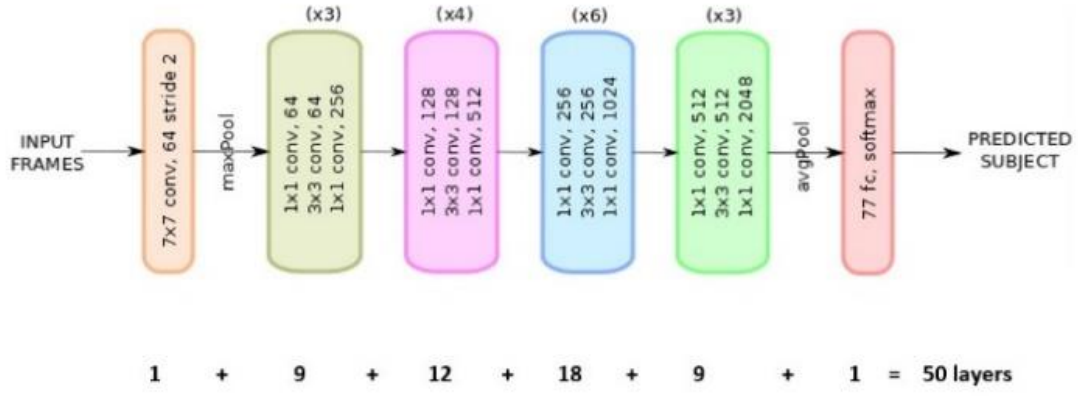


Fig. 3.5 Working of ResNet50

3.4 VGG16

The 16 weight layers that make up the VGG16 convolutional neural network architecture are arranged as follows: convolutional layers come first, then max-pooling layers, and finally fully connected layers for classification. There is a recurring trend in the architecture: max-pooling layers have 2x2 filters with a stride of 2, while convolutional layers have 3x3 filters with a stride of 1. VGG16 adds max-pooling layers with a 2x2 filter size and a stride of 2 after each set of convolutional layers. Max-pooling contributes to translation invariance and computational efficiency by reducing the spatial dimensions of the feature maps while maintaining the most crucial information. After every convolutional layer, the network is given the opportunity to learn intricate patterns and relationships in the input data by use of Rectified Linear Unit (ReLU) activation functions, which bring non-linearity into the system. VGG16 has completely connected layers at the end of the network, which are followed by a softmax activation function for classification as shown in Fig. 3.6. These layers create the final output probabilities for various classes by combining characteristics that were learnt from previous levels.

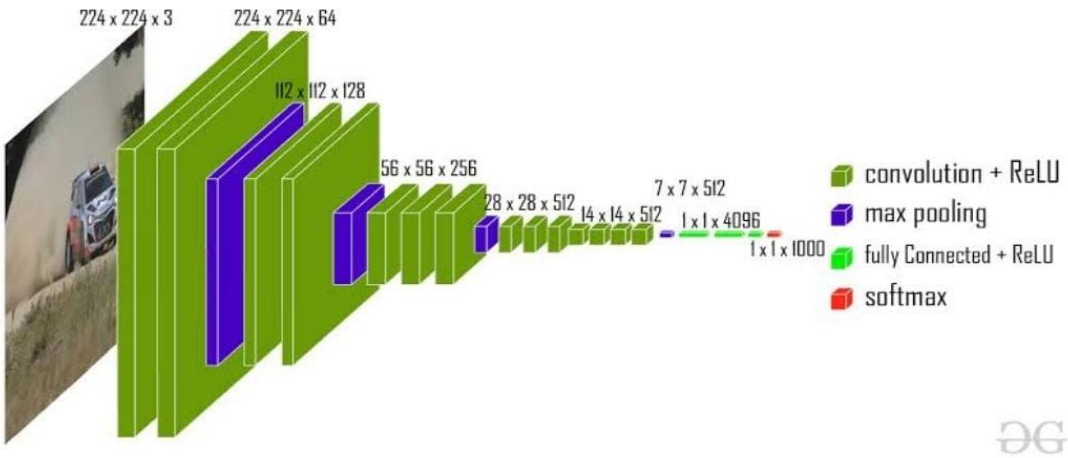


Fig. 3.6 Working of VGG16

3.5 GRAM-CAM SEGMENTATION

In normal segmentation, the goal is to partition an image into semantically meaningful regions or objects. Segmentation methods can be supervised or unsupervised and can include techniques like thresholding, edge detection, region growing, clustering, and deep learning-based approaches like semantic segmentation. Normal segmentation aims to provide a comprehensive understanding of the image content by labeling each pixel or region with a corresponding class or category, such as foreground and background or specific object classes. In computer vision and image processing, Gradient-weighted Class Activation Mapping, often known as Gram-Cam segmentation is a method that divides an image into its component parts according to its visual properties, usually with the use of deep learning algorithms. The "gram" is shorthand for the Gram matrix, a mathematical depiction of the relationship between several characteristics in the intermediate layers of a neural network. The Gram matrix is produced for the features extracted from intermediate layers of a convolutional neural network (CNN), which is a sort of pre-trained deep neural network that have been trained on a big dataset for a particular purpose like object identification or image classification. This process is known as Gram-Cam segmentation. The network can distinguish between various visual patterns or textures in the picture by examining the correlations between features in these intermediary levels. Distinct segmentation approaches may be used to detect and separate distinct regions or objects in the picture based on their visual similarities or differences once the Gram matrix has been generated. In general, Gram-Cam segmentation divides

pictures into meaningful areas or objects using mathematical representations of visual data and deep learning, which may be helpful for tasks like object recognition, scene understanding, and image understanding. The predictions on random samples are given in Fig 3.7.

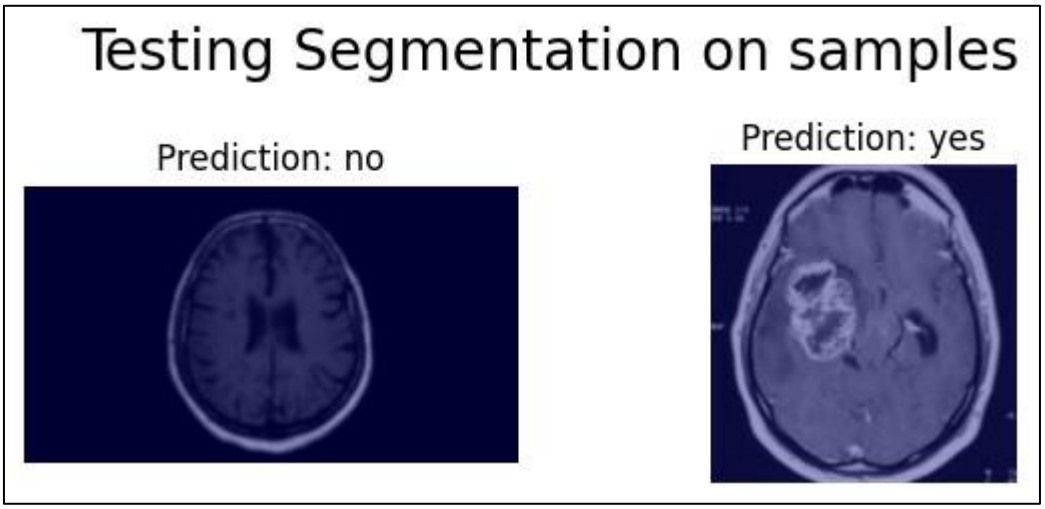


Fig 3.7 Output through Grad-Cam Segmentation

3.6 DESIGN OF MODULES

Designing modules for brain tumor detection involves breaking down the process into smaller components, each responsible for specific tasks. Here's a high-level overview of the modules typically involved in brain tumor detection:

1. **Import libraries:** Import the necessary libraries required for performing operations on the CNNs.

2. **Extract zip folders and make data frame:** The main folders are inside subfolders hence they need to be extracted. Similar to a spreadsheet or database table, data frames offer an easy way to express structured data in a tabular style. Users may now deal with datasets that have rows and columns in an understandable manner. Strong data manipulation tools, such as filtering, sorting, grouping, combining, and aggregating data, are provided by data frames. These procedures are necessary for extracting

insights from data or getting it ready for additional analysis by cleaning, converting, and summarizing it. Data scientists, analysts, and researchers in a variety of fields find data frames to be an invaluable tool in their arsenal since they are so adaptable for handling, analyzing, and visualizing data.

3. **Data imbalance check:** When a dataset's class distribution is skewed, meaning certain classes have noticeably more instances than others, it's referred to as a data imbalance. This may often be a difficulty in a lot of data mining and machine learning projects, especially those involving categorization. Machine learning algorithms may encounter difficulties when dealing with imbalanced datasets since they may give preference to the majority class while disregarding the minority class. This can result in biased models that perform poorly in predictions, particularly for the minority class.

4. **Train and validation sets:** The model is trained using the training set. The model gains the ability to predict or classify data by learning patterns and relationships in the training set. The trained model's performance is assessed using the validation set. We can predict how well the model will generalize to new, unknown data by evaluating its performance on untested data during training. When a model learns to memorize the training data instead of drawing conclusions from it, it is said to be overfitting. We may identify overfitting by assessing the model's performance on the validation set and modifying the model's regularization strategies or complexity to enhance generalization performance.

5. **Data augmentation and rescaling:** Data augmentation is the process of creating new training samples from the original data by transforming it in different ways, such as flipping, rotating, translating, or changing the contrast or brightness. By exposing the model to changes in the input data, augmentation of the data broadens the training set and can assist enhance the model's generalization performance. Rescaling entails bringing the input attributes or picture scales into line with a common distribution or range. In order to avoid any one input feature from dominating due to its magnitude and to ensure that all input characteristics contribute equally to the model's learning process, rescaling is crucial.

6. **Grey scale conversion and guassian bluring:** When a color image is converted to grayscale, it becomes a grayscale image with each pixel value denoting the amount of light present at that particular location. Typically, this ranges from 0 (black) to 255 (white). Grayscale picture conversion preserves crucial details about forms, structures, and textures while simplifying the data. By concentrating on the most important characteristics, it reduces the computing needs and can enhance the model's performance. Gaussian blurring is a filtering technique that uses a Gaussian kernel to weight the average of the pixel values within a small region in order to smooth out a picture. By reducing noise and abrupt transitions, blurring strengthens the image's resistance to changes in illumination and small abnormalities. Additionally, it aids in eliminating high-frequency information that might not be pertinent to the job, enhancing the generality of the model.

7. **Image resizing:** In computer vision applications, image scaling is a frequent preprocessing technique, especially when dealing with deep learning models such as convolutional neural networks (CNNs). Resizing is the process of altering an image's width and height while maintaining its aspect ratio. Image resizing is done for a number of reasons. Ensuring that all input photos have the same proportions requires resizing them to a consistent size. The majority of machine learning models assume that inputs have fixed dimensions, therefore this is significant. Images of a certain size are frequently used as input for deep learning models. By lowering the amount of memory and processing power required for training and inference, resizing pictures to the necessary size can increase computational efficiency. Techniques for augmenting data may include resizing. To provide more training examples and broaden the dataset's variety, photos can be enlarged to various scales, for instance. By lowering the possibility that the model would retain particular information from the training photos that might not translate well to new data, resizing images might assist avoid overfitting. A more universal representation of the input data can be produced with the use of resizing.

8. **Segmentation:** The practice of dividing a picture into many segments or areas is known as image segmentation. This is usually done on the basis of particular attributes like color, texture, intensity, or boundaries. Image segmentation has wide-ranging applications in several domains, such as satellite imaging (e.g., land cover classification), autonomous driving (e.g., object recognition), medical imaging (e.g.,

tumor detection), and more. For tasks like object recognition, scene interpretation, and image-based measurements or analysis, it is an essential preprocessing step. Complex deep learning-based techniques and basic thresholding methods are two examples of segmentation algorithms. The segmentation approach selected is determined by several aspects, including the image's nature.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 RESULTS

Accurate detection of brain tumors is paramount for effective treatment planning and improved patient outcomes. In this section, we present the results of our comparative analysis of four different methods: MobileNetV2, EfficientNet, ResNet50, and VGG16. These methods were evaluated based on various performance metrics, including detection accuracy, sensitivity, specificity, and overall classification accuracy. The training and validation accuracies of each method were recorded over multiple epochs to assess their performance in identifying brain tumors from MRI scans. Here are some common outcomes:

1. **Detection Accuracy and Model Performance Evaluation:** Primary focus of our analysis was on the detection accuracy of the four methods. Detection accuracy refers to how effectively the algorithm can identify the presence or absence of a brain tumor in MRI scans. To evaluate this, we utilized metrics such as sensitivity, specificity, and overall classification accuracy. Sensitivity is used to measure the proportion of true positive predictions out of all actual positive cases, while specificity is used to measure the proportion of true negative predictions out of all actual negative cases. Overall classification accuracy provides an overall measure of the model's performance in correctly classifying tumor and non-tumor regions.

   The training and validation accuracies of MobileNetV2, EfficientNet, ResNet50, and VGG16 were monitored and compared over multiple epochs. Figures below illustrates the training and validation accuracies of each method across epochs. As seen in the graphs, all four methods show an increase in accuracy during training, indicating the learning capability of the models. However, variations in performance are observed between the methods, with some showing faster convergence and higher accuracy than others. These differences in performance highlight the importance of selecting an appropriate neural network architecture for brain tumor detection tasks.

Overall, our evaluation of the four methods provides valuable insights into their performance in brain tumor detection. The results presented here form the basis for selecting the most suitable method for accurate and reliable detection of brain tumors from MRI scans, ultimately contributing to improved patient care and outcomes.

1.1 MobileNetV2 - The training & validation accuracies are as follows:

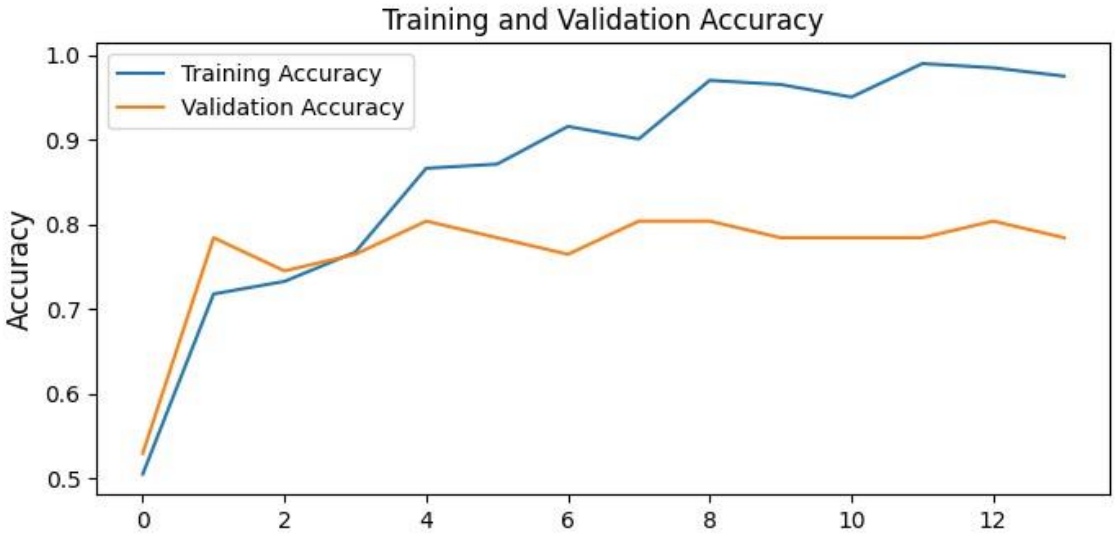


Fig 4.1 Accuracy of MobileNetV2

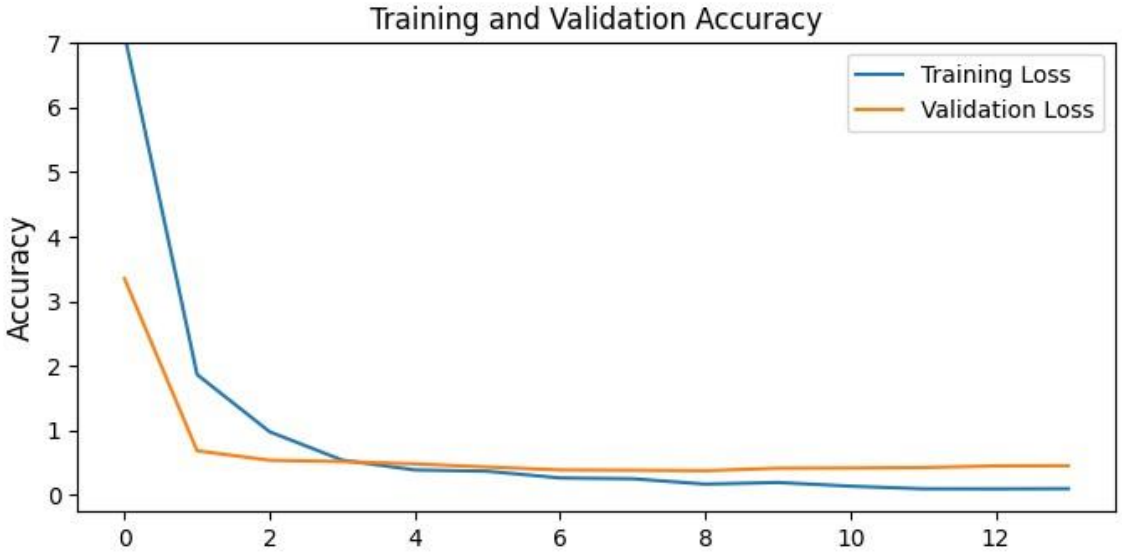


Fig 4.2 Loss of MobileNetV2

1.2 EffecientNet - The training accuracy & validation accuracy are as follows:

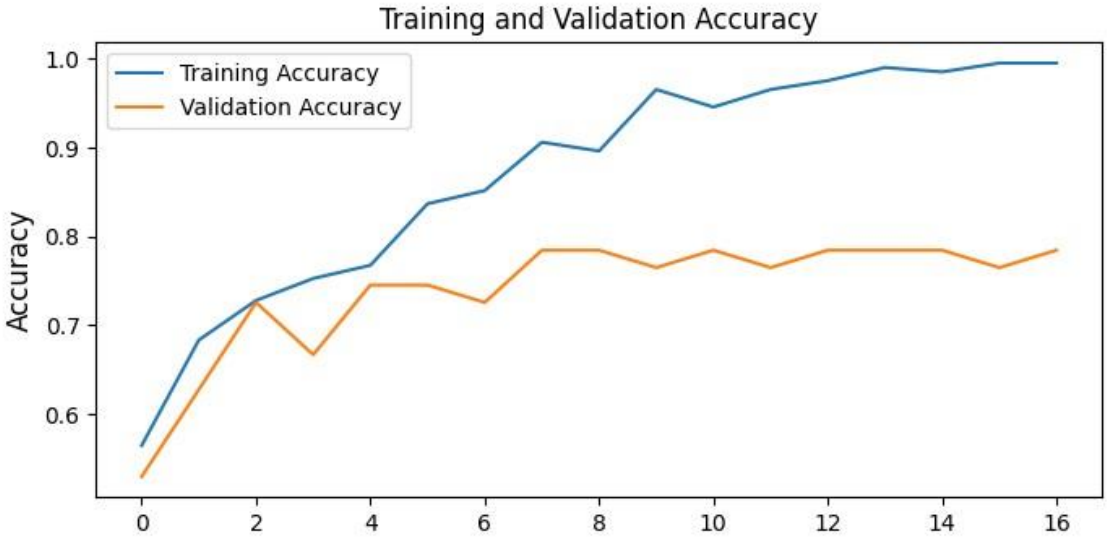


Fig 4.3 Accuracy of EfficientNet

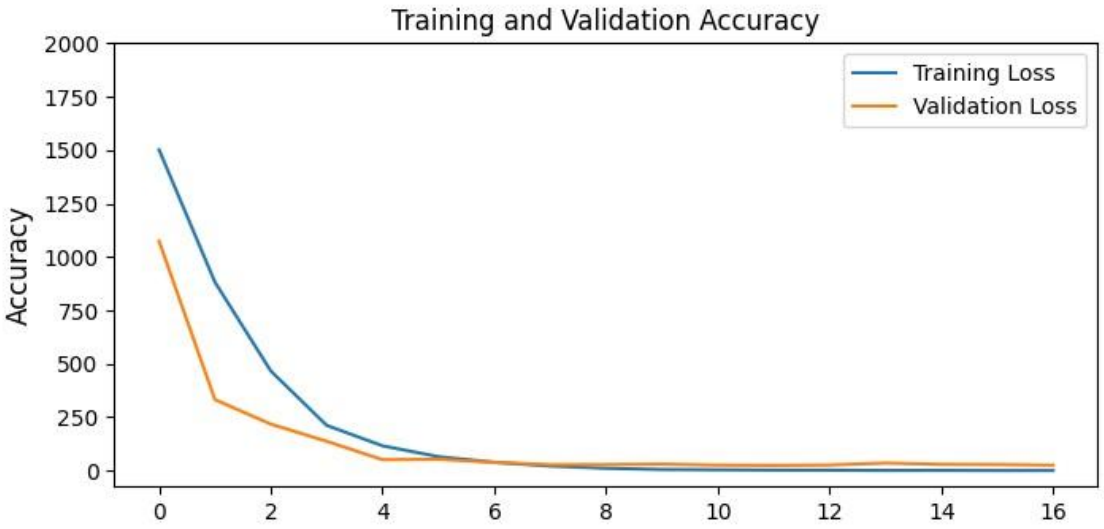


Fig 4.4 Loss of EfficientNet

1.3 RestNet50 - The training accuracy & validation accuracy are as follows:

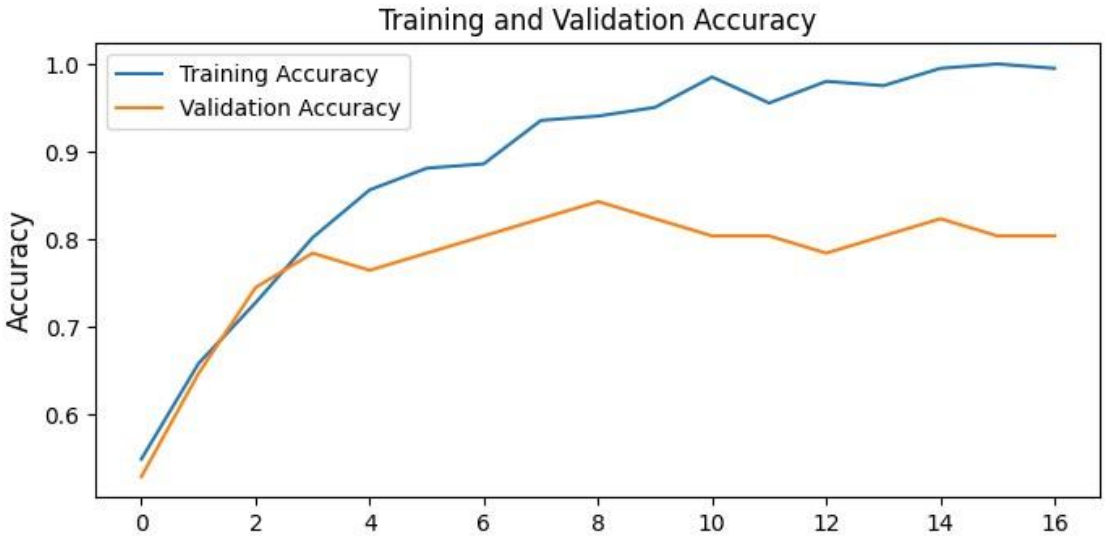


Fig 4.5 Accuracy of ResNet50

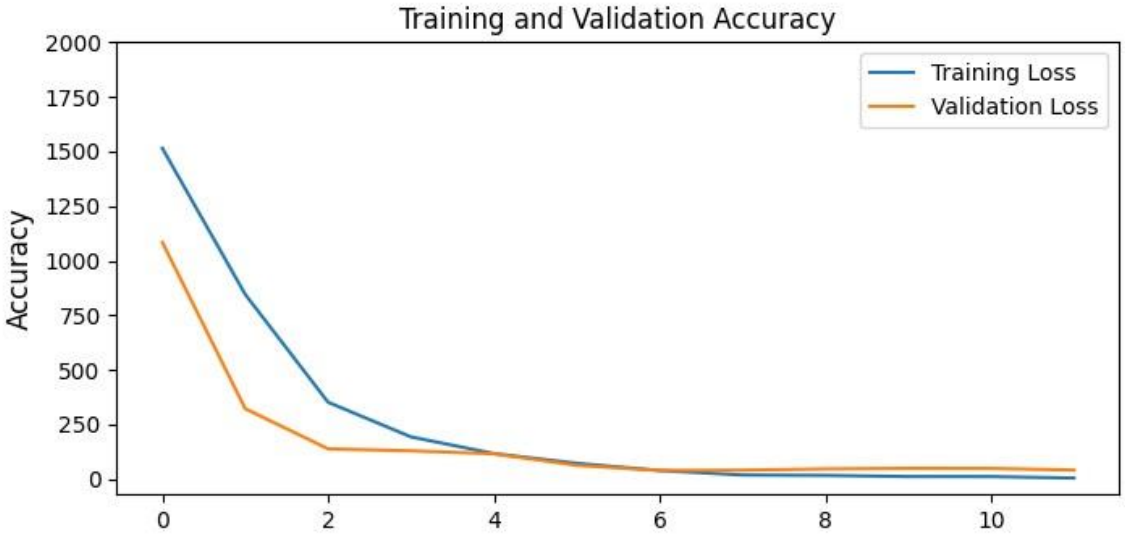


Fig 4.6 Loss of ResNet50

1.4 VGG16 - The training accuracy & validation accuracy are as follows:

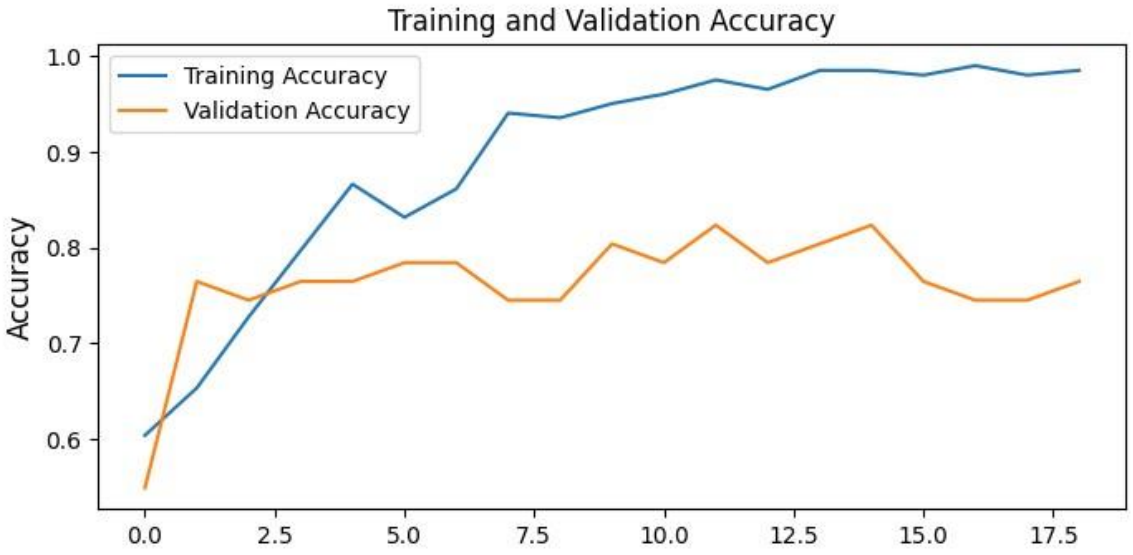


Fig 4.7 Accuracy of VGG16

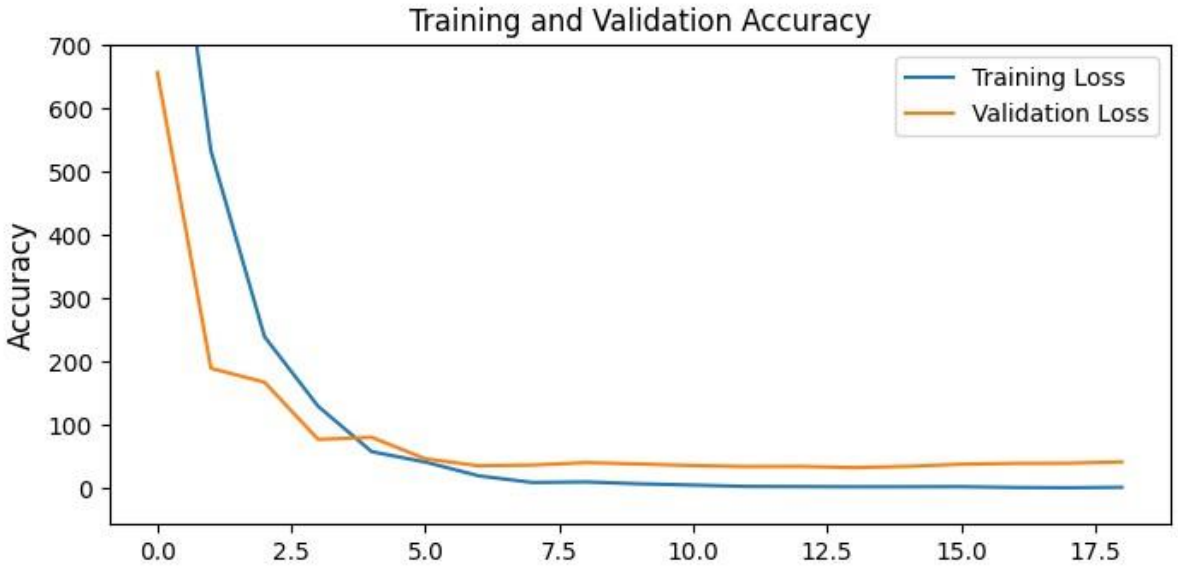


Fig 4.8 Loss of VGG16

2. **False Positives and False Negatives :** False positives occur when the algorithm incorrectly identifies a healthy region as a tumor, while false negatives occur when the algorithm fails to detect an actual tumor. Minimizing false positives and false negatives is crucial for the reliability of the detection system. The confusion matrix for the four algorithms is given below.

   For MobileNetV2, true positive is 26 which means it detects the positive cases, false positive is 5, false negative 6 and true negative is 39 which means it detects the negative cases.

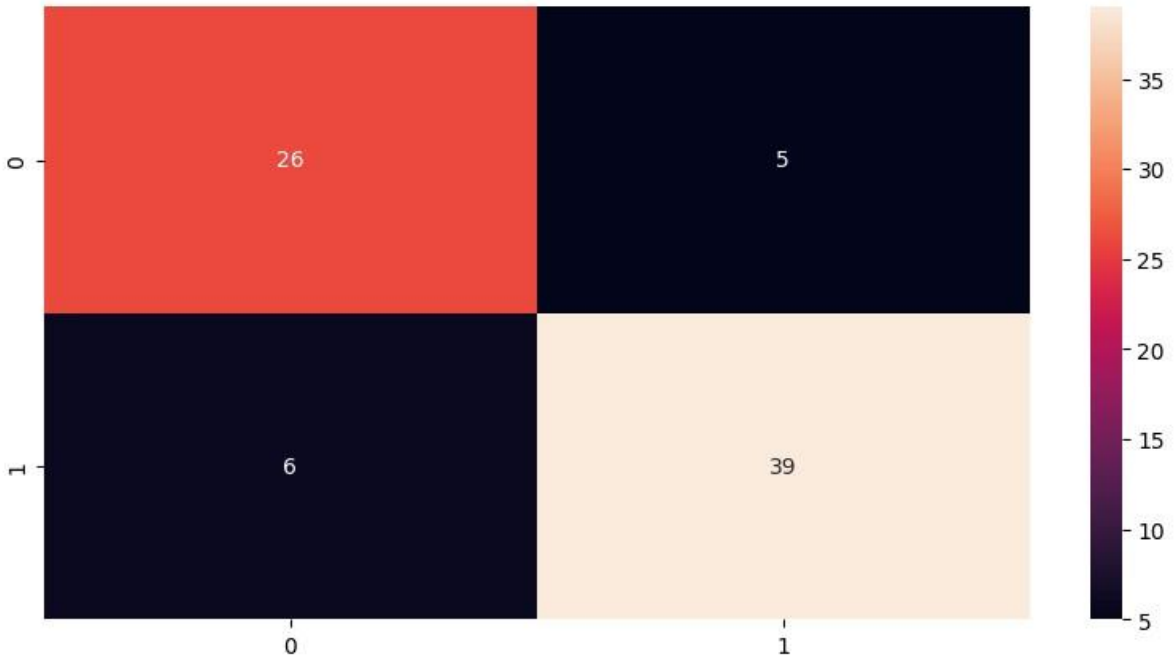


Fig 4.9 Confusion matrix of MobileNetV2

   For EfficientNet, true positive is 24 which means it detects the positive cases, false positive is 7, false negative 5 and true negative is 40 which means it detects the negative cases.

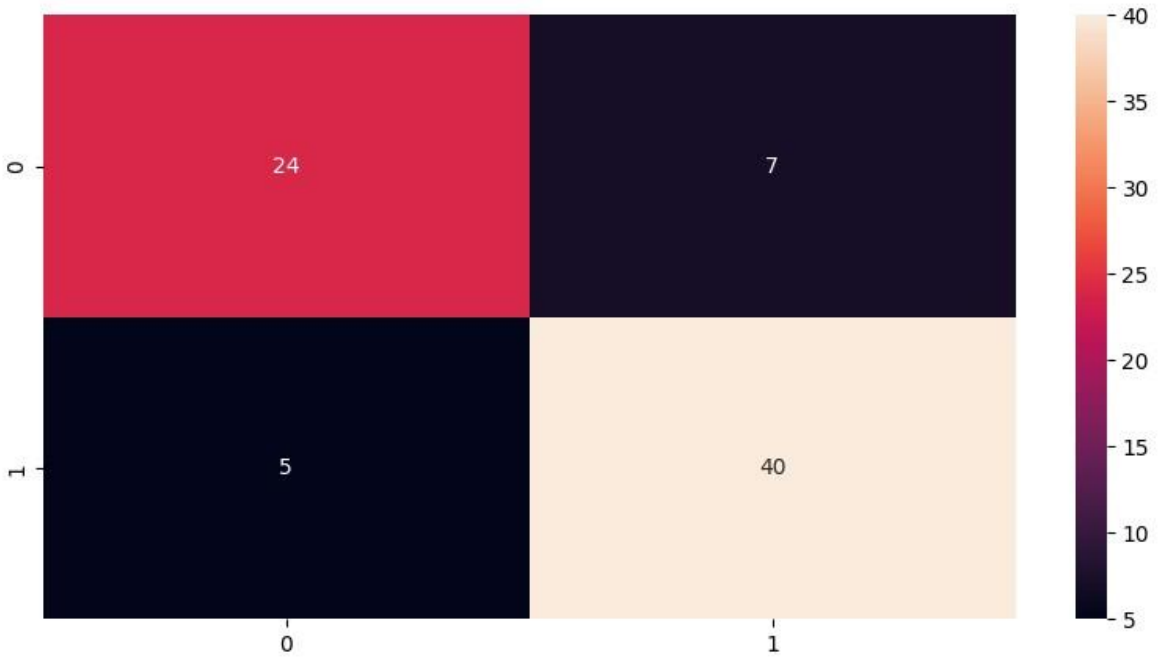


Fig 4.10 Confusion matrix of EfficientNet

For ResNet50, true positive is 22 which means it detects the positive cases, false positive is 9, false negative 4 and true negative is 41 which means it detects the negative cases.

Fig 4.11 Confusion matrix of ResNet50

For VGG16, true positive is 20 which means it detects the positive cases, false positive is 11, false negative 2 and true negative is 43 which means it detects the negative cases.

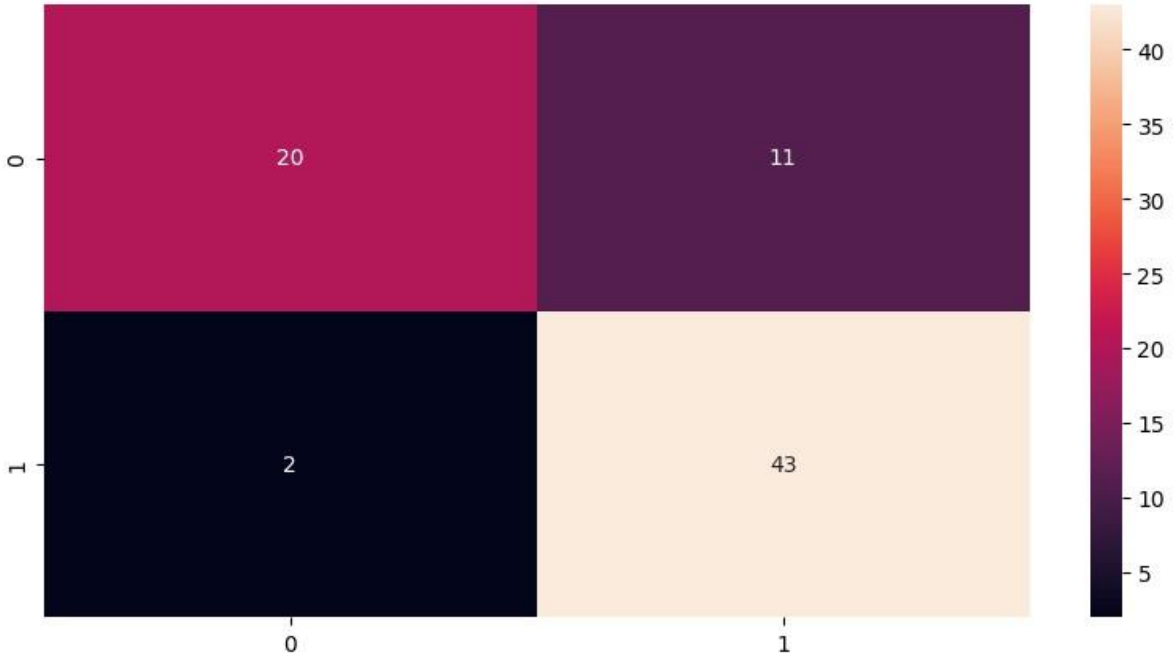


Fig 4.12 Confusion matrix of VGG16

3. **Segmentation:** In addition to detecting the presence of a tumor, many detection methods also provide segmentation results, outlining the boundaries or regions of the tumor within the image. Accurate segmentation is essential for further analysis and treatment planning.

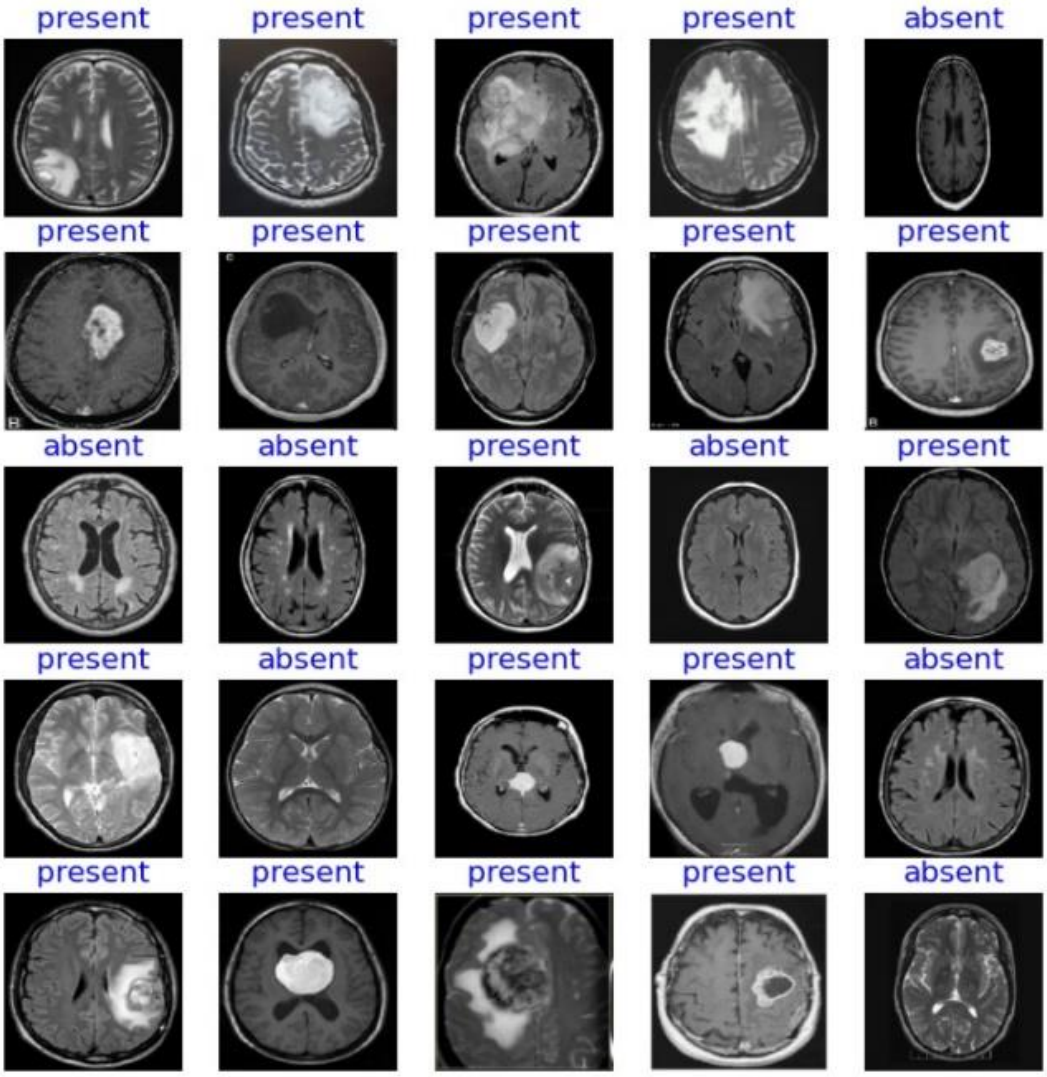


Fig. 4.13 Segmentation results of some datasets

4. **Quantification:** Depending on the approach, detection methods may also provide quantitative information about the tumor, such as its size, volume, shape, and growth rate. This information helps clinicians assess the severity of the tumor and monitor its progression over time.

5. **Speed and Efficiency:** The speed and computational efficiency of the detection method are also important factors, particularly in clinical settings where rapid diagnosis is crucial for patient care. The accuracies of different models are given in table 4.1.

Table 4.1 Table of Accuracy of Different Models

Model Name	Accuracy
MobileNetV2	86.8421052631579
EfficientNet	84.21052631578947
ResNet50	80.26315789473685
VGG16	88.1578947368421

Hence, VGG16 gives the best accuracy of 88% roughly.

Overall, the results of brain tumor detection aim to provide accurate, reliable, and actionable information to healthcare professionals for diagnosis, treatment planning, and monitoring of patients with brain tumors. These results can have a significant impact on patient outcomes and the effectiveness of treatment strategies.

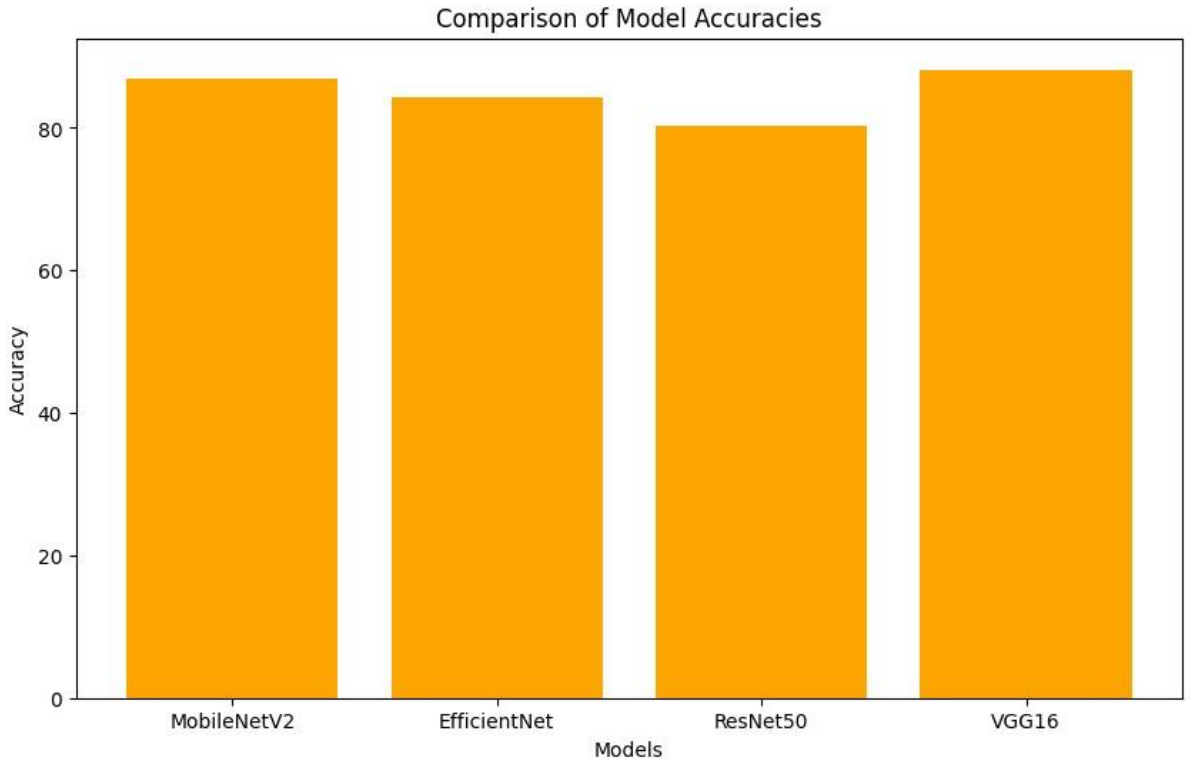


Fig 4.14 Comparison analysis of accuracies

4.2 DISCUSSIONS

Talks on brain tumor detection include a wide range of topics, such as the difficulties, developments, methods, and consequences. Convolutional neural networks, one type of deep learning technology, have demonstrated encouraging results in the identification of brain tumors. By automatically extracting pertinent elements from imaging data, these models can increase the precision of detection. Tumor identification algorithms may be made more sensitive and specific by combining data from many imaging modalities, such as CT, PET, and MRI scans. The accuracy and resilience of detection can be further increased by combining several models or detection algorithms using ensemble learning.

For prompt treatment and better patient outcomes, brain tumors must be accurately and early detected. Treatment planning decisions, such as surgery, radiation therapy, or chemotherapy, are influenced by the results of the detection, including the features, location, and size of the tumor. Detection techniques offer important data for tracking the evolution of tumors over time and evaluating the effectiveness of treatments, supporting patient prognosis and aftercare. To uphold patient confidence and adhere to legal requirements, it is vital to guarantee the confidentiality and integrity of patient data used in tumor identification algorithms. Improving healthcare outcomes worldwide requires ensuring fair access to modern detection technology, especially in areas with low resources.

In general, conversations on brain tumor detection emphasize the continuous work to create precise, effective, and clinically meaningful detection techniques to enhance patient care and results in the identification and management of brain tumors.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENT

5.1 CONCLUSION

This marks a critical turning point in medical technology as improvements in brain tumor detection have greatly improved diagnostic accuracy and patient outcomes. By combining state-of-the-art imaging modalities (MRI, CT, and PET scans) with novel machine learning algorithms and artificial intelligence, medical professionals can now efficiently and precisely diagnose brain tumors, pinpointing their location and characteristics with previously unheard-of accuracy. This game-changing combination of computational analysis and medical imaging not only speeds up diagnosis but also makes it possible to create individualized treatment plans that are specific to the needs of each patient, improving prognosis and therapeutic efficacy in the process. Additionally, the development of non-invasive techniques reduces danger and discomfort for patients, allowing for early intervention and lowering morbidity related to late-stage illness. Future developments in neuroimaging and computational medicine promise to yield even more sophisticated methods, which could completely alter the field of brain tumor diagnosis and treatment and ultimately enhance patient outcomes and quality of life across the globe.

5.2 FUTURE ENHANCEMENT

Future developments in the field of brain tumor detection include artificial intelligence algorithms in conjunction with advances in imaging technology, such as high-resolution MRI and functional imaging modalities, will make it possible to identify even the smallest anomalies early on. Furthermore, including non-invasive biomarkers into diagnostic procedures—like exosomes and circulating tumor DNA—will provide a less invasive way to track the development of tumors and their response to therapy.

REFERENCES

[1] Machiraju Jaya Lakshmi & S.Nagaraja Rao., "Brain tumor magnetic resonance image classification: a deep learning approach," Soft Computing, vol. 26, pp.6245–6253, 2022.

[2] Jiang, Y. et.al., "A Brain Tumor Segmentation New Method Based on Statistical Thresholding and Multiscale CNN," Intelligent Computing Methodologies, vol. 2, issue (3), pp. 235-245, 2019.

[3] Liu, D. et.al., "3D Large Kernel Anisotropic Network for Brain Tumor Segmentation," Neural Information Processing, pp. 444-454, 2018.

[4] Bhanothu, Y. et.al., "Detection and classification of brain tumor in mri images using deep convolutional network," International Conference on Advanced Computing and Communication Systems , pp. 248–252, 2020.

[5] Ahmed et.al., "Fine-tuning convolutional deep features for MRI based brain tumor classification," Medical Imaging, Vol. 10134, 2017.

[6] Tuhin MA.et.al., "Detection and 3d visualization of brain tumor using deep learning and polynomial interpolation," IEEE Asia-Pacific Conference on Computer Science and Data Engineering, pp. 1-6, 2020.

[7] Hu Y&Xia, Y.,"3D deep neural network-based brain tumor segmentation using multimodality magnetic resonance sequences." International MICCAI Brainlesion Workshop, pp. 423–434, 2017.

[8] 8. Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical image analysis, 42, 60-88.

[9] Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., ... & Pal, C. (2017). Brain tumor segmentation with deep neural networks. Medical image analysis, 35, 18-31.

[10] Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., ... & Lanczi, L. (2015). The multimodal brain tumor image segmentation benchmark (BRATS). IEEE transactions on medical imaging, 34(10), 1993-2024.

[11] Pereira, S., Pinto, A., Alves, V., & Silva, C. A. (2016). Brain tumor segmentation using convolutional neural networks in MRI images. IEEE transactions on medical imaging, 35(5), 1240-1251.

[12] Chen, L. C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (pp. 801-818). Springer.

[13] Bakas, S., Reyes, M., Jakab, A., Bauer, S., Rempfler, M., Crimi, A., ... & Menze, B. (2018). Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge. arXiv preprint arXiv:1811.02629.

[14] Havaei, M., Guizard, N., Chapados, N., & Bengio, Y. (2016). HeMIS: Hetero-modal image segmentation. arXiv preprint arXiv:1606.01286.

[15] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer.

[16] Chen, H., Dou, Q., Yu, L., Qin, J., & Heng, P. A. (2018). VoxResNet: Deep voxelwise residual networks for brain segmentation from 3D MR images. NeuroImage, 170, 446-455.

[17] 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.

[18] Isensee, F., Kickingereder, P., Wick, W., Bendszus, M., & Maier-Hein, K. H. (2018). Brain tumor segmentation and radiomics survival prediction: Contribution to the BRATS 2017 challenge. arXiv preprint arXiv:1802.10508.

[19] McKinley, R., Meier, R., Wiest, R., & Reyes, M. (2016). MRBrainS challenge: Online evaluation framework for brain image segmentation in 3T MRI scans. Computers in Biology and Medicine, 74, 76-88.

[20] Maier, O., Menze, B. H., von der Gablentz, J., Häni, L., Heinrich, M. P., Liebrand, M., ... & Bickelhaupt, S. (2017). ISLES 2015–A public evaluation benchmark for ischemic stroke lesion segmentation from multispectral MRI. Medical image analysis, 35, 250-269.

APPENDIX A

SOURCE CODE

Import Libraries

```
# @title Import Libraries
import os
import numpy as np
import pandas as pd
import cv2
from IPython.display import Image, display
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import callbacks
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
import matplotlib.image as img
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.losses import SparseCategoricalCrossentropy
```

Extract zip folders

```
# @title Extract zip folders
import zipfile

# Open the ZIP file
with zipfile.ZipFile('/content/Dataset.zip', "r") as zip_ref:
    # Extract all contents to the specified directory
    zip_ref.extractall("target_dir")
```

Make Dataframe

```
# @title Make Dataframe
tumor_dir='/content/target_dir/yes'
healthy_dir='/content/target_dir/no'

filepaths = []
labels= []
dict_list = [tumor_dir, healthy_dir]
for i, j in enumerate(dict_list):
    flist=os.listdir(j)
    for f in flist:
        fpath=os.path.join(j,f)
        filepaths.append(fpath)
        if i==0:
          labels.append('present')
        else:
          labels.append('absent')

Fseries = pd.Series(filepaths, name="filepaths")
Lseries = pd.Series(labels, name="labels")
tumor_data = pd.concat([Fseries,Lseries], axis=1)
tumor_df = pd.DataFrame(tumor_data)
print(tumor_df.head())
print(tumor_df["labels"].value_counts())
```

Check for class imbalance

```
# @title Check for class imbalance
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame containing class labels
class_distribution = tumor_df['labels'].value_counts()

# Visualize the class distribution
class_distribution.plot(kind='bar')
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Number of Samples')
plt.show()

# Calculate class proportions
class_proportions = class_distribution / class_distribution.sum()

# Print class proportions
print("Class Proportions:")
print(class_proportions)
```

Make a new directory for classifying into training and validation sets

```
# @title Make a new directory for classifying into training and validation sets
import os
import shutil

# Path to your original directory containing all images
original_dir = '/content/target_dir'

# Path to a new directory where you'll organize the images for Keras
keras_dir = 'new_dir/'

# Create subdirectories for 'train' and 'validation'
train_dir = os.path.join(keras_dir, 'train')
os.makedirs(train_dir, exist_ok=True)
validation_dir = os.path.join(keras_dir, 'validation')
os.makedirs(validation_dir, exist_ok=True)

# Function to move images to appropriate directories
def move_images_to_dir(original_dir, target_dir):
    for filename in os.listdir(original_dir):
        if filename.endswith('.jpg'):
            class_name = filename.split('_')[0]  # Extract class label from filename
            class_dir = os.path.join(target_dir, class_name)
            os.makedirs(class_dir, exist_ok=True)
            shutil.copy(os.path.join(original_dir, filename), os.path.join(class_dir, filename))
```

```
[ ] # Split ratio for train-validation split
split_ratio = 0.8

# Move images to train directory
move_images_to_dir(original_dir, train_dir)

# Move remaining images to validation directory
move_images_to_dir(original_dir, validation_dir)
```

Move images to train and validation sets

```
[ ] # @title Move images to train and validation sets
import os
import shutil
import random

def copy_images_with_subdirectories(source_dir, dest_dir):
    os.makedirs(dest_dir, exist_ok=True)
    for subdir in os.listdir(source_dir):
        sub_source_dir = os.path.join(source_dir, subdir)
        sub_dest_dir = os.path.join(dest_dir, subdir)
        os.makedirs(sub_dest_dir, exist_ok=True)
        for file in os.listdir(sub_source_dir):
            source_path = os.path.join(sub_source_dir, file)
            dest_path = os.path.join(sub_dest_dir, file)
            shutil.copy(source_path, dest_path)
```

```
[ ] # Define the directory containing your original images
    original_data_dir = '/content/target_dir'

    # Define the directory where you want to store your training and validation data
    base_dir = '/content/new_dir'
    os.makedirs(base_dir, exist_ok=True)

    # Define the subdirectories for train and validation data
    train_dir = os.path.join(base_dir, 'train')
    validation_dir = os.path.join(base_dir, 'validation')

    # Define the ratio of data to use for training
    train_ratio = 0.8  # 80% for training, 20% for validation

    # List all the subdirectories in the original data directory
    subdirs = os.listdir(original_data_dir)

    # Shuffle the subdirectories to ensure randomness
    random.shuffle(subdirs)

    # Split the subdirectories into training and validation sets
    train_subdirs = subdirs[:int(len(subdirs) * train_ratio)]
    validation_subdirs = subdirs[int(len(subdirs) * train_ratio):]

    # Copy images to the train directory
    copy_images_with_subdirectories(original_data_dir, train_dir)
```

Preprocessing of data begins here..

1. Data augmentation and rescaling
2. Grey scale conversion and guassian blurring
3. Image resizing

Data augmentation and rescaling

```
    # @title Data augmentation and rescaling
    import os
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from sklearn.utils.class_weight import compute_class_weight

    # Define directories
    train_dir = '/content/new_dir/train'
    validation_dir = '/content/new_dir/validation'

    # Define image data generators with data augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
```

```
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Validation data generator (only rescaling)
validation_datagen = ImageDataGenerator(rescale=1./255)

# Define batch size
batch_size = 32

# Create train and validation data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary'
)
```

```
# Compute class weights
class_weights = compute_class_weight('balanced', classes=np.unique(train_generator.classes), y=train_generator.classes)


# Define and compile the model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])

# Train the model with class weights
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
```

Grey scale conversion and guassian blurring

```
# @title Grey scale conversion and guassian blurring
import os
import cv2

# Function to process images in a directory
def process_images_in_directory(directory, output_directory):
    os.makedirs(output_directory, exist_ok=True)

    # Iterate through all files in the directory
    for filename in os.listdir(directory):
        file_path = os.path.join(directory, filename)
        if os.path.isdir(file_path):
            # If the file is a directory, recursively process images in it
            sub_output_directory = os.path.join(output_directory, filename)
            process_images_in_directory(file_path, sub_output_directory)
        elif filename.endswith('.jpg') or filename.endswith('.png'):
            # Load the image
            image = cv2.imread(file_path)

            # Convert the image to grayscale
            gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

            # Apply Gaussian blur
            blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
```

```
            # Write the processed image to the output directory
            output_path = os.path.join(output_directory, filename)
            cv2.imwrite(output_path, blurred_image)

# Directory containing the main directory with subdirectories
main_directory = '/content/new_dir'

# Output directory for processed images
output_directory = 'processed_images/'

# Process images in the main directory
process_images_in_directory(main_directory, output_directory)
```

Image resizing

```
[ ] # @title Image resizing
    import os
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from sklearn.utils.class_weight import compute_class_weight

    # Define directories
    train_dir = '/content/processed_images/train'
    validation_dir = '/content/processed_images/validation'
```

```python
# Define image data generators with data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Validation data generator (only rescaling)
validation_datagen = ImageDataGenerator(rescale=1./255)

# Define batch size
batch_size = 32

# Create train and validation data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
```

Segmentation

```python
# @title Segmentation
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout, Conv2DTranspose, concatenate

# Define U-Net architecture
def unet(input_shape=(256, 256, 1)):
    inputs = Input(input_shape)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, 3, activation='relu', padding='same')(conv4)
```

```
drop4 = Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

# Bottleneck
conv5 = Conv2D(1024, 3, activation='relu', padding='same')(pool4)
conv5 = Conv2D(1024, 3, activation='relu', padding='same')(conv5)
drop5 = Dropout(0.5)(conv5)

# Decoder
up6 = Conv2DTranspose(512, 2, strides=(2, 2), activation='relu', padding='same')(drop5)
merge6 = concatenate([drop4, up6], axis=3)
conv6 = Conv2D(512, 3, activation='relu', padding='same')(merge6)
conv6 = Conv2D(512, 3, activation='relu', padding='same')(conv6)

up7 = Conv2DTranspose(256, 2, strides=(2, 2), activation='relu', padding='same')(conv6)
merge7 = concatenate([conv3, up7], axis=3)
conv7 = Conv2D(256, 3, activation='relu', padding='same')(merge7)
conv7 = Conv2D(256, 3, activation='relu', padding='same')(conv7)

up8 = Conv2DTranspose(128, 2, strides=(2, 2), activation='relu', padding='same')(conv7)
merge8 = concatenate([conv2, up8], axis=3)
conv8 = Conv2D(128, 3, activation='relu', padding='same')(merge8)
conv8 = Conv2D(128, 3, activation='relu', padding='same')(conv8)

up9 = Conv2DTranspose(64, 2, strides=(2, 2), activation='relu', padding='same')(conv8)
merge9 = concatenate([conv1, up9], axis=3)
conv9 = Conv2D(64, 3, activation='relu', padding='same')(merge9)
conv9 = Conv2D(64, 3, activation='relu', padding='same')(conv9)
```

```
[ ]     # Output layer
        conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)

        model = Model(inputs=inputs, outputs=conv10)
        return model

    # Define loss function (dice coefficient) for segmentation
    def dice_coefficient(y_true, y_pred, smooth=1):
        intersection = tf.reduce_sum(y_true * y_pred)
        return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

    # Compile the model
    model = unet()
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[dice_coefficient])
```

```
[ ] train_images, test_images=train_test_split(tumor_df, test_size=0.3, random_state=42)
    train_set, val_set=train_test_split(tumor_df, test_size=0.2, random_state=42)
```

```
[ ] print(train_set.shape)
    print(val_set.shape)
    print(train_images.shape)
    print(test_images.shape)
```

MobileNetv2 Architechture

```
# @title MobileNetv2 Architechture
image_gen = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input)
train=image_gen.flow_from_dataframe(dataframe=train_set, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)

test=image_gen.flow_from_dataframe(dataframe=test_images, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)
val=image_gen.flow_from_dataframe(dataframe=val_set, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)
```

```
[ ] # Accessing class indices of train_generator
    classes = list(train.class_indices.keys())
    print(classes)
```

```
['absent', 'present']
```

```
def show_brain_images(image_gen):
    test_dict = test.class_indices
    classes = list(test_dict.keys())
    images, labels=next(image_gen) # get a sample batch from the generator
    plt.figure(figsize=(10,10))
    length = len(labels)
    r=25
    for i in range(r):
        plt.subplot(5,5,i+1)
        image=(images[i]+1)/2 #scale images between 0 and 1
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color="blue",fontsize=16)
        plt.axis('off')
    plt.show()
show_brain_images(train)
```

```
[ ] model=Sequential()

    model.add(Conv2D(filters=32, kernel_size=(3,2),strides=(1,1),activation='relu',padding='valid',input_shape=(244,244,3)))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(rate=0.32))
    model.add(Dense(64, activation='relu'))

    model.add(Dense(2, activation='sigmoid'))

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

```
[ ]  #Accuracy curves
     acc=History.history["accuracy"]
     val_acc=History.history["val_accuracy"]

     loss=History.history["loss"]
     val_loss=History.history["val_loss"]

     plt.figure(figsize=(8,8))
     plt.subplot(2,1,1)

     plt.plot(acc,label="Training Accuracy")
     plt.plot(val_acc, label="Validation Accuracy")

     plt.legend()
     plt.ylabel("Accuracy", fontsize=12)
     plt.title("Training and Validation Accuracy", fontsize=12)

     #Loss Curves
     plt.figure(figsize=(8,8))
     plt.subplot(2,1,1)

     plt.plot(loss,label="Training Loss")
     plt.plot(val_loss, label="Validation Loss")

     plt.legend()
     plt.ylim([min(plt.ylim()),7])
     plt.ylabel("Accuracy", fontsize=12)
     plt.title("Training and Validation Accuracy", fontsize=12)
```

```
[ ]  model.evaluate(test, verbose=1)
     pred=model.predict(test)
     pred=np.argmax(pred, axis=1)

     labels=(train.class_indices)
     labels=dict((v,k) for k,v in labels.items())
     pred2=[labels[k] for k in pred]
```

```
3/3 [==============================] - 1s 290ms/step - loss: 0.3287 - accuracy: 0.8684
3/3 [==============================] - 2s 479ms/step
```

```
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
     y_test=test_images.labels
     print(classification_report(y_test,pred2))
     acc_1 = accuracy_score(y_test, pred2)*100
     print("\nAccuracy of the model:",acc_1)
```

EfficientNet Architechture

```
[ ] # @title EfficientNet Architechture
    image_gen = ImageDataGenerator(preprocessing_function=tf.keras.applications.efficientnet.preprocess_input)
    train=image_gen.flow_from_dataframe(dataframe=train_set, x_col='filepaths',y_col='labels',
                                        target_size=(244,244),
                                        color_mode='rgb',
                                        class_mode="categorical",
                                        batch_size=32,
                                        shuffle=False)

    test=image_gen.flow_from_dataframe(dataframe=test_images, x_col='filepaths',y_col='labels',
                                        target_size=(244,244),
                                        color_mode='rgb',
                                        class_mode="categorical",
                                        batch_size=32,
                                        shuffle=False)
    val=image_gen.flow_from_dataframe(dataframe=val_set, x_col='filepaths',y_col='labels',
                                        target_size=(244,244),
                                        color_mode='rgb',
                                        class_mode="categorical",
                                        batch_size=32,
                                        shuffle=False)
```

```
model=Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,2),strides=(1,1),activation='relu',padding='valid',input_shape=(244,244,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(rate=0.32))
model.add(Dense(64, activation='relu'))

model.add(Dense(2, activation='sigmoid'))

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

```
[ ] #Accuracy curves
    acc=History.history["accuracy"]
    val_acc=History.history["val_accuracy"]

    loss=History.history["loss"]
    val_loss=History.history["val_loss"]

    plt.figure(figsize=(8,8))
    plt.subplot(2,1,1)

    plt.plot(acc,label="Training Accuracy")
    plt.plot(val_acc, label="Validation Accuracy")

    plt.legend()
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training and Validation Accuracy", fontsize=12)

    #Loss Curves
    plt.figure(figsize=(8,8))
    plt.subplot(2,1,1)

    plt.plot(loss,label="Training Loss")
    plt.plot(val_loss, label="Validation Loss")

    plt.legend()
    plt.ylim([min(plt.ylim()),7])
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training and Validation Accuracy", fontsize=12)
```

```
[ ] model.evaluate(test, verbose=1)
    pred=model.predict(test)
    pred=np.argmax(pred, axis=1)

    labels=(train.class_indices)
    labels=dict((v,k) for k,v in labels.items())
    pred2=[labels[k] for k in pred]
```

```
3/3 [==============================] - 1s 289ms/step - loss: 8.0391 - accuracy: 0.8553
3/3 [==============================] - 1s 284ms/step
```

```
[ ] from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    y_test=test_images.labels
    print(classification_report(y_test,pred2))
    acc_2 = accuracy_score(y_test, pred2)*100
    print("\nAccuracy of the model:",acc_2)
```

```
    import seaborn as sns
    plt.figure(figsize = (10,5))
    cm = confusion_matrix(y_test, pred2)
    sns.heatmap(cm, annot=True, fmt = 'g')
```

ResNet50 Architechture

```
# @title ResNet50 Architechture
image_gen = ImageDataGenerator(preprocessing_function=tf.keras.applications.resnet50.preprocess_input)
train=image_gen.flow_from_dataframe(dataframe=train_set, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)

test=image_gen.flow_from_dataframe(dataframe=test_images, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)
val=image_gen.flow_from_dataframe(dataframe=val_set, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)
```

```
model=Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,2),strides=(1,1),activation='relu',padding='valid',input_shape=(244,244,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(rate=0.32))
model.add(Dense(64, activation='relu'))

model.add(Dense(2, activation='sigmoid'))

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

```
[ ] #Accuracy curves
    acc=History.history["accuracy"]
    val_acc=History.history["val_accuracy"]

    loss=History.history["loss"]
    val_loss=History.history["val_loss"]

    plt.figure(figsize=(8,8))
    plt.subplot(2,1,1)

    plt.plot(acc,label="Training Accuracy")
    plt.plot(val_acc, label="Validation Accuracy")

    plt.legend()
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training and Validation Accuracy", fontsize=12)

    #Loss Curves
    plt.figure(figsize=(8,8))
    plt.subplot(2,1,1)

    plt.plot(loss,label="Training Loss")
    plt.plot(val_loss, label="Validation Loss")

    plt.legend()
    plt.ylim([min(plt.ylim()),7])
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training and Validation Accuracy", fontsize=12)
```

```
[ ] model.evaluate(test, verbose=1)
    pred=model.predict(test)
    pred=np.argmax(pred, axis=1)

    labels=(train.class_indices)
    labels=dict((v,k) for k,v in labels.items())
    pred2=[labels[k] for k in pred]
```

```
3/3 [==============================] - 1s 289ms/step - loss: 8.0391 - accuracy: 0.8553
3/3 [==============================] - 1s 284ms/step
```

```
[ ] from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    y_test=test_images.labels
    print(classification_report(y_test,pred2))
    acc_2 = accuracy_score(y_test, pred2)*100
    print("\nAccuracy of the model:",acc_2)
```

```
    import seaborn as sns
    plt.figure(figsize = (10,5))
    cm = confusion_matrix(y_test, pred2)
    sns.heatmap(cm, annot=True, fmt = 'g')
```

VGG16 Architechture

```
# @title VGG16 Architechture
image_gen = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input)
train=image_gen.flow_from_dataframe(dataframe=train_set, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)

test=image_gen.flow_from_dataframe(dataframe=test_images, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)
val=image_gen.flow_from_dataframe(dataframe=val_set, x_col='filepaths',y_col='labels',
                                    target_size=(244,244),
                                    color_mode='rgb',
                                    class_mode="categorical",
                                    batch_size=32,
                                    shuffle=False)
```

```
model=Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,2),strides=(1,1),activation='relu',padding='valid',input_shape=(244,244,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(rate=0.32))
model.add(Dense(64, activation='relu'))

model.add(Dense(2, activation='sigmoid'))

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

```
[ ] #Accuracy curves
    acc=History.history["accuracy"]
    val_acc=History.history["val_accuracy"]

    loss=History.history["loss"]
    val_loss=History.history["val_loss"]

    plt.figure(figsize=(8,8))
    plt.subplot(2,1,1)

    plt.plot(acc,label="Training Accuracy")
    plt.plot(val_acc, label="Validation Accuracy")

    plt.legend()
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training and Validation Accuracy", fontsize=12)

    #Loss Curves
    plt.figure(figsize=(8,8))
    plt.subplot(2,1,1)

    plt.plot(loss,label="Training Loss")
    plt.plot(val_loss, label="Validation Loss")

    plt.legend()
    plt.ylim([min(plt.ylim()),7])
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training and Validation Accuracy", fontsize=12)
```

```
[ ] model.evaluate(test, verbose=1)
    pred=model.predict(test)
    pred=np.argmax(pred, axis=1)

    labels=(train.class_indices)
    labels=dict((v,k) for k,v in labels.items())
    pred2=[labels[k] for k in pred]
```

```
3/3 [==============================] - 1s 289ms/step - loss: 8.0391 - accuracy: 0.8553
3/3 [==============================] - 1s 284ms/step
```

```
[ ] from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    y_test=test_images.labels
    print(classification_report(y_test,pred2))
    acc_2 = accuracy_score(y_test, pred2)*100
    print("\nAccuracy of the model:",acc_2)
```

```
import seaborn as sns
plt.figure(figsize = (10,5))
cm = confusion_matrix(y_test, pred2)
sns.heatmap(cm, annot=True, fmt = 'g')
```

```python
# Get the names of all layers in the model
layer_names = [layer.name for layer in model.layers]

# Print the layer names to inspect them
print(layer_names)

# Identify the correct name of the last convolutional layer
# Then use it in your function
```

Grad-Cam Segmentation

```python
# @title Grad-Cam Segmentation
#Define Some Functions :
last_conv_layer_name = "max_pooling2d_10"

from tensorflow.keras.preprocessing import image

def get_img_array(img_path, size=(224, 224)):
    img = image.load_img(img_path, target_size=size)
    array = image.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array

import matplotlib.pyplot as plt
def make_gradcam_heatmap(img_array, model = model , last_conv_layer_name = last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations of the last conv layer as well as the output predictions
    grad_model = keras.models.Model(
        model.inputs, [model.get_layer(last_conv_layer_name).output, model.output]
    )
    # Then, we compute the gradient of the top predicted class for our input image with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]
```

```python
    # This is the gradient of the output neuron (top predicted or chosen) with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()

def save_and_display_gradcam(img_path, heatmap, cam_path, alpha=0.4, view=False):
    # Load the original image
    img = cv2.imread(img_path)

    # Resize the heatmap to match the original image size
    heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

    # Convert the heatmap to the RGB color space
    heatmap = np.uint8(255 * heatmap)
```

```
[ ]     # Apply the heatmap to the original image
        heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

        # Combine the heatmap with the original image
        superimposed_img = heatmap * alpha + img * (1 - alpha)

        # Save the image with heatmap
        cv2.imwrite(cam_path, superimposed_img)

        # Display the image if view is True
        if view:
            plt.imshow(cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB))
            plt.axis('off')
            plt.show()

    def decode_predictions(preds):
        classes = ['no','yes']
        prediction = classes[np.argmax(preds)]
        return prediction

    def make_prediction (img_path , model = model , last_conv_layer_name = last_conv_layer_name , campath = "cam.jpeg" , view = False):
        img = get_img_array(img_path = img_path)
        img_array = get_img_array(img_path, size=(244 , 244))
        preds = model.predict(img_array)
        heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
        save_and_display_gradcam(img_path, heatmap , cam_path=campath , view = view)
        return [campath , decode_predictions(preds)]
```

```
from tensorflow import keras
from tensorflow.keras.preprocessing import image

def get_img_array(img_path, size=(224, 224)):
    img = image.load_img(img_path, target_size=size)
    array = image.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array
address="/content/processed_images/validation/yes/Y180.jpg"
campath , prediction = make_prediction(address, campath="123.jpeg" , view = False)
print(prediction)
test_img = plt.imread(campath)
plt.imshow(test_img)
plt.title(prediction)
plt.axis("off")
```

Trying segmentation on few random samples

```
[ ]  # @title Trying segmentation on few random samples
     # Define photos paths:

     path_array = [
         "/content/processed_images/validation/no/11 no.jpg",
         '/content/processed_images/validation/yes/Y14.jpg'
     ]
     fig = plt.figure(figsize=(7, 7))
     fig.suptitle("Testing Segmentation on samples", fontsize=20)

     # Assuming `train` is your DataFrameIterator object
     # Iterate over the DataFrameIterator object
     for i, (images, labels) in enumerate(train):
         if i >= len(path_array):
             break  # Break the loop if we have processed all paths in path_array

         ax = plt.subplot(3, 2, i + 1)
         campath, prediction = make_prediction(path_array[i], campath=f"{i}.jpeg")
         test_img = plt.imread(campath)
         plt.imshow(test_img)
         plt.title("Prediction: " + prediction)
         plt.axis("off")

         # Break the loop if we have processed all paths in path_array
         if i + 1 == len(path_array):
             break

     plt.show()
```

```
[ ]  import matplotlib.pyplot as plt

     # Define model names and their corresponding accuracies
     model_names = ['MobileNetV2', 'EfficientNet', 'ResNet50', 'VGG16']  # Example model names
     accuracies = [acc_1, acc_2, acc_3, acc_4]  # Example accuracies corresponding to the models

     # Create bar graph
     plt.figure(figsize=(10, 6))
     plt.bar(model_names, accuracies, color='orange')

     # Add title and labels
     plt.title('Comparison of Model Accuracies')
     plt.xlabel('Models')
     plt.ylabel('Accuracy')

     # Show the plot
     plt.show()
```

APPENDIX B

PAPER ACCEPTANCE

regarding paper acceptance Inbox ×

IRCCTSD SRM VDP <ircctsd@srmist.edu.in> Fri, 3 May, 21:09 (6 days ago)
to av8937, P, SHREE, Muthurasu, Sridevi, RAGHUL, VAIBAVI, A, 201001121, MOHAMED, ABINASH, P, Neelam, KARTHIKK, KALPANA, RISHI, Paavai, R, N, C, Bharathi, VISHNU, KAVYA, DEEPAK, An

Dear Authors,
We are glad to inform you that your following paper is accepted in the International Research Conference on Computing Technologies for Sustainable Development (IRCCTSD'24) for presentation.

Decision after 3 Reviews :

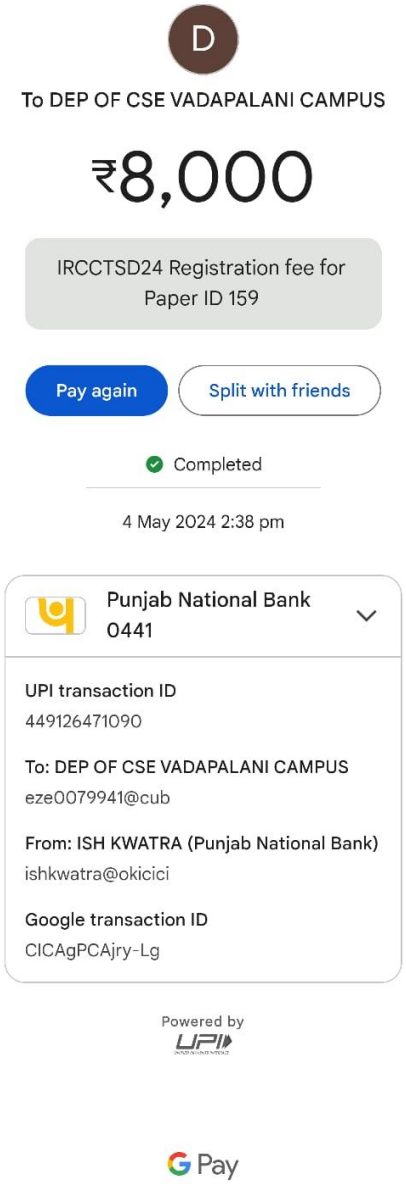
(a) Recommended for Scopus Indexed Proceedings with Springer

Registration Amount : Last date for **Registration : 4th May 2024**

	India Authors (Rs)	Foreign Authors (USD)
UG/PG Students (Scopus Proceedings)	8000	200
Academicians (Scopus Proceedings)	9000	250
Industry Participants	12000	250
Additional Authors	1000/Author	50 USD/Author

APPENDIX C

PAPER REGISTRATION



IRCCTSD'24-Invitation & Session Track Details Inbox ×

IRCCTSD SRM VDP Tue, 7 May, 18:05 (

to nagamuthukrishnan, chowdam, me, ISH, Angelin, sakethsreeram7, rayankiroopamchowdary, 1NH20AI042, chengirohith, 201001110, 201001088, sandhiya.m, sampr

Dear Authors,
Greetings from SRMIST, Vadapalani, Chennai.

All authors are invited to join and grace the inauguration ceremony on **9th May 2024 at 9.00am at C-Block, auditorium, SRMIST, vadapalani, Chennai**.

The track details and WhatsApp link of all accepted papers is attached with this for your kind information.

Online session Etiquette
1. Join the meet 10 minutes before the session
2. Rename yourself with paper id
3. Enable camera during presentation
4. Ensure good internet connection

Looking forward to having your presence on **9th May 2024 f**or the inauguration ceremony

APPENDIX D

PLAGIARISM REPORT

swati_Major Project Report (5).docx

ORIGINALITY REPORT

8%	6%	6%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.univ-eloued.dz Internet Source	<1%
2	Shubhangi Solanki, Uday Pratap Singh, Siddharth Singh Chouhan, Sanjeev Jain. "Brain Tumor Detection and Classification using Intelligence Techniques: An Overview", IEEE Access, 2023 Publication	<1%
3	Submitted to University of Hull Student Paper	<1%
4	Submitted to University of Lancaster Student Paper	<1%
5	bmjopen.bmj.com Internet Source	<1%
6	www.arxiv-vanity.com Internet Source	<1%
7	ebin.pub Internet Source	<1%
8	mdpi-res.com Internet Source	

		<1 %
9	"Advances in IoT and Security with Computational Intelligence", Springer Science and Business Media LLC, 2023 Publication	<1 %
10	ses.library.usyd.edu.au Internet Source	<1 %
11	tudr.thapar.edu:8080 Internet Source	<1 %
12	"Medical Image Computing and Computer-Assisted Intervention – MICCAI 2017", Springer Science and Business Media LLC, 2017 Publication	<1 %
13	Submitted to University of Wales Institute, Cardiff Student Paper	<1 %
14	cse.anits.edu.in Internet Source	<1 %
15	fastercapital.com Internet Source	<1 %
16	"Medical Image Computing and Computer-Assisted Intervention – MICCAI 2017", Springer Nature, 2017 Publication	<1 %

17	Submitted to University of Surrey Student Paper	<1 %
18	"Intelligent Computing Methodologies", Springer Science and Business Media LLC, 2018 Publication	<1 %
19	Pushparaj. E, Arun. A. "Exploring the developments in brain MRI classification-Review study, challenges and future direction", 2023 International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE), 2023 Publication	<1 %
20	Archana Potnurwar, Abha Chaudhari, Nitanshu Burbure, Kritik Wankhede, Kartik Badkhal, Anisha Parshatwar. "Brain Tumor Segmentation on MRI Images", 2023 11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP), 2023 Publication	<1 %
21	Submitted to Berlin School of Business and Innovation Student Paper	<1 %

22	Submitted to Liverpool John Moores University Student Paper	<1 %
23	Submitted to University of East London Student Paper	<1 %
24	Submitted to University of Greenwich Student Paper	<1 %
25	Toufique A. Soomro, Lihong Zheng, Ahmed J. Afifi, Ahmed Ali, Shafiullah Soomro, Ming Yin, Junbin Gao. "Image Segmentation for MR Brain Tumor Detection Using Machine Learning: A Review", IEEE Reviews in Biomedical Engineering, 2023 Publication	<1 %
26	deepai.org Internet Source	<1 %
27	link.springer.com Internet Source	<1 %
28	"Scientific Abstracts and Sessions", Medical Physics, 2020 Publication	<1 %
29	Submitted to University of Hertfordshire Student Paper	<1 %
30	Zilong Hu, Jinshan Tang, Ziming Wang, Kai Zhang, Lin Zhang, Qingling Sun. "Deep learning for image-based cancer detection	<1 %

and diagnosis–A survey", Pattern Recognition, 2018
Publication

31	res.mdpi.com Internet Source	<1%
32	www.diseasemaps.org Internet Source	<1%
33	www.frontiersin.org Internet Source	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		