City, University Of London

MSc Data Science

Project Report

2023

# The Detection of Brain Tumors through Convolutional Neural Networks (CNNs) utilizing Residual Networks (ResNets) and Visual Geometry Group (VGG) Architectures

Researcher: Uthman Bello Maigari

Supervised by: Dr Artur Garcez

Date: 9/12/2023

# Declaration

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: *Uthman Bello Maigari*

# LIST OF ACRONYMS

| | |
|---|---|
| **MRI** | Magnetic Resonance Imaging |
| **VGG** | Visual Geometric Group |
| **CNN** | Convolutional Neural Network |
| **DL** | Deep Learning |
| **IPCV** | Image Processing Computer Vision |
| **DIP** | Digital Image Processing |
| **TL** | Transfer Learning |
| **RELU** | Rectified Linear Unit |
| **DNN** | Deep Neural Network |
| **ResNET** | Residual Network |
| **ADAM** | Adaptive Moment Estimation |
| **CCE** | Categorical Cross Entropy |
| **BCE** | Binary Cross Entropy |

# ABSTRACT

A brain tumor can be detected using MRI images as the first input. It would be very beneficial for the healthcare system if autonomous detection systems were developed. The digital nature of MRIs has enabled automated classification methods to be applied to them through image processing. The purpose of this study was to investigate the use of transfer learning and convolutional neural networks for classifying MRI images of the brain. Transfer learning techniques along with MRI image processing and classification techniques were used to develop a model for identifying brain tumors.

Several convolutional neural network architectures are investigated in this study, including VGG-16 and ResNet-50, when it comes to predicting brain tumors. By analyzing key metrics such as accuracy, precision, recall, and F1-score, the study provides an in-depth analysis of their relative performance. Using MRI images of the brain, both architectures are rigorously evaluated. A comprehensive analysis of classification results is conducted after the training and validation phases, as well as the tuning of hyperparameters.

The accuracy of VGG-16 in classifying brain tumors was 81.77%, which is very good. A remarkable 97.39% accuracy was achieved by ResNet-50 when compared to VGG-16, demonstrating its robustness and robustness in identifying tumors accurately. As well as exhibiting higher precision and recall, ResNet-50 consistently performed better for classes of tumors and non-tumors. Each architecture's strengths and weaknesses were analyzed in detail by analyzing confusion matrices. Additionally, other performance metrics are evaluated within the project based on experiments.

In the next phase, brain tumor images will be localized by ResUnet after the current phase has been completed. By precisely identifying and defining the regions of interest in the medical images, the project's capability to detect and analyze brain tumors is further enhanced.

A comparative analysis of VGG-16 and ResNet-50 in the context of brain tumor identification is presented in this research, which contributes to the ongoing discussion of optimal neural network architectures for medical image classification tasks.

*Keywords: MRI images, brain tumor, deep learning, CNN, VGG, ResNet, ResUnet, Machine Learning, Artificial Intelligence.*

# TABLE OF CONTENT

# CHAPTER 1

# Introduction

## 1.1 Background

Recent advances in medical imaging have proven particularly effective in detecting and classifying brain tumors. Increasingly, medical professionals are in need of accurate and efficient tools to aid them in diagnosing brain tumors as technology evolves. The purpose of this project is to explore the use of deep learning techniques in the classification of brain tumors from magnetic resonance imaging (MRI) data, specifically to examine the performance of VGG-16 and ResNet-50 architectures and construct a segmentation ResUnet to pinpoint the location of brain tumors.

Brain tumors have the highest death rate of any cancer in the human body, according to the statistics. According to the International Agency for Research on Cancer (IARC), more than 1,000,000 people are diagnosed with brain tumors every year around the world, and the mortality rate is ever increasing. Among children and adults younger than 34 years, it is the second most common cause of death related to cancer.

Deep learning allows for the easy identification of brain tumors based on images from MRI scans. MRI scans provide high-resolution scans of different brain tissues. By developing image processing technologies, early detection and accurate categorization can be achieved.

In image classification, images or portions of images are classified according to the information retrieved during image processing. Computer vision uses different categorization methods; convolution neural networks are the most advanced. Image classification problems are typically solved using convolution neural networks (CNNs).

## 1.2 Brain Tumor

Human brains are complex and functional, and there are many medical conditions that can profoundly impact their intricate functioning. Cancerous brain tumors are one of these conditions that present a formidable challenge. There are several types of brain tumors, which are defined as abnormal growths of cells within the brain or central spinal column. In both benign (noncancerous) and malignant (cancerous) forms of cancer, a person faces unique challenges and risks.

The following types of brain tumors are encountered in the context of this project:

1. Gliomas: In the most common type of brain tumor, gliomas, neurons receive structural support from glial cells. Astrocytomas are classified into three subtypes: oligodendrogliomas, ependymomas, and astrocytomas.

2. Meningiomas: The majority of meningiomas arise from the tissues covering the brain and spinal cord (meninges).

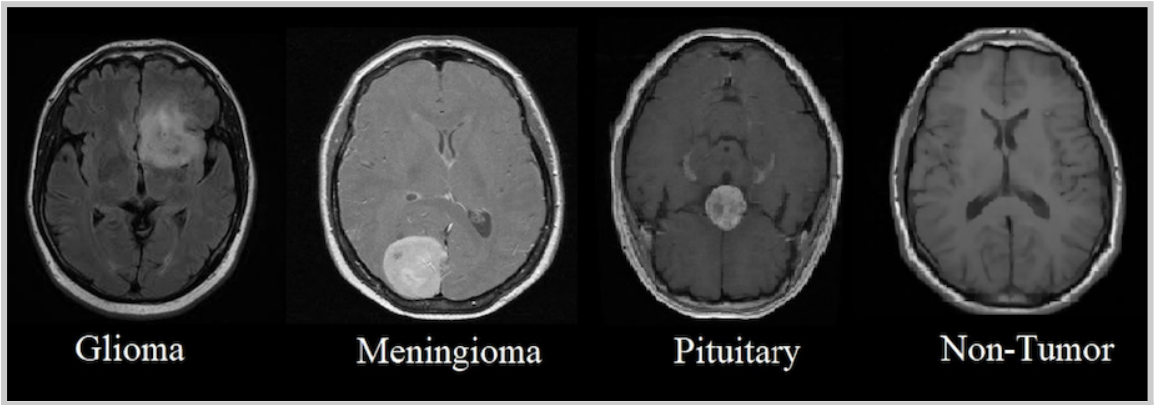



*Fig 1.1: shows the Brain MRI for Gliomas and Meningioma*

## 1.2 Motivation

Identifying brain tumors uses traditional methods, which presents challenges. Therefore, this research addresses these challenges. Manual interpretation is common in conventional approaches, leading to subjectivity and possible errors. As a result of using deep learning models, the diagnostic process can be significantly improved in terms of accuracy, speed, and objectivity for identifying and categorizing brain tumors.

It is time-consuming to detect tumors in medical images since human judgment is required. CT scans, MRIs, PET scans are examined by radiologists and specialized doctors who make treatment recommendations based on their findings. There is a lot of time involved in this process. By automating medical image analysis, we can reduce the time and effort needed to complete this task as well as the workload of a human.

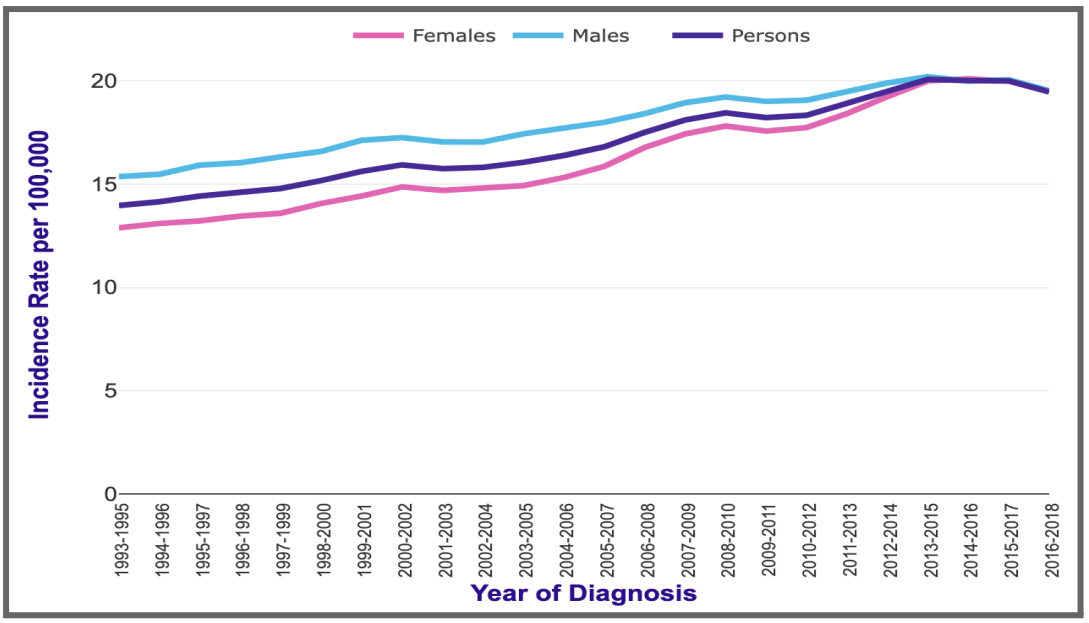



*Fig 1.2: shows Brain tumor trends over time.*

Researchers at Cancer Research UK [6] reported that "the number of females and men with brain, other CNS, and intracranial tumors increased by 39% between 1993-1995 and 2016-2018. Between 1993-1995 and 2016-2018, the UK incidence rate of brain, other CNS and intracranial tumors increased by 51% for females". There was a 27% increase in the incidence rate of brain, other CNS, and intracranial tumors in the UK for males between 1993-1995 and 2016-2018.

Females and males together have experienced an 11% increase in brain, CNS and intracranial tumor incidence rates over the last decade in the UK (between 2006-2008 and 2016-2018). There was a 16% increase in female AS incidence rates, and a 6% increase in male AS incidence rates.

## 1.3 Statement Of The Problem

The inherent complexity of medical images makes it difficult to accurately identify and classify brain tumors. A number of factors contribute to the development of advanced computational models. These factors include subtle variations in tumor characteristics, image noise, and the need to diagnose tumors in real time. Our objective is to contribute to solutions to these challenges by exploring deep learning architectures in this research.

Accuracy is essential for accurate classification of medical images. The use of a computer-based strategy by radiologists will enable them to make better decisions. The combination of image processing technology and a deep learning algorithm facilitates the identification of brain tumors. In many cases, conventional MRI can be helpful in diagnosing brain tumors, but it should not be relied upon heavily, especially in the case of specific types

of cancer. In the case of a discrepancy between the preoperative diagnosis and the report, neurosurgeons should consult with the neuroradiologists.

In comparison to experienced doctors 79 percent accuracy, machine learning approaches can identify cancer with 91 percent accuracy. In order to identify and categorize brain tumors, Deep Learning and machine learning approaches can be quite helpful. In order to create deep learning models, many internet datasets are available. In order to overcome this issue, this project used images collected from TCIA (Cancer Imaging Archive). For the purpose of detecting brain tumors with the most advanced deep learning-based models. In order to address this issue and support specialists, a model for detecting brain tumors is necessary.

## 1.4 Research Questions

In this project I aim to answer the following key research questions as a guide:

- In terms of brain tumor classification, how do VGG-16 and ResNet-50 architectures compare?
- How do the strengths and weaknesses of each architecture differ when applied to brain tumor image analysis?
- What contribution can this study make to the advancement of methods for detecting brain tumors?

## 1.5 Objective

When it comes to classifying brain tumors, it mostly involves assessing and contrasting the VGG-16 and ResNet-50 designs. The following are some particular objectives:

- Training and using VGG-16 and ResNet-50 models included using an MRI dataset of carefully selected brain images.
- reviewing the accuracy, precision, and recall of the models in order to gauge their performance.
- Examining and evaluating the outcomes to determine the advantages and disadvantages of each architecture.
- An algorithm for identifying brain tumors will be created by utilizing digital image processing and transfer learning approaches for classification.

## 1.6 Thesis Structure

Following is the structure of the thesis:

- Chapter 2: Background and Related papers
  There are several fundamental concepts covered in this chapter, including digital images, artificial intelligence, deep learning, medical images, etc.

- Chapter 3: Methodology
  A description of the materials and methodologies used in the project is provided in this chapter. Analysis and classification are carried out using CNN, VGG-16, and ResNet-50 as the key techniques. Also the ResUnet model was also used to build segments and locate the brain tumor.

- Chapter 4: Experimental Results and Evaluation
  Here, the dissertation concludes its research process with findings, analyses, and interpretations of the results derived from applying the proposed methodology.

- Chapter 5: Comparative Analysis with Related Project
  In this section, I will analyze my results and performance in comparison with those of similar projects and papers published in the past. In order to assess how similar or different the work done by others is.

- Chapter 6: Discussion and Conclusion
  It is in the final part of the thesis in which I discuss how I constructed a segmentation ResUnet to pinpoint the location of brain tumors. The dissertation concludes with a comprehensive conclusion that summarizes the entire work and provides a broad outline of what is to come.

# CHAPTER 2

# Background and Related papers

## 2.1 Introduction

Medical imaging has been transformed by artificial intelligence (AI), which has revolutionized diagnosis and detection processes. Artificial neural networks have been proven to match or even surpass the ability of human radiologists to identify cancer signs and other conditions through extensive research. In this chapter, we provide a detailed overview of Convolutional Neural Networks (CNNs) and an overview of deep learning, which forms the basis of my project.

## 2.2 Artificial Intelligence

A number of definitions of artificial intelligence (AI) have been proposed over the past decades, [7] one of which is as follows:

“Artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. While AI is an interdisciplinary science with multiple approaches, advancements in machine learning and deep learning, in particular, are creating a paradigm shift in virtually every sector of the tech industry.” [7]

According to this paragraph, artificial intelligence (AI) is a multidisciplinary field concerned with the creation of intelligent machines and the development of computer programs that display intelligent behavior. A major aspect of AI is the attempt to comprehend how humans perceive, learn, reason, and make decisions. This involves understanding and simulating human intelligence. Overall, artificial intelligence does study human intelligence to a certain extent, but its scope extends beyond imitation of biological observation. An intelligent machine or program is one which exhibits intelligent behavior and is the result of scientific and engineering efforts to achieve this goal through the use of a variety of techniques and disciplines.

## 2.3 Machine Learning

Based on Simplilearn "Machine learning is a discipline of computer science that uses computer algorithms and analytics to build predictive models that can solve business problems." [8]

Machine learning, according to McKinsey & Co., involves analyzing data without following rules. A computer program learns from experience E upon completing tasks in some class T and measuring performance with performance measure P if its performance at tasks in T improves as a result of experience E. According to Tom Mitchell, "A computer program learns from experience E by improving its performance as measured by performance measure P." [8]

## 2.4 Machine Learning approaches

The following section provides more information about the different types of machine learning;

### 2.4.1 Supervised Learning

Supervised learning utilizes data that is already labeled, thus providing you with an understanding of the target variable. By analyzing past data, systems are able to predict future outcomes. The model must be trained with at least one input variable and one output variable. Linear regressions, logistic regressions, support vector machines, Naive Bayes, and decision trees all fall under the supervised learning category.

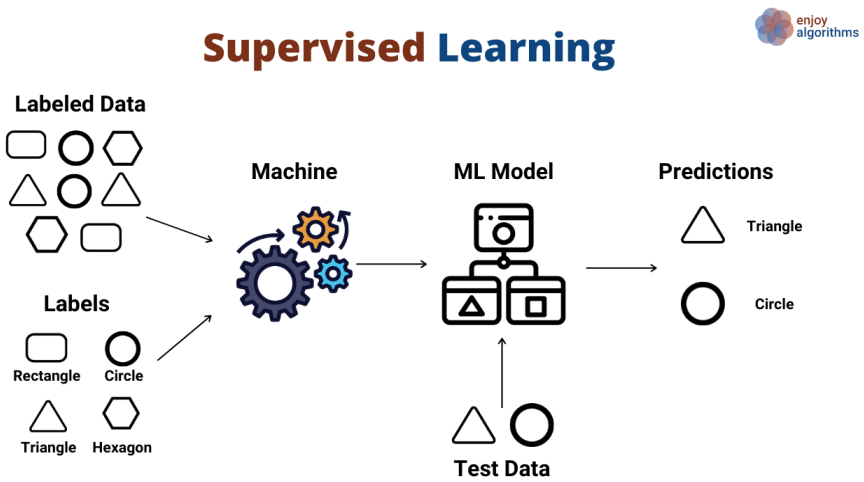



*Fig 2.1: shows the supervised learning demonstration*

### 2.4.2 Unsupervised Learning

Data is unlabeled in unsupervised learning algorithms, so they can discover patterns independently. By analyzing data input, the systems can identify hidden features. A better understanding of the data allows for more obvious patterns and similarities to emerge. The k-means clustering algorithm, hierarchical clustering, and anomaly detection are some types of unsupervised learning.

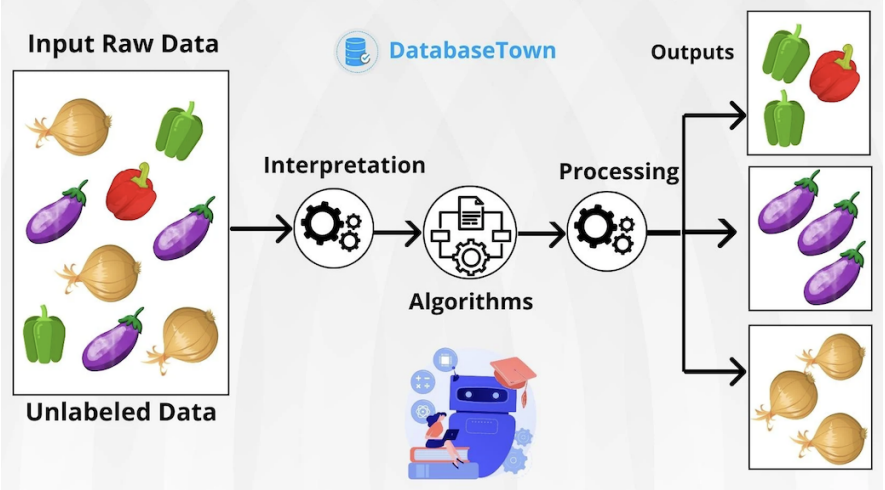



*Fig 2.2: shows the Unsupervised learning demonstration*

### 2.4.3 Reinforcement Learning

An agent is trained to perform a task in an uncertain environment through reinforcement learning. Observations and rewards are sent to the agent from the environment by the environment, and actions are sent in return. Depending on how successful the action is at achieving the task goal, the reward will be determined.

## 2.5 Deep Learning

The goal of deep learning is to develop algorithms that mimic the structure and function of the brain. The algorithms of deep learning can handle enormous amounts of structured and unstructured data. Artificial neural networks play a key role in deep learning's decision-making process.

Machine learning vs deep learning is mainly a matter of how data is presented to the machine. Artificial neural networks are used in deep learning networks, while machine learning algorithms require structured data. Neural networks have different architectures that are suitable for different data types and tasks, such as multilayer perceptrons, convolutional neural networks, and recurrent neural networks. In these architectures, there are many layers stacked on top of one another in a cascading fashion. For deep learning to be implemented successfully, stochastic optimization algorithms must be designed well, initialization techniques must be appropriate, and structures must be carefully chosen. Some applications

of deep learning include; cancer tumor detection, captionbot, image coloring, object detection, music generation, and music generation.

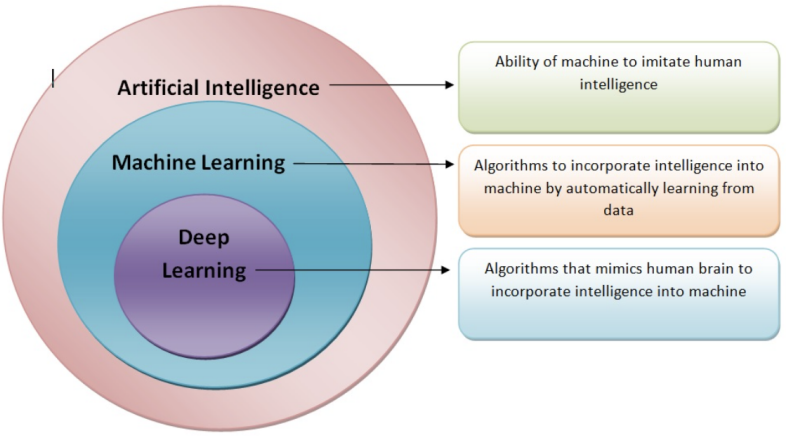



*Fig 2.3: shows the demonstration of AI, ML and DL*

There are several types of deep neural networks

- Convolutional Neural Network (CNN) - CNNs are a type of deep neural network that typically analyzes images (which I utilize for this project).
- Recurrent Neural Network (RNN) - This network uses sequential information to build a model. In many cases, it's more effective when it comes to models that have to remember past data.

## 2.6 Medical Imaging

All forms of radiation are used in medical imaging, and technical systems are designed to extract clinically relevant information, which is then presented as images. There are a variety of medical images, from X-rays of the chest to functional magnetic resonance imaging (fMRI), which displays temporal phenomena. [9]

### 2.6.1 Medical imaging types

Medical imaging can be classified into the following types:

### 2.6.2 X-rays

The use of X-rays in medical imaging began almost immediately after they were discovered. X-ray machines use radiation to create images of your body by passing it through. There is a difference in the amount of radiation absorbed by different parts of your body. X-rays pass

through soft tissues and lungs less efficiently than they do through teeth and bones, so they appear darker in an X-ray. On the images, teeth and bones appear whiter.

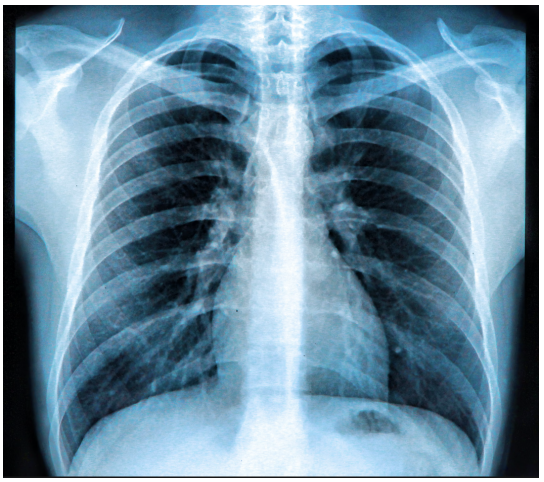


*Fig 2.4: shows an Image of a X-ray*

### 2.6.3 Ultrasound and MRI (Magnetic Resonance Imaging)

In this type of imaging, there are no X-rays or radioactive substances used. Various conditions can be diagnosed using these tests, but not all of them. In the case of these types of imaging, your imaging specialist will always consider them if they are an option for you.

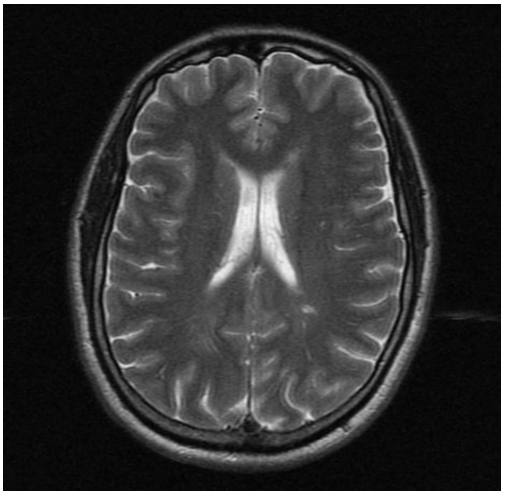


*Fig 2.5: shows an Image of MRI*

## 2.7 Neural Networks Architecture

Based on dremio, "Neural Network Architecture refers to the design and structure of an artificial neural network (ANN), which is a **machine learning** model inspired by the human

brain. It defines how the network's layers, nodes, and connections are organized to process and analyze data." [10]

There are various layers of nodes called neurons in neural networks, each of which processes input data and passes the results on to the next layer. To produce an output, neuron inputs are applied to a mathematical function, which is then fed into the next layer. In neural networks, the architecture determines their complexity, capacity, and functionality.

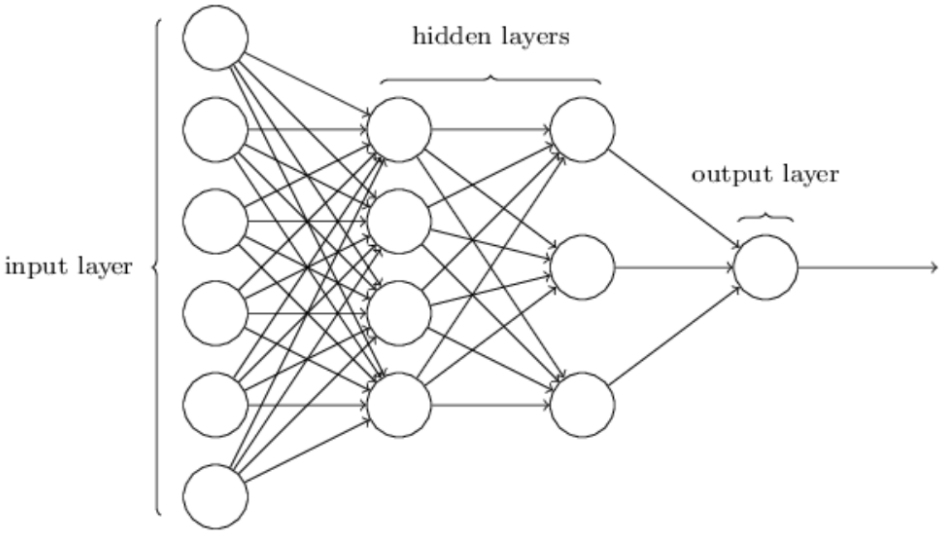



*Fig 2.6: shows general structure of an ANN*

## 2.8 Convolutional Neural Network (CNN)

In many ways, Convolutional Neural Networks resemble ordinary Neural Networks from the previous section: they are made up of neurons with learnable biases and weights. Neurons receive some inputs, perform a dot product, and optionally follow it with nonlinearity. [11]

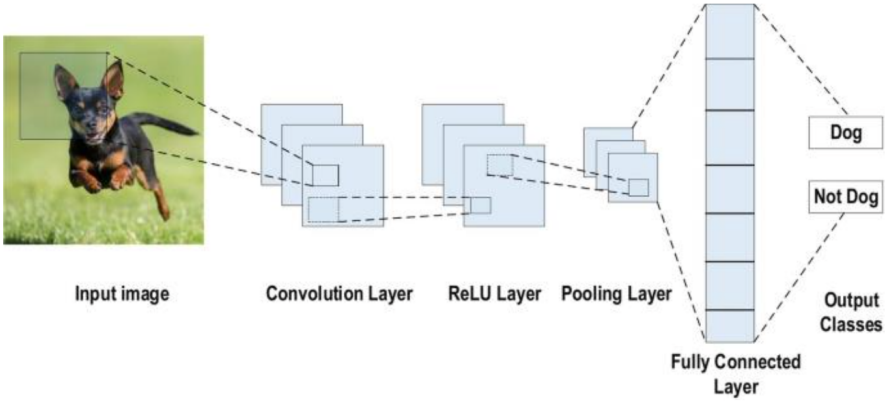



*Fig 2.7: shows CNN architecture for image classification*

## 2.9 Basic Architecture of CNN

CNN has the following layers as its basic architecture:

- Convolution layer
- RELU layer
- Pooling layer
- Fully connected layer
- Output layer (soft-max).

### 2.9.1 Convolutional Layer

In a convolutional layer, input data is filtered or reshaped using a set of 'filters' or 'kernels'. As the layers become deeper, each filter can detect a broader range of features and patterns, including edges, corners, and more complex shapes. The filters create a map representing the areas where those features were found as they move across the image. [14]

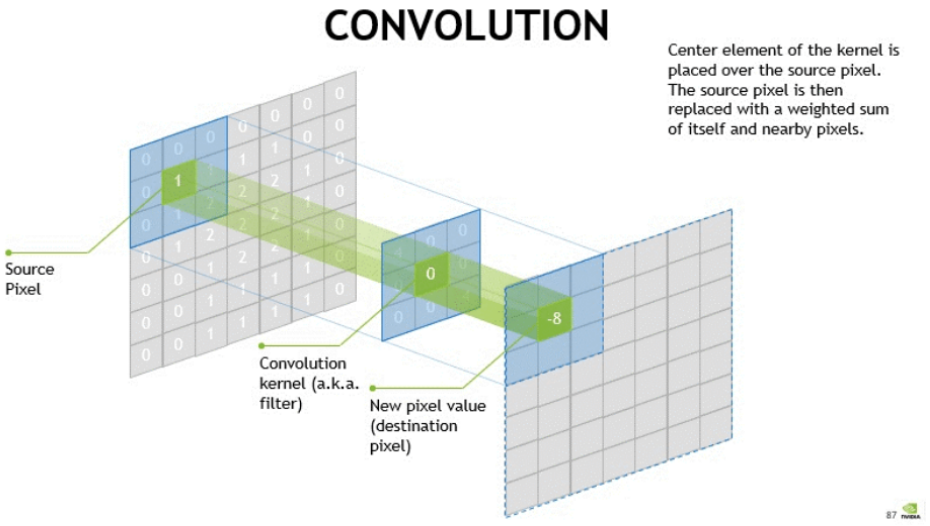



*Fig 2.8: shows Convolutional Layer [12]*

### 2.9.2 RELU Layer

The **negative** values from the filtered image are removed from this layer and replaced with zero values. *(fig 2.9)*

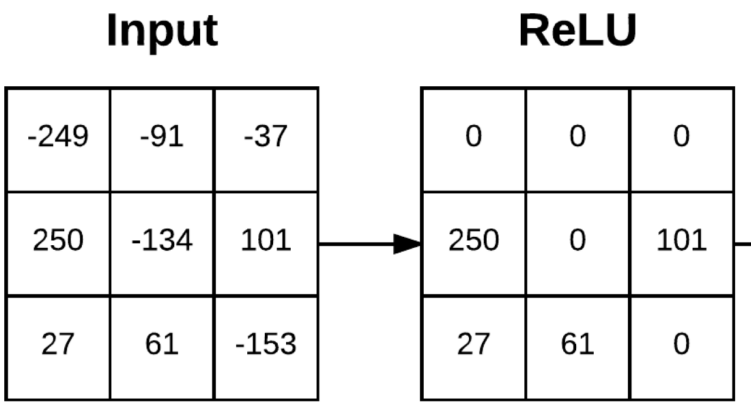



*Fig 2.9: shows ReLu Layer [13]*

### 2.9.3 Pooling Layer

As the input volume is progressively reduced in width and height, the POOL layer serves as a primary function. By doing this, we can reduce the number of parameters and computations in the network pooling process. This also reduces the chance of overfitting. [13]

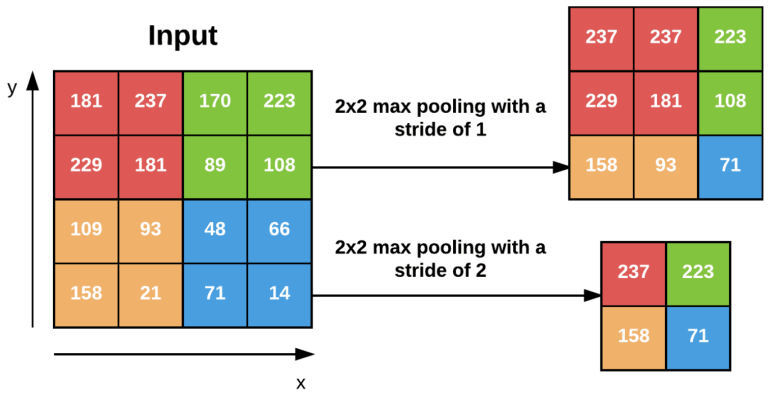



*Fig 2.10: shows Pooling Layer [13]*

### 2.9.4 Fully Connected Layers

In CNNs, fully connected layers are usually used at the end to make predictions about input, such as classifying it to a label, using features learned by convolutional and maximum pooling layers. [14]

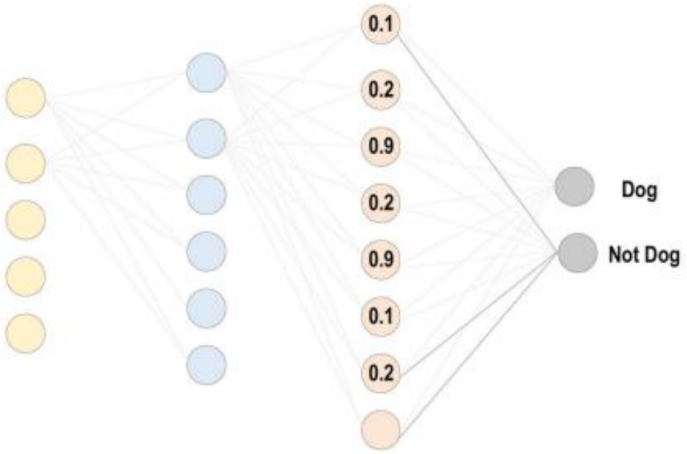



*Fig 2.10: shows fully connected Layer*

### 2.9.5 Output Layer

Convolutional Neural Networks (CNNs) have an output layer that is responsible for producing the final output of the network, similar to a classification or regression result. [14]

## 2.10 Convolutional Autoencoder

The autoencoder is a deep learning architecture that reconstructs data instances from the feature vectors associated with them. The project focuses mainly on their application to image data, but they work on all sorts of data. Autoencoders are composed of three main components, which include an ***encoder***, a ***bottleneck***, and a ***decoder.*** [15]

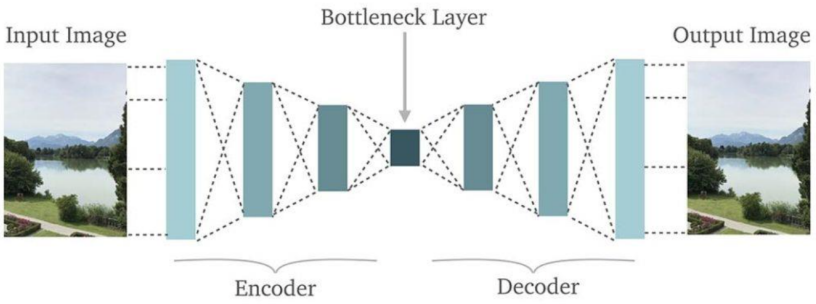



*Fig 2.11: shows convolutional encoder*

### 2.10.1 Encoder

Autoencoders include an encoder, which is a connecting network that performs the function of extracting features. An image is analyzed using this tool in order to extract the most important features and return them as vectors.

### 2.10.2 Bottleneck

In addition to encoding features, the bottleneck acts as an additional layer that helps compress them into a smaller vector representation. To force the decoder to learn more complex mappings, this technique is designed to make it harder for the decoder to understand the features.

### 2.10.2 Decoder

An autoencoder's decoder is the last component that attempts to reconstruct the original image after the encoder has compressed the features, so that the original encoded image can be reconstructed.

## 2.11 Conclusion to Background

A brief summary of some important concepts has been provided in this chapter, starting with the digital image, processing and deep learning, moving on to the convolutional neural network and its architecture, as well as ending with the auto-encoder. It was important for us to provide some background information about the work, so that the work and its context would be clearer to understand.

## 2.12 Related Papers

Throughout the course of this project, we have meticulously positioned ourselves within the dynamic realm of artificial intelligence (AI) and its burgeoning applications in healthcare, in particular medical imaging to diagnose diseases. In order to construct the nuanced trajectory of the study, an exhaustive exploration of pertinent literature guides the formulation of the research question, its contextual framing, and the methodology.

1. **LeCun, Bengio, & Hinton, G. (2015). Deep learning. Nature.** In this seminal study, deep learning is presented as a transformative technology that can be applied in various sectors, including healthcare. Researchers argue that convolutional neural networks (CNNs), which are particularly suitable for image recognition tasks, are particularly suitable for deep learning algorithms. This paper provides insights that guide the choice of CNNs for use in the AI model proposed. https://www.nature.com/articles/nature14539

2. **Kaiming He, Xiangyu Zhang, Ren, Jian Sun (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.** It is the purpose of this paper to introduce the concept of Residual Networks (ResNet) and to demonstrate their effectiveness in solving the vanishing gradient problem in deep neural networks. ResNet's unique architecture, featuring skip connections or

identity mappings, has been adopted in the design of the proposed model for this project. https://ieeexplore.ieee.org/document/7780459

3. **Oalf Ronneberger, Philip Fischer, & Thomas Brox. (2015). U-Net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention.** The purpose of this paper is to introduce the U-Net architecture, a modification of the conventional CNN, designed specifically for the segmentation of biomedical images. As a result of the concepts and insights from this paper, we have gained a better understanding and implementation of the ResUNet architecture which will be used in this project. https://link.springer.com/chapter/10.1007/978-3319-24574-4_28

4. **Foivos Diakogiannis, Francois Waldner, & Peter Caccetta. (2020). ResUNet-a: a deep learning framework for semantic segmentation of remotely sensed data.** As a result of this paper, we propose a novel ResUNet architecture, which combines ResNet and U-Net strengths. This study demonstrates the superior performance of ResUNet in semantic segmentation tasks, thus justifying its use as part of the present research. https://www.researchgate.net/publication/339443963_ResUNet-a_A_deep_learning_framework_for_semantic_segmentation_of_remotely_sensed_data

5. **Kenneth Clark, Bruce Vendt, John Freymann, Justin Kirby, Paul Koppel, Stephe Moore, Stanley Phillips, David Maffitt, Michael Pringle. (2013). The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository. Journal of Digital Imaging.** The purpose of this document is to introduce and explain how the Cancer Imaging Archive was created and how it has been maintained, the primary source of the brain MRI scans used in this project. It provides an essential context for understanding the dataset and its potential applications in diagnosing diseases using artificial intelligence. https://pubmed.ncbi.nlm.nih.gov/23884657/

## 2.13 Conclusion to Related Papers

Bringing together these selected references, a symphony unravels AI's power in segmenting images and detecting diseases. A comprehensive comparative analysis is warranted even though certain papers align seamlessly with the research goals. To create a robust critical context, it is imperative to evaluate each approach critically, juxtapose their performance characteristics, and assess their adaptability to the project's circumstances. A comparative analysis leads to a better understanding of the selected methods and lays the groundwork for a smart artificial intelligence model capable of facilitating transformational change in the detection of brain tumors.

# CHAPTER 3

# Methodology

## 3.1 Introduction

This chapter explains the techniques used in the project and lays out the methodology employed. Two architectural styles are represented in our work. To extract features from the dataset for the first architecture, we utilized pre-trained models like ResNet-50 and VGG-16. Following the completion of the current phase, we will get brain tumor images which will be localized by ResUnet. To detect and analyze brain tumors, the project uses medical images that precisely identify and define regions of interest. A significant picture classification task had already been trained on these models, which made them good feature extractors. In order to improve classification performance, these two models were combined to combine their distinct strengths.

## 3.2 Convolutional Neural Network (CNN)

An image classification model using convolutional neural networks (CNNs) is presented in this section. Multiple layers are involved in the classification process, each serving a different purpose. Our model is also trained over a set number of epochs so that it can learn and optimize its performance over the course of the training process.

## 3.2 Activation Function

How well the network model learns from the training dataset will be determined by the activation function in the hidden layer. Choosing the activation function for the output layer will determine how the model makes predictions. Here are the activation function used in this project:

### 3.2.1 Rectified Linear unit (ReLU)

Essentially, a rectified linear activation function or ReLU outputs the input directly if it is positive, otherwise it outputs zero. [16]

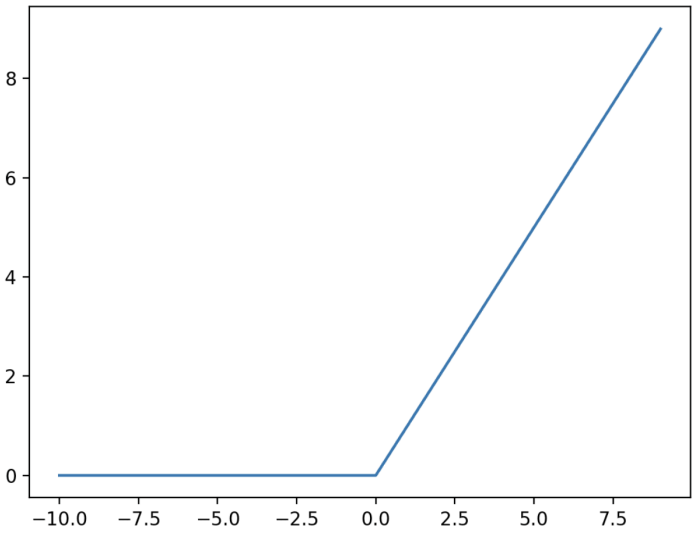



*Fig 3.1: shows ReLu activation function [16]*

### 3.2.2 Softmax Function

The softmax function makes it possible to convert a vector of numbers into a vector of probabilities, whereby the probabilities of the values in the vector are proportional to the relative scales of the values within the vector, which can be either positive or negative. This function is one of the most commonly used in applied machine learning because it is commonly used to activate a model of neural networks in the context of applied machine learning. [16]

## 3.3 Loss Function

Machine learning models use loss functions to quantify differences between predicted and actual values. By providing feedback on how well the model fits the data, it measures the model's performance and guides the optimization process. Here are the Loss function used in this project:

### 3.3.1 Categorical cross-entropy

In Multiclass classification and softmax regression, categorical cross entropy is used. A comparison is made between the predicted probabilities for different classes and the actual class labels. Categorical Cross-Entropy measures the difference between predictions and the actual distributions in order to quantify the error between predictions and the actual distributions. [17]

### 3.3.2 Binary cross-entropy

The method is used to solve binary classification problems, such as those involving two classes. An example would be whether a person has a covid or not, or whether my article is popular or not. In binary cross entropy, each predicted probability is compared to the actual

class output, which is either 0 or 1. Following that, the probability scores are penalized based on how far they are from the expected value. [17]

## 3.4 VGG Architecture

At Oxford University, Karen Simonyan, Andrew Zisserman, and others developed **VGG**, a CNN architecture. It is an acronym for "**Visual Geometry Group** 16". This neural network architecture was developed by the Visual Geometry Group at the University of Oxford. The number 16 indicates that the model has **16** layers with weights, including convolutional and fully connected layers. This 16-layer CNN is trained on over a billion images (1000 classes) and has 95 million parameters. With 4096 convolutional features, it can process images with 224 x 224 pixels. [14]

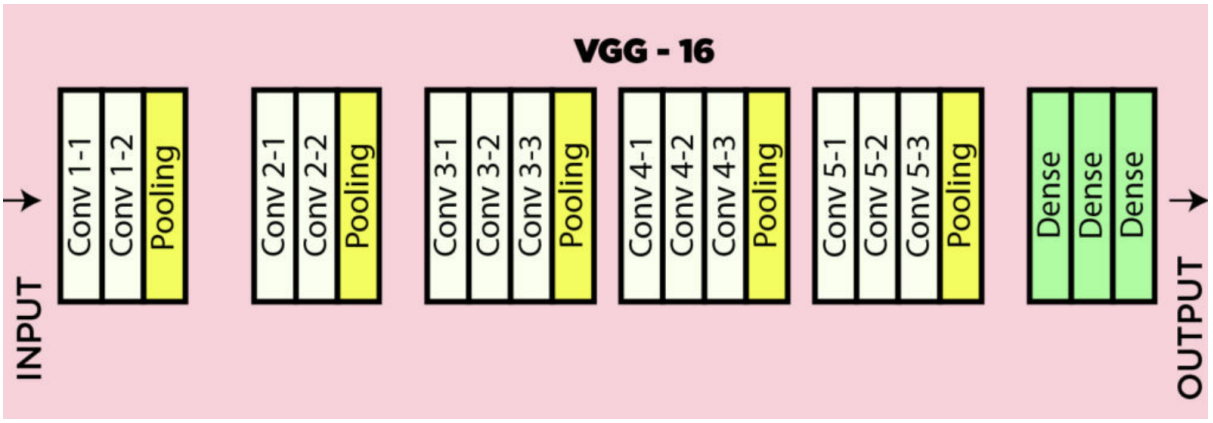



*Fig 3.2: shows the VGG-16 Architecture [18]*

## 3.5 ResNet Architecture

Deep learning models such as ResNet are often used in computer vision applications. There are hundreds or thousands of convolutional layers in this Convolutional Neural Network (CNN) architecture. The name ResNet stands for "**Residual Network**".

### 3.5.1 ResNet-50 Architecture

In ResNet-50, there are 50 layers of convolutional neural networks. Pretrained neural networks can be loaded from the ImageNet database based on more than a million images. An image can be classified into 1000 object categories by the pretrained neural network, including keyboard, mouse, pencil, and many animals. This process has resulted in the neural network learning rich feature representations for a wide range of images. [19]

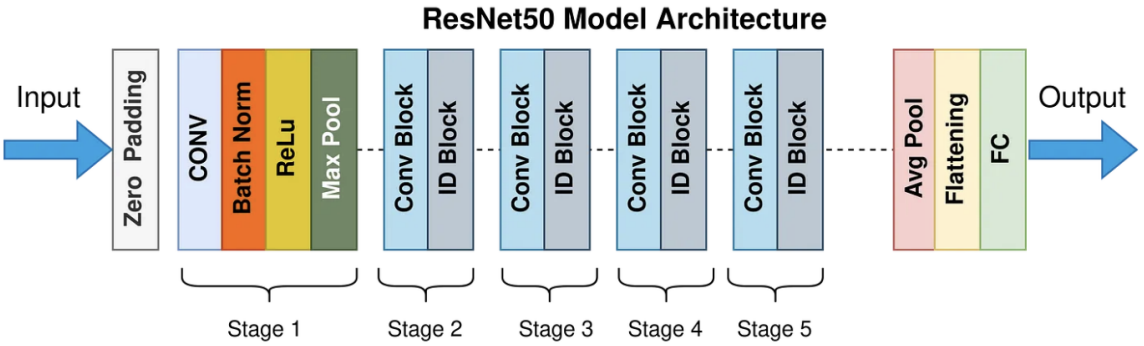



*Fig 3.3: shows the ResNet-50 Architecture*

### 3.5.2 ResUnet Architecture

**RESUNET** stands for Deep Residual UNET. A semantic segmentation system based on an encoder-decoder architecture was developed by Zhengxin Zhang. In remote sensing image analysis, it was originally used to extract roads from high-resolution aerial images. Researchers later applied it to polyp segmentation, brain tumor segmentation, and human image segmentation as well. With RESUNET, you'll get high performance with fewer parameters and a fully convolutional neural network. A significant improvement over UNET's existing architecture. With RESUNET, you can take advantage of both the UNET architecture and the Deep Residual Learning algorithm. [20]
In addition to creating deeper networks, residual blocks help to avoid the problem of vanishing gradients or explosions of gradients. Also, it simplifies the process of training the network. I will advance deeper in the 'discussion' chapter, on how I utilize the RESUNET to localize the brain tumor.

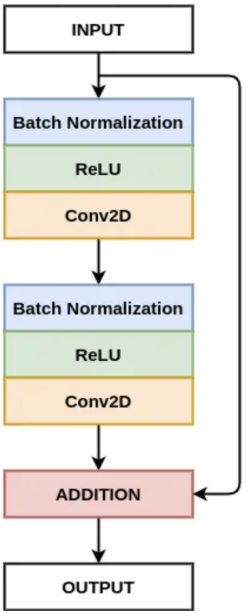



*Fig 3.3: shows the ResNet-50 Architecture [20]*

## 3.6 Visualization Of Model Architectures

This section provides a visual overview of the core codebase components of the project, highlighting crucial elements such as activation functions, loss functions, and other relevant configurations. Providing a clear understanding of the architecture will be the aim of this presentation.

### 3.6.1 VGG-16 Model

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

```
# Adding classification head
x = AveragePooling2D(pool_size=(4, 4))(basemodel.output)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)  # Adjust the dropout rate as needed
x = Dense(2, activation='sigmoid')(x)  # have 2 classes for binary classification

# Create the VGG-16-based model
model = Model(inputs=basemodel.input, outputs=x)

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.001),  # Adjust the learning rate
    loss='categorical_crossentropy',  # Use categorical cross-entropy for binary classification
    metrics=['accuracy']
)
```

*Fig 3.4: shows the VGG-16 Architecture code*

1. **AveragePooling2D:** Trims the input feature maps' spatial dimensions.
2. **Feature maps are flattened into one-dimensional vectors by flattening them.**
3. **Dense(256, activation='relu'):** Replaces the 256 neurons with 256 neurons and an activation function for ReLU.
4. **Dropout(0.5):** Reduces overfitting during training by introducing dropout.
5. **Dense(2, activation='sigmoid'):** Suitable for binary classification problems because it contains 2 neurons (binary classification) and a sigmoid activation function.
6. With a specified learning rate, categorical cross entropy for the loss function (suitable for binary classification), and accuracy for the evaluation metric, the model is compiled using the Adam optimizer.

### 3.6.2 ResNet-50 Model

```
# freeze the model weights

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

```
# Add classification head to the base model

headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)#
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = 'softmax')(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)
```

```
# compile the model

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

*Fig 3.5: shows the ResNet-50 Architecture code*

As shown in the code above, we have added a classification head to our pre-trained base model. Here is a breakdown of each step:

1. **headmodel = AveragePooling2D(pool_size = (4,4))(headmodel):** An Average Pooling layer is applied to the output tensor with a pool size of (4,4). Input tensors can be reduced in spatial dimensions by average pooling while essential features are maintained.
2. **headmodel = Flatten(name= 'flatten')(headmodel):** A Flatten layer is used to flatten the output of the pooling layer. As a result of this operation, the 3D tensor is transformed into a 1D tensor, allowing it to be fully connected.
3. **headmodel = Dense(256, activation="relu")(headmodel):** We introduce 256 neurons in the Dense layer, which is activated using Rectified Linear Units (ReLU). In this layer, the model learns complex patterns due to its non-linearity.
4. **headmodel = Dropout(0.3)(headmodel):** This method drops out the previous layer at a rate of 0.3, which helps prevent overfitting by randomly setting a fraction of input units to zero during training.
5. **headmodel = Dense(256, activation="relu")(headmodel):** 256 neurons are added to the model with ReLU activation, further enhancing its ability to convey complex patterns.
6. **headmodel = Dropout(0.3)(headmodel):** Overfitting is mitigated by the addition of another Dropout layer.
7. **headmodel = Dense(2, activation='softmax')(headmodel):** This model is based on a Dense layer with 2 neurons and a softmax activation function. To calculate probability scores for each class in a binary classification task, softmax activation is commonly used.

8. **model = Model(inputs=basemodel.input, outputs=headmodel):** Using the Model class in Keras, one can specify inputs (from the base model) and outputs (from the classification head).
9. **loss='categorical_crossentropy':** Used during training, this parameter specifies the loss function. A multi-class classification problem is commonly solved using 'categorical_crossentropy'.
10. **optimizer='adam':** During training, the optimizer determines the update rule for the weights of the network. In terms of efficiency and adaptability, Adam is one of the most popular optimization algorithms.
11. **metrics=["accuracy"]:** Accuracy serves as a metric for evaluating the model's performance during training. As a measure of the proportion of samples correctly classified, accuracy is a commonly used metric for classification tasks.

## 3.7 Brain Tumor Model Architecture

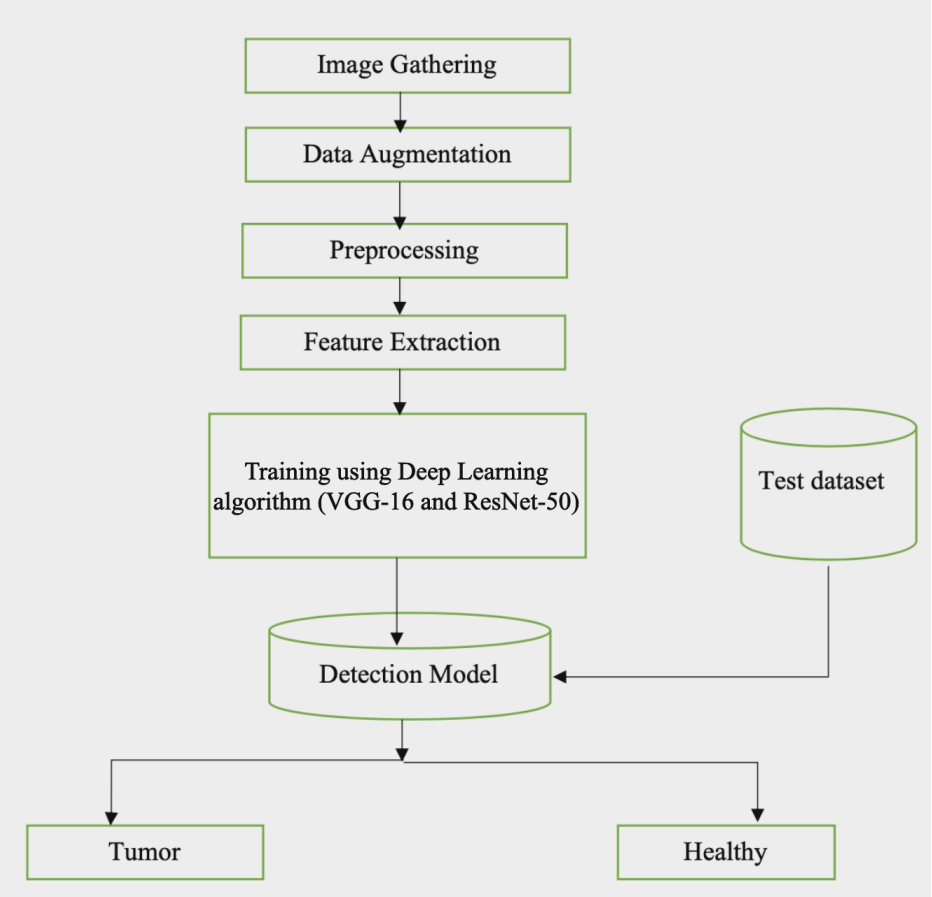



*Fig 3.6: shows the Architecture of the brain tumor model*

## 3.8 Conclusion

The following section presents a comprehensive methodology for classifying brain tumors with the power of convolutional neural networks, VGG-16 and ResNet-50. We employed these architectures because they are proven to be effective at image classification tasks, especially in the medical imaging field. A comparative analysis of VGG-16 and ResNet-50 was enabled by their distinct architectures.

Utilizing a curated dataset of brain MRI images, our methodology included a thorough data collection process. Our models are more robust after performing essential data preprocessing steps, such as converting pixel values to a consistent scale and categorizing target labels. A data augmentation technique was also employed to mitigate the risk of overfitting by diversifying the training dataset.

In addition to the classification head, architectural modifications were made to suit the characteristics of our task. To capture intricate features within images, average pooling, flattening, and dense layers with appropriate activation functions are used. To enhance our models' generalization capability, we strategically introduced dropout layers to prevent overfitting.

Critical decisions were made in constructing our models, including the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric. We make these decisions in accordance with best practices in binary classification tasks and contribute to the overall effectiveness of our approach.

The methodology we developed lays a solid foundation for subsequent phases of our research. We are now able to investigate the variability in brain tumor classification performance between VGG-16 and ResNet-50 as a result of the comprehensive steps we have taken, including data preprocessing, model compilation, and data analysis.

# CHAPTER 4

# Experimental Results &Evaluation

## 4.1 Overview

In this pivotal section, I delve into the center of my research project, where the culmination of my efforts is shown. Presentation and interpretation of results are integral components, serving as the cornerstone upon which my work's success and impact are measured. Over the following pages, I meticulously unravel the outcomes of the experiments. I offer an insightful analysis of the performance of the VGG-16 (Visual Geometry Group) and ResNet-50 (Residual Networks) models in the challenging domain of brain tumor classification.

The extensiveness with which our research was conducted is evident in this section of our report. I'm optimistic that the reader will join me on this path of discovery, understanding, and reflection as I navigate through the maze of numbers, charts, and statistical significance. In addition to encapsulating the essence of my exploration, the narrative presented here lays the foundation for more in-depth discussions that transcend the boundaries of my immediate findings in order to advance our understanding of how to classify brain tumors for future research.

It is my aim in these pages not only to provide insights into what has been found, but also to foster a dialogue - an intellectual exchange that asks questions and provides an incentive for further study. In this chapter, I encourage the reader to see how our research process unfolds as we present our results and findings.

## 4.2 Dataset

In my project on "brain tumors," precise data was necessary. Thus, we selected a dataset from TCIA (Cancer Imaging Archive), available on Kaggle also, that consists of radiographic images of MRI type captured from various angles for tumor identification.

- The dataset contains '*Image Path*', '*Mask Path*', and a '*mask*'. There are two values in the mask: **0** and **1**. A **0** indicates that there is no tumor and a **1** indicates that there is a tumor. The path contains an actual MRI image in the dataset.

- In total, **3929** images have been included in the dataset which are appropriately grouped into

| Mask | Number Of Images |
|---|---|
| 0 | 2556 |
| 1 | 1373 |
| ***Total*** | ***3929*** |

*Table 4.1: description of the brain tumor dataset.*

## 4.3 Data Images

An overview of the images in the dataset can be found in this section:

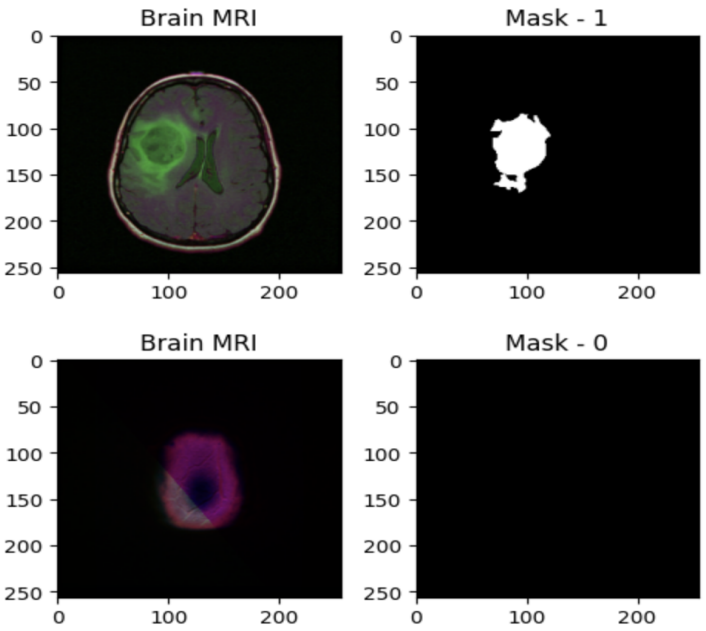



*Fig 4.1: shows the Brain MRI and Mask Image*

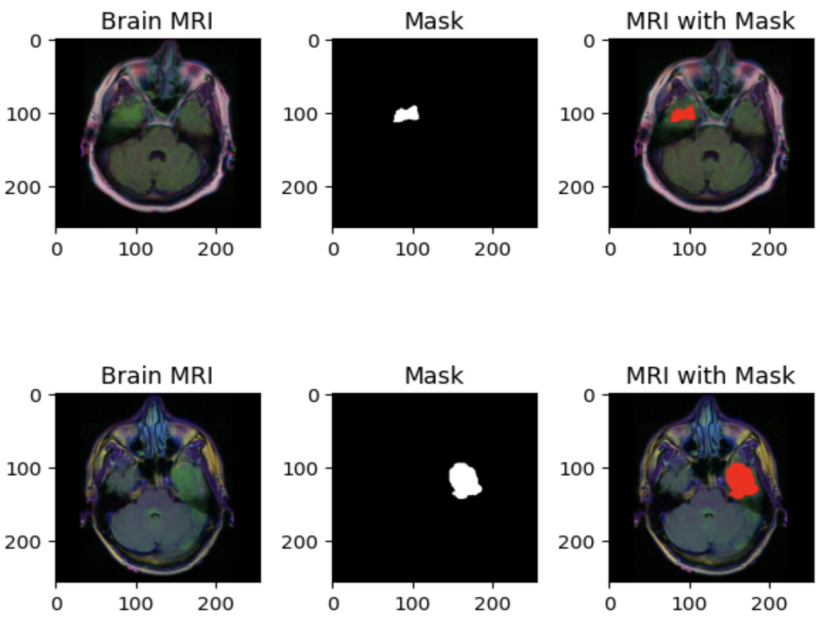



*Fig 4.2: shows the Brain MRI, Mask Image and MRI with mask*

## 4.4 Preprocessing steps

The following preprocessing steps were employed to standardize the data and facilitate training of the model:

- **Normalization:** To enhance model convergence, pixel intensity values were scaled between 0 and 1.
- **Resizing:** Images were resized to a uniform 256x256 pixel size to ensure consistent input dimensions for VGG-16 and ResNet-50.

## 4.5 Experimental Setup

This implementation makes use of the Google Colab and several Python packages such as Numpy, Pandas, OpenCV TensorFlow, Matplotlib, Keras, etc. Scikit-Learn is used for traditional classifiers. Throughout the project, we used Python 3.11. When training and testing the models through CNN, we used Tensorflow and Keras. Google Colab provided a dedicated GPU for us to use.

## 4.6 Performance Measures

In order to determine the accuracy of our model, we need to look at the performance matrices. During the evaluation of our model, we take into account the performance measures described here.

### 4.6.1 Confusion Matrix

In order to determine whether or not a model is accurate and correct, the confusion matrix is used as one of the most intuitive metrics. Classification problems with multiple types of output are solved with this method. In order to calculate the Confusion Matrix, we need to know the following four parameters:

- True Positive **(TP):** Number of correctly classified tumor images.
- True Negative **(TN):** Number of non-tumor images correctly classified.
- False Positive **(FP):** Number of non-tumor images misclassified as tumors.
- False Negative **(FN):** Number of tumor images misclassified as non-tumor.

### 4.6.2 Accuracy

The accuracy of a result or prediction is determined by how close it is to the true or expected value. Typically, the number of correct predictions or outcomes is compared to the total number of predictions made to evaluate the performance of a model or system. Basically, accuracy measures the degree of accuracy in an output produced by a process or algorithm. The accuracy value is often expressed as a percentage, with a higher value indicating greater precision and reliability.

$$Accuracy = \frac{Correct\ \ Predictions}{Total\ \ number\ \ of\ \ images} = \frac{TP + TN}{TP + TN + FP + FN}$$

*Fig 4.3: shows an accuracy equation*

### 4.6.3 Precision

Information retrieved from the model is relevant information. Precision is calculated as the ratio between tumor images correctly classified (TP) and tumor images classified or misclassified (TP + FP). When the FP is lower, the Precision is higher. When the precision rate is higher, the model is more effective.

$$Precision = \frac{TP}{TP + FP}$$

*Fig 4.4: shows an Precision equation*

### 4.6.4 Recall

This is the percentage of images that are successfully retrieved. In order to measure recall, we must compare the number of tumor images that are correctly classified with the number of images that need to be predicted. In addition to Sensitivity, Hit Rate, and True Positive Rate, Recall is also known as True Positive Rate. Generally, a lower False Negative means a higher recall since less tumor images are classified as non-tumors.

$$Recall = \frac{TP}{TP + FN}$$

*Fig 4.5: shows an Recall equation*

### 4.6.5 F-Score

In machine learning and statistics, the F1 score incorporates precision and recall measures into one value. This method provides a balanced measure of a model's accuracy since it takes into account both the accuracy of the model's ability to identify positive instances and its ability to capture all positive instances that are relevant (recall). F-score reaches its best value at 1 (100% precision and recall) and worst value at 0.

$$F - Score = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

*Fig 4.6: shows an F-score equation*

### 4.6.6 Classification Report

A classification-based machine learning model uses this metric to evaluate its performance. The F1 score, precision, recall, and support of your model are displayed. As a result, we are better able to understand the overall performance of the trained model. Additionally, there is a **Macro-average** that shows average performance across classes, considering each to be equally important.
Also a **Micro-average** that gives equal weight to every instance and displays average performance across all predictions. For multi-class classification, precision, recall, and accuracy are all micro-averaged.

## 4.7 Hyper-parameters Setting for ResNet and VGG Models

We discuss here the hyperparameter values used in the initialization and training stages for implementing our ResNet, VGG model. Below you will find a chart illustrating the values used for the 1 and 2 experiment:

| Hyper parameter | ResNet (Residual Network) | VGG (Visual Geometry Group) |
|---|---|---|
| **Learning Rate** | 0.001, 0.0001 | 0.001, 0.0001 |

| | | |
|---|---|---|
| **Optimizer** | Adam | Adam |
| **Batch Size** | 16, 32 | 16, 32 |
| **Epochs** | 10, 30 | 10, 30 |
| **Activation Function** | ReLu, Softmax | ReLu, Sigmoid |
| **Loss** | Categorical Cross-entropy | Categorical Cross-entropy |

*Table 4.2: shows description of hyper-parameter setting.*

## 4.8 Initial Experimental Results

I started the experimentation by establishing a baseline to train two prominent convolutional neural network (CNN) architectures - VGG-16 and ResNet-50 - on Brain MRI images to classify tumors. These models were tested under standard configurations and parameters in order to gauge their initial performance.

Key Experimental Details:

1. **Models Evaluated:** The VGG-16 and ResNet-50 architectures were chosen because they are widely used and have demonstrated success for a variety of computer vision tasks.
2. **Number of Epochs:** The training duration was limited to ***10*** epochs to provide a conservative approach. Thus, the initial learning pattern and convergence behavior of the models could be quickly assessed.
3. **Dataset Split:** The dataset was split into an ***80***% training set and a ***20***% test set. To ensure the training data was sufficient for the models, but a substantial portion would be used for rigorous analysis, this division aimed to strike a balance.
4. **Learning Rate:** To facilitate a stable learning process during optimization, a learning rate of 0.001 was chosen.
5. **Batch Size:** The batch size of ***16*** optimized computational efficiency while enabling mini-batch stochastic gradient descent.

Experimental Objective

This experiment was conducted in order to obtain an understanding of how VGG-16 and ResNet-50 performed when classified Brain MRI tumors. We observed early trends in model convergence by limiting training to 10 epochs, assessed the models' generalization abilities to unknown data, and identified potential areas for improvement. Our subsequent experiments refined and optimized the model performance by introducing variations in hyperparameters and training duration based on insight gained from this foundational experiment.

### 4.8.1 Overview Of Training and Validation metrics

A review of the training and validation metrics provides insights into the learning dynamics that have taken place in the models over the course of ten epochs. Our approach is to observe how loss and accuracy metrics evolve over time, and examine the extent to which each model generalizes to unseen data in the future.

**VGG-16 Architecture**

The VGG-16, a deep and simple model, was trained over ten epochs as part of its training procedure. There is great insight to be gained from examining the progression of training and validation loss along with accuracy as it relates to the learning process.

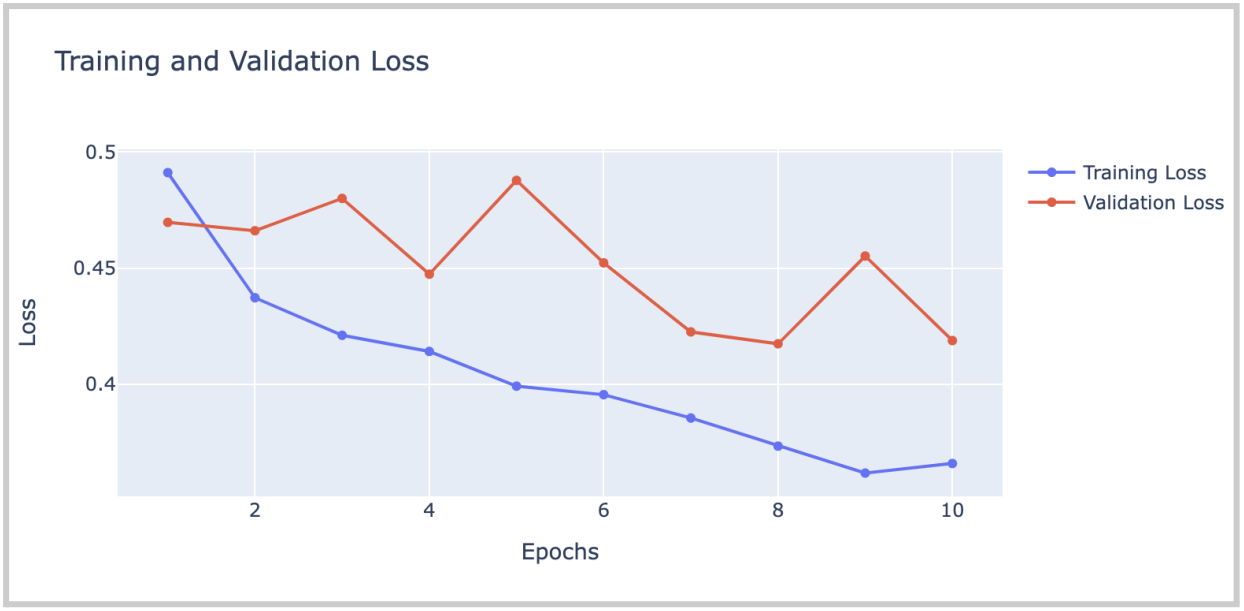



*Fig 4.7: shows the Training and Validation Loss for VGG-16*

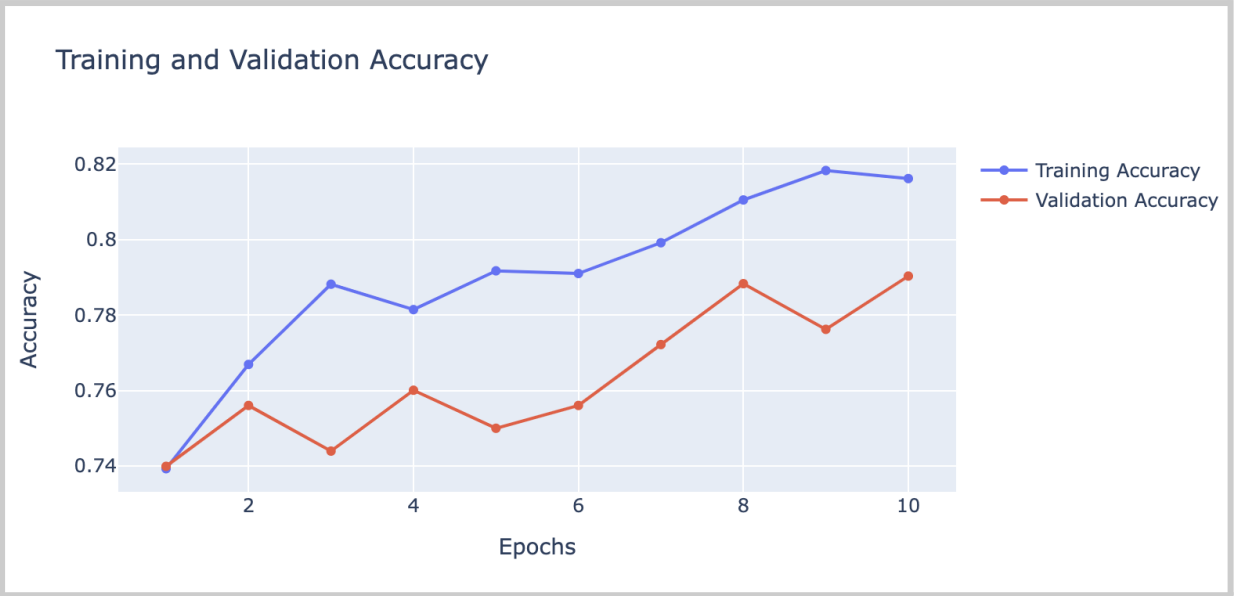



*Fig 4.8: shows the Training and Validation Accuracy for VGG-16*

This is an overview of the epochs 1 through 10:

- **Training Loss:** The training loss started at 0.4912, but gradually reduced to 0.3658 at the end of the training process.
- **Validation Loss:** Results exhibit a wide range of variation, ranging from 0.4174 to 0.4878.
- **Training Accuracy:** The accuracy began at 73.93%, but by the tenth epoch had reached 81.62%.
- **Validation Accuracy:** The accuracy of the validation varied, peaking at 79.03% at one point in time.

As the training loss consistently decreases, the model's ability to generalize is evident, which indicates that the model is learning effectively. The validation metrics demonstrate how well the model performs on unseen data, highlighting both its strengths and potential areas for improvement based on the results of the validation.

In conclusion, the VGG-16 model training over 10 epochs has demonstrated significant progress in learning and adapting to the provided Brain MRI dataset. During training, the training loss decreased consistently from 0.4912 to 0.3658, indicating that the model effectively learns and optimizes its parameters.

There was a wide range of variations in validation loss and accuracy from 0.4174 to 0.4878, peaking at 79.03%, suggesting that the model's performance on unseen data had varied throughout the training period. Insights into potential areas for model improvement could be gained by understanding and addressing these variations.

Although there have been some fluctuations in the training accuracy, the overall trajectory of the model's learning has been promising, indicated by its upward trend from 73.93% to 81.62%. On the validation set, the model achieved an accuracy peak of 79.03%, which shows that it is capable of generalizing from the training data to a degree.

## ResNet-50 Architecture

In addition to the ResNet-50, which uses a residual architecture, ten epochs of training was also conducted on it. Here is a summary of the key metrics that are included in the report.

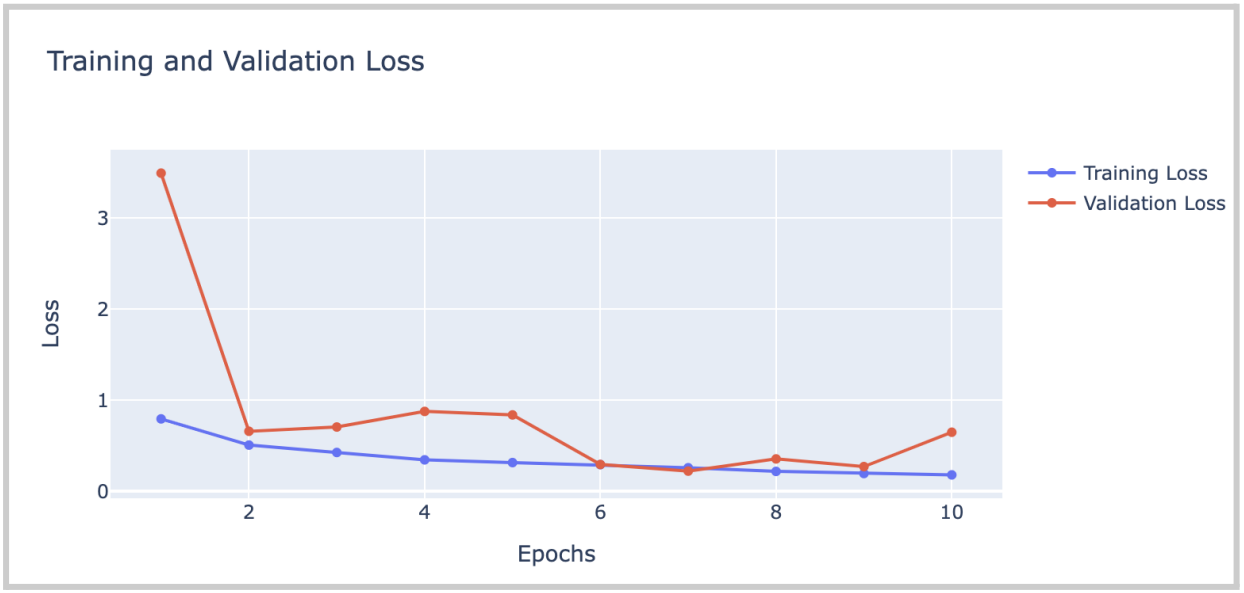



*Fig 4.9: shows the Training and Validation Loss for ResNet-50*

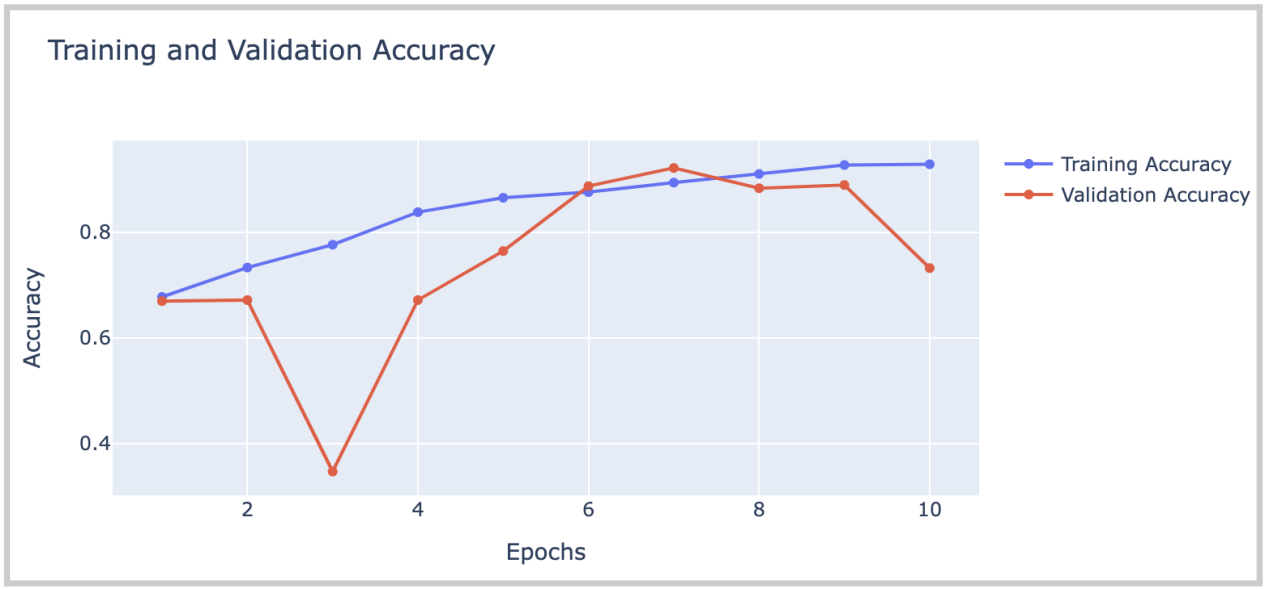



*Fig 4.10: shows the Training and Validation Accuracy for ResNet-50*

This is an overview of the epochs 1 through 10**:**

- **Training Loss:** After initiation, the loss was 0.7961, which was significantly reduced to 0.1817 at the end.
- **Validation Loss:** The model showed a number of variations, but it also displayed a significant improvement from 0.66029 to 0.2231.
- **Training Accuracy:** The accuracy rate for the training began at 67.73%, and reached an impressive 92.84% by the end.
- **Validation Accuracy:** Variable, with an improved accuracy score of 88.31% being noted.

This model shows a remarkable ability to learn intricate features based on the steady decline in training loss and substantial improvement in accuracy it exhibits. It is noteworthy that the validation loss reduced from 0.66029 to 0.22319, which indicates an improvement in generalization.

In summary, the results obtained from training the ResNet-50 model over the course of ten epochs have resulted in a highly effective neural network that can be used to classify brain tumor MRI images. This is owing to its ability to learn from MRI data. A robust learning capability and its capacity to generalize to unknown data are evident in the model's training and validation metrics.

From a value of 0.7961 at the beginning to a remarkable 0.1817 at the end of training, the model's training loss significantly decreased. This indicated effective optimization of model parameters during training. The accuracy of the model improved substantially, starting at 67.73 percent and reaching an impressive **92.84**%.

It is evident from the validation metrics that the model is capable of generalizing. There was a substantial decrease in the validation loss from 0.66029 to 0.22319, signaling improved performance on unseen data despite some fluctuations. The validation accuracy also showed commendable progress, peaking at **88.31**%.

In addition to its strong performance across various metrics, this model accurately classifies brain tumor MRI images. In the future, it might be possible to refine the hyperparameters of the model or explore additional techniques for data augmentation. The ResNet-50 model shows promising results, setting the stage for continued optimization and application to real-world Brain Tumor MRI classification scenarios.

In the following sections, we will examine the metrics relating to model accuracy, classification reports, and confusion matrices to provide a comprehensive understanding of the models' performance.

### 4.8.2 Model Accuracy

**VGG-16 Architecture**

Analyzing the test dataset, VGG-16's model accuracy has been calculated to be impressively **81.77**% based on evaluation of the test dataset. Metrics like these indicate how many instances were classified correctly compared to how many samples were analyzed.

With **81.77**% accuracy, the model is capable of detecting tumors in brain images and non-tumors in those images. It is therefore possible to study the model's strengths and potential areas for improvement using it.

**ResNet-50 Architecture**

It is noteworthy that ResNet-50 displays a remarkable model accuracy of **97.40**% when compared with other models. By doing so, it is evident that the model has a high degree of precision in correctly predicting the image classes of brain tumors.

With an accuracy rate of **97.40**%, ResNet-50 demonstrates a superior capability for accurately classifying a wide range of images of brain tumors in a very fast and efficient manner. The high level of accuracy is a strong indication of the reliability of ResNet-50 for classification tasks of this type.

The accuracy of the model is an important measure when it comes to evaluating the efficiency of the models in making accurate predictions, which is why we use it as a basis for the analysis that follows.

### 4.8.3 Classification Report Analysis

Below is the classification Report for **VGG-16**:

| | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 0.88 | 0.84 | 0.86 | 379 |
| 1 | 0.72 | 0.77 | 0.74 | 197 |
| **micro avg** | 0.82 | 0.82 | 0.82 | 576 |
| **macro avg** | 0.80 | 0.81 | 0.80 | 576 |
| **weighted avg** | 0.82 | 0.82 | 0.82 | 576 |

*Table 4.3: shows description of classification report for vgg-16.*

It is important to note that the classification report for VGG-16 provides a detailed performance breakdown of the model, which includes precision, recall, and the F1-score for both classes (0 and 1).

**Class 0** (No Tumor)**:**

- *Precision:* 88%
- *Recall:* 84%
- *F1 Score:* 86%

**Class 1** (Tumor)**:**

- *Precision:* 72%
- *Recall:* 77%
- *F1 Score:* 74%

The F1-score for the 'No Tumor' class is commendable at 86%, which demonstrates that the class performs well in both classes, with slightly more emphasis placed on correctly classifying instances without tumors.

This accuracy-recall balance is maintained, demonstrating that VGG-16 performs consistently across both classes, even when there is a change in class. In a nutshell, the F1-score provides an indication of both the false positives as well as the false negatives resulting from the model's application.

Below is the classification Report for **ResNet-50**:

| | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 | 386 |
| 1 | 0.97 | 0.95 | 0.96 | 190 |
| **micro avg** | 0.97 | 0.97 | 0.97 | 576 |
| **macro avg** | 0.97 | 0.97 | 0.97 | 576 |
| **weighted avg** | 0.97 | 0.97 | 0.97 | 576 |

*Table 4.4: shows description of classification report for resnet-50.*

ResNet-50's classification report contains a summary of the model's precision and recall in both classes (0 and 1), as well as the F1-score for both classes (0 and 1).

**Class 0** (No Tumor)**:**

- *Precision:* 97%
- *Recall:* 99%
- *F1 Score:* 98%

**Class 1** (Tumor)**:**

- *Precision:* 97%
- *Recall:* 95%

- *F1 Score:* 96%

The model exhibits exceptional precision, recall, and F1-score for both classes, highlighting its robust performance when it is used to classify both tumor-related images and those without tumors. The overall outstanding F1-score is mainly a result of the very high precision and recall rate for the 'No Tumor' class.

This model demonstrates a remarkable ability to accurately identify tumors as well as non-tumors by displaying a remarkable precision and recall rate. There is no doubt that ResNet-50 is very effective in classifying tasks as this is demonstrated by its high F1-score.

As a result of the analysis of the classification report, valuable insights are gained into the strengths of each model and the potential areas for improvement. As a result of VGG-16's balanced performance and ResNet-50's outstanding precision-recall harmony, these models can be utilized to develop brain tumor image classification algorithms. However, the Resnet architecture would be more appealing to users due to its more robust performance.

### 4.8.4 Confusion Matrix Insight

**A. <u>VGG-16 Architecture</u>**

For VGG-16, the confusion matrix provides a detailed breakdown of predicted and actual classifications for each of the two classes (0 and 1).

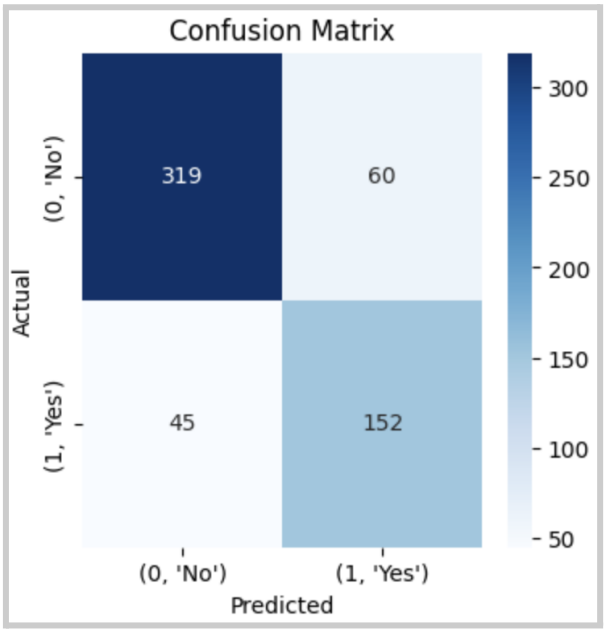



*Fig 4.11: shows the confusion matrix for VGG-16*

- *True Positives (1,1): 152*
- *True Negatives (0,0): 319*

- *False Positives (0,1): 60*
- *False Negatives (1,0): 45*

This study showed that the VGG-16 model exhibited an impressive ability to correctly identify images without tumors (No Tumor) with 319 true negatives (TN) for images without tumors (No Tumor) and 152 true positives (TP) for images with tumors (Tumor). Nonetheless, the percentage of false positives (FP) and false negatives (FN) indicate that the model misclassified images in 60 cases, while 45 of them were falsely classified.

The confusion matrix provides a visual representation of the performance of VGG-16 in the context of identifying tumors from non-tumor instances. This model exhibits a high degree of accuracy when it comes to detecting tumors while balancing accuracy in non-tumor cases at the same time.

**B. <u>ResNet-50 Architecture</u>**

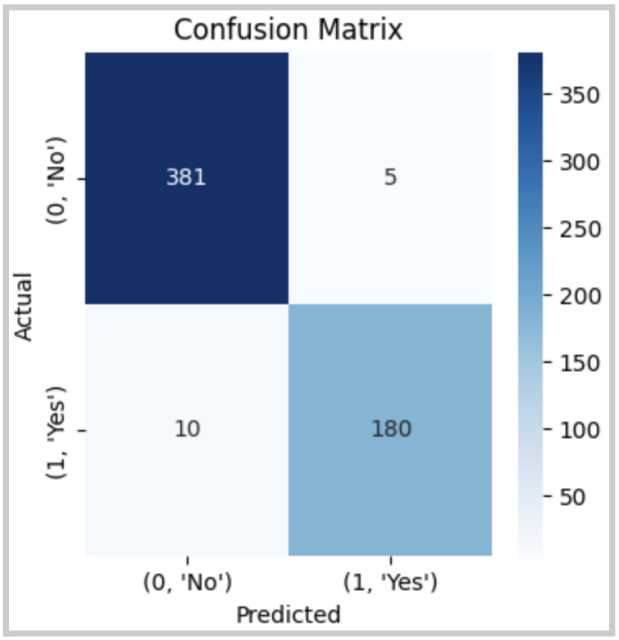



*Fig 4.12: shows the confusion matrix for ResNet-50.*

- *True Positives (1,1): 180*
- *True Negatives (0,0): 381*
- *False Positives (0,1): 5*
- *False Negatives (1,0): 10*

There is no doubt that ResNet-50 has an impressive performance showing 381 true negatives as well as 190 true positives while having a very low number of false positives and false negatives as well as having only 5 false positives (FP) and 10 false negatives (FN). In

addition to its impressive true negative rate, it also has a minimal false positive rate and false negative rate.

This confusion matrix shows that ResNet-50 has excellent accuracy in classifying tumors as well as non-tumor cases, as demonstrated in the confusion matrix. It is evident from the performance of the model that it offers robust performance with few cases of misclassifications occurring.

Comparing the two models, ResNet-50 has performed better as far as minimizing false positives and false negatives has been concerned compared to VGG-16. In the confusion matrix of the ResNet-50, the lower percentage of misclassifications indicates that there is a higher ability to discriminate between images containing tumors and images without them.

Therefore, in conclusion, based on results in the confusion matrix analysis, ResNet-50 shows a more robust performance, which for this task makes it a more effective model to use when classifying brain tumor images.

## 4.9 Second Experimental Results

Having gained insights from our initial experiment, we conducted a second phase of exploration to refine and optimize the performance of VGG-16 and ResNet-50 architectures for the classification of Brain MRI tumors. The goal of this experiment was to enhance the model capabilities by adjusting key hyperparameters such as epoch count, batch size, learning rate, and dataset split.

Key Experimental Adjustments

1. **Increased Epochs:** I extended training duration to 30 epochs to ensure that models can be learned and converged more comprehensively. Through this extended training period, intricate patterns within the dataset were captured, possibly leading to better classification results.
2. **Batch Size Enhancement:** The batch size was increased to 32, enabling more parallel training of models. Modifying the model in this way aimed at increasing computational efficiency and possibly enhancing generalization abilities.
3. **Optimized Learning Rate:** The learning rate was fine-tuned to 0.0001, a lower value than in the initial experiment. By reducing the step size during optimization, we could refine the convergence process and update the model weights with more precision.
4. **Adjusted Dataset Split:** A split of 85% was applied to the training dataset and 15% was applied to the testing dataset. Models could therefore learn robust features and patterns thanks to a larger pool of diverse training samples.

Experimental Objective

The objective of Experiment II was to systematically refine the critical hyperparameters of the VGG-16 and ResNet-50 models in order to improve their performance over time. It was anticipated that the extended training duration, larger batch size, and adjusted learning rate would contribute to a more nuanced understanding of the model as well as a higher classification accuracy. To determine if the models could generalize more effectively to unseen data, we rearranged the dataset split. Based on the results of this experiment, we were able to assess how hyperparameter adjustments affected the overall accuracy of the models for brain MRI tumor classification.

### 4.9.1 Overview Of Training and Validation metrics

During this final experiment, we examine the training and validation metrics of the ResNet-50 model over the course of 30 epochs to gain insight into the learning dynamics. Observing loss and accuracy metrics over time and examining future data, we wish to examine how well the model generalizes to unseen data in the future.

**A. VGG-16 Architecture**

Training was conducted over thirty epochs for VGG-16. From 1 to 30, here is a breakdown of the epochs:

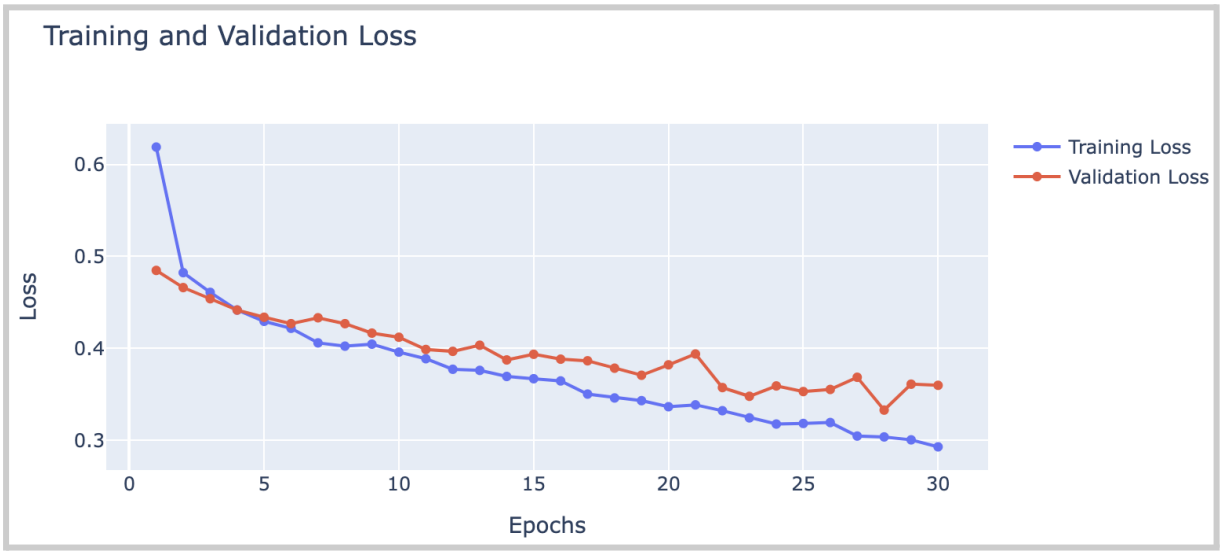



*Fig 4.13: shows the Training and Validation Loss for VGG-16*

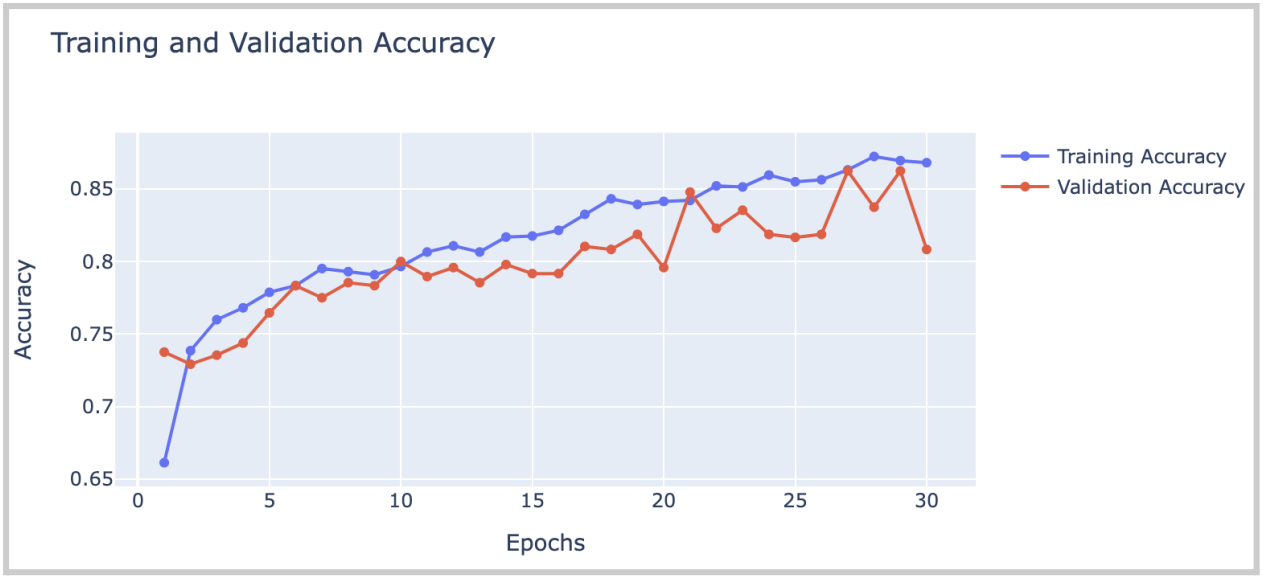



*Fig 4.14: shows the Training and Validation Loss for VGG-16*

- **Training Loss:** Started at 0.6189 and gradually reduced to 0.2929 at the end of the training process.
- **Validation Loss:** Exhibited a wide range of variation, ranging from 0.4847 to 0.3598.
- **Training Accuracy:** Began at 66.12% and reached 86.81% by the 30th epoch.
- **Validation Accuracy:** Varied but peaking at 86.25% at one time. At the 30th epochs it resolved to 80.83%

The metrics also demonstrate that the VGG-16 model learns and adapts effectively to the dataset provided. Losses during training consistently decrease, indicating that the model parameters have been optimized successfully. Although the validation loss fluctuates, it generally follows a decreasing trend. The model's accuracy metrics reveal respectable accuracy values in later epochs, demonstrating the model's generalization ability.

**B. ResNet-50 Architecture**

In addition to VGG-16, the ResNet-50 model, using a residual architecture, underwent 30 training epochs. Here are key metrics:

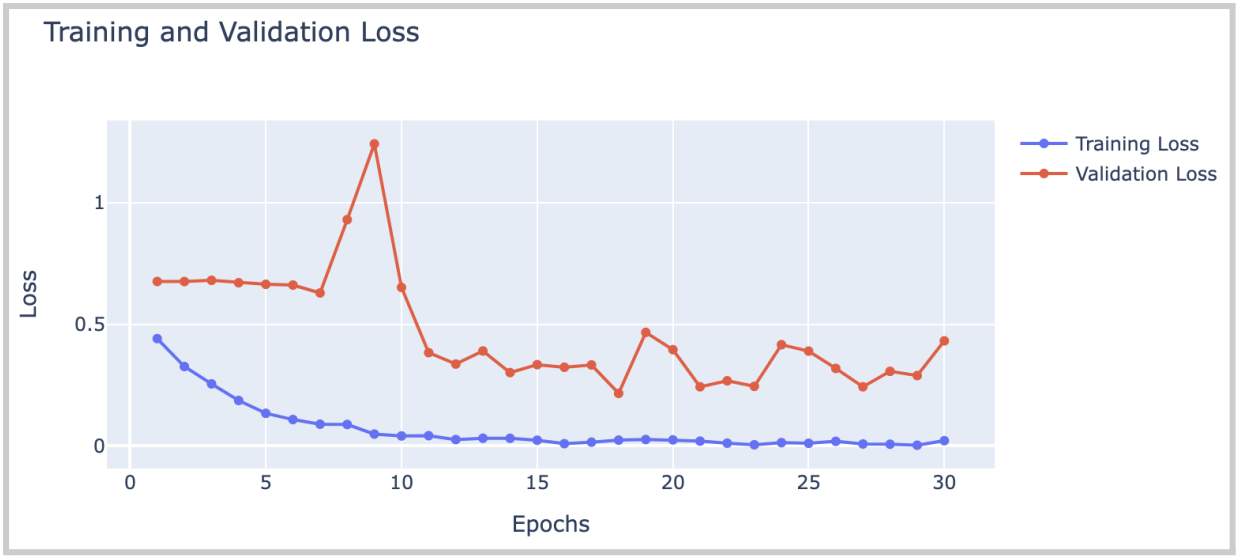



*Fig 4.15: shows the Training and Validation Loss for ResNet-50*

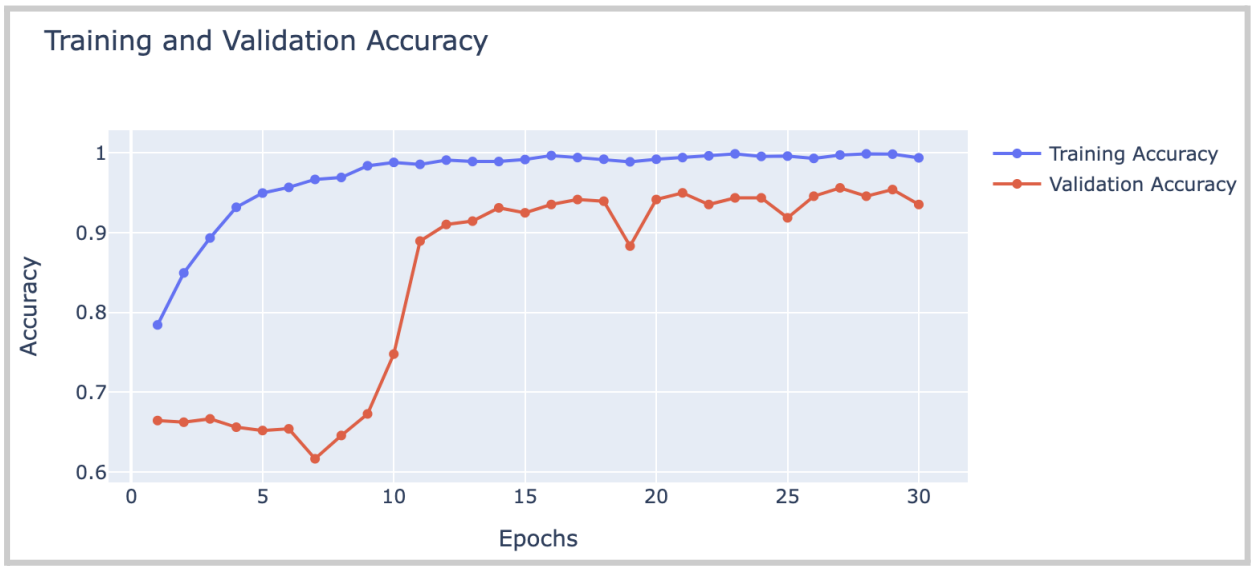



*Fig 4.16: shows the Training and Validation Loss for ResNet-50*

- **Training Loss:** Started at 0.4413, significantly reduced to 0.0215 at the end.
- **Validation Loss:** Displayed variations, improving from 0.6762 to 0.4321.
- **Training Accuracy:** Started at 78.44% and reached an impressive 99.39% at the end.
- **Validation Accuracy:** Variable, with an improved accuracy score of 93.54%.

It is evident from the training and validation metrics that the ResNet-50 model is capable of learning from the provided dataset and adapting accordingly. Consistently decreasing training losses indicate that the model has the capability to optimize its parameters. It is noteworthy that the validation loss has also improved, although there are occasional fluctuations. A high accuracy value can be observed during the later epochs, which indicates that the model is able to generalize, as reflected by the accuracy metrics.

**Conclusion to Training and Validation metrics**

Based on the significant reduction in training loss and notable improvements in accuracy, it can be concluded that ResNet-50 is a robust learning model, trained over 30 epochs. As a result of the validation metrics, the model was further demonstrated to be effective in generalizing unseen data, providing a solid foundation for its application in real-world environments.

### 4.9.2 Model Accuracy

**VGG-16 Architecture**

The VGG-16 model achieved a commendable accuracy of approximately **88.54**% on the test set. According to this accuracy score, the model was able to classify brain tumor MRI images correctly, which indicates its ability to learn from training data.

**ResNet-50 Architect**

The ResNet-50 model outperformed VGG-16 in terms of accuracy, achieving an impressive accuracy score of approximately **98.78**% on the test set. The excellent performance of ResNet-50 in classifying brain tumor MRI images can be attributed to its robust learning capability.

ResNet-50 has a more effective architecture for the given task, based on the substantial difference in accuracy between the two models. A more detailed understanding of the models' classification performance will be reached in the following sections, which will discuss precision, recall, and confusion matrices.

### 4.9.3 Classification Report Analysis

Below is the classification Report for **VGG-16**:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.95 | 0.86 | 0.90 | 359 |
| 1 | 0.80 | 0.93 | 0.86 | 217 |
| micro avg | 0.89 | 0.89 | 0.89 | 576 |
| macro avg | 0.88 | 0.89 | 0.88 | 576 |
| weighted avg | 0.89 | 0.89 | 0.89 | 576 |

*Fig 4.17: shows the classification Report for VGG-16*

Based on the classification report below, the VGG-16 model exhibits strong performance when classifying brain tumor MRI images:

**Class 0** (No Tumor):

- *Precision:* 0.95
- *Recall:* 0.86
- *F1 Score:* 0.90

**Class 1** (Tumor Present):

- *Precision:* 0.80
- *Recall:* 0.93
- *F1 Score:* 0.86

Based on its classification report, the VGG-16 model can be evaluated across various classes. As for recall, the model accurately identifies both classes with an impressive 93% recall for Class 1 and 86% for Class 0. The precision of positive predictions for Class 0 stands at 95%, while for Class 1, it stands at 80%. An overall measure of a model's performance is its harmonic mean of precision and recall, called the F1-score. VGG-16 displays an **F1**-score of **90**% for Class 0 and **86**% for Class 1. Based on these metrics, we can see that the VGG-16 model has a strong balance between precision and recall when it comes to correctly classifying brain tumor images.

Below is the classification Report for **ResNet-50**:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 | 361 |
| 1 | 0.99 | 0.98 | 0.98 | 215 |
| micro avg | 0.99 | 0.99 | 0.99 | 576 |
| macro avg | 0.99 | 0.99 | 0.99 | 576 |
| weighted avg | 0.99 | 0.99 | 0.99 | 576 |

*Fig 4.18: shows the classification Report for ResNet-50*

In the following classification report, the ResNet-50 model demonstrates exceptional classification performance for brain tumor MRI images:

**Class 0** (No Tumor)**:**

- *Precision:* 0.99
- *Recall:* 0.99
- *F1-Score*: 0.99

**Class 1** (Tumor Present)**:**

- *Precision:* 0.99
- *Recall:* 0.98
- *F1-Score:* 0.98

ResNet-50 performs exceptionally well at classifying brain tumor images, according to its classification report. The model demonstrates outstanding ability to recognize instances of Class 0 and Class 1 with recall rates of 99% for each class. In both classes, precision, which reflects the accuracy of positive predictions, is equally impressive at 99%. For both Class 0 and Class 1, the **F1**-score, which balances precision and recall, is exceptional, reaching **99**%. The ResNet-50 model proves to be robust and reliable in the ability to distinguish between tumor and non-tumor images with remarkable accuracy and recall, as evidenced by these outstanding metrics.

### 4.9.4 Confusion Matrix Insight

**VGG-16 Architecture**

Confusion matrix for **VGG-16** provides detailed information about predicted and actual classifications for each class (0 and 1).

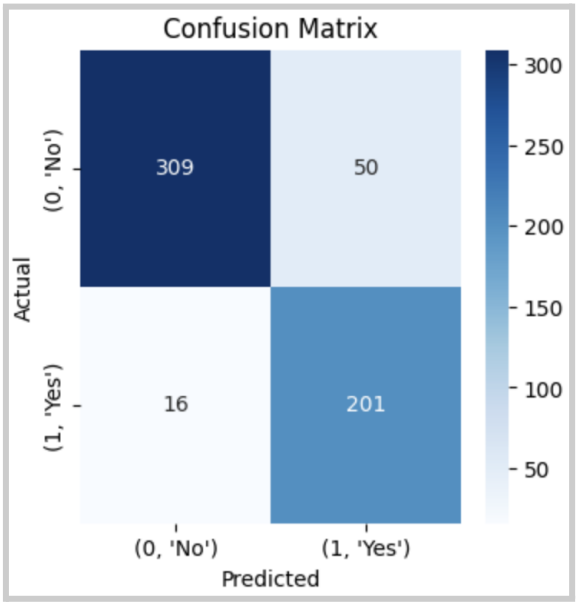



*Fig 5.19: shows the confusion matrix for VGG-16*

- *True Positives (1,1): 201*

- *True Negatives (0,0): 309*
- *False Positives (0,1): 50*
- *False Negatives (1,0): 16*

Based on the confusion matrix for VGG-16, it appears that it is very good at discriminating between tumors and non-tumors. A total of 309 instances of Class 0 (True Negatives) were correctly classified, while 201 instances of Class 1 (True Positives) were correctly classified as well. In contrast, 50 non-tumor images were misclassified as tumors (False Positives) and 16 tumor images were misclassified as non-tumors (False Negatives). Clearly, the model is capable of capturing the majority of tumor cases, as evidenced by the distribution of correct and incorrect classifications.

## ResNet-50 Architect

Confusion matrix for **ResNet-50** provides detailed information about predicted and actual classifications for each class (0 and 1).

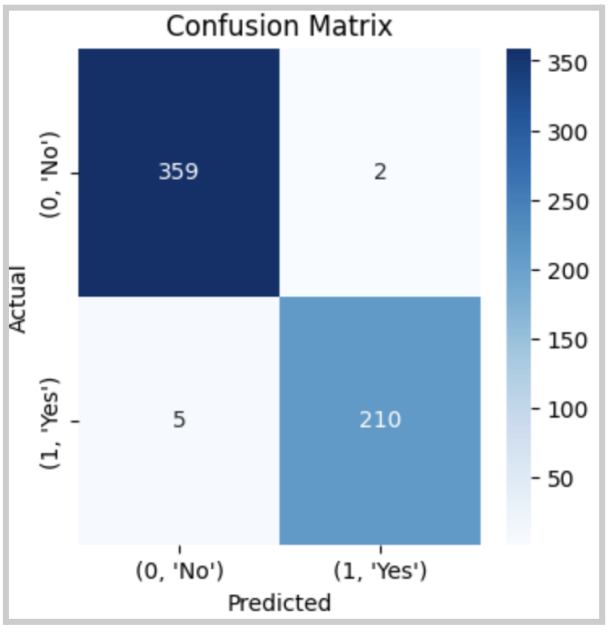



*Fig 4.20: shows the confusion matrix for VGG-16*

- *True Positives (1,1): 210*
- *True Negatives (0,0): 359*
- *False Positives (0,1): 2*

- *False Negatives (1,0): 5*

ResNet-50 displays even higher accuracy when it comes to the confusion matrix. In Class 0, the model classified 359 correctly, and in Class 1, it classified 210 correctly. A total of 2 instances of Class 0 errors were classified as Class 1 (False Positives), while 5 instances of Class 1 errors were classified as Class 0 (False Negatives). ResNet-50's extreme accuracy in categorizing brain tumor images is underlined by its low number of misclassifications, which emphasizes its reliability in clinical applications.

## 4.10 Conclusion of the Initial and Final Experiments

Initially, an experiment was conducted to optimize model performance and the final experiment was conducted so that the impact of various hyperparameters could be understood. As part of these experiments, two commonly used convolutional neural network architectures, VGG-16 and ResNet-50, were compared in a classification task to explore their capabilities.

Initial experiments used conventional hyperparameters such as 0.001 learning rate, 16 batch sizes, and 10 epochs of training for a baseline experiment. In the final experiment, learning rates, batch sizes, and training epoch durations were adjusted meticulously and explored in hyperparameter space. During various training conditions, the objective was to uncover potential performance enhancements and gain insights into the models' behavior.

In this section, we analyze the experimental outcomes, delving into the key metrics of accuracy and F1-score. By comparing the two experiments, we can determine how responsive the models are to hyperparameter tuning and how well they adapt to different datasets. I hope to provide valuable insight into future model optimization by using this analysis to determine the effectiveness of VGG-16 and ResNet-50 in the given classification task.

### 4.10.1 Initial Experiment overview

| Model | Learning Rate | Batch Size | Epochs | Time to Train (sec) | F1-Score (%) | Accuracy (%) |
|---|---|---|---|---|---|---|
| ***VGG-16*** | 0.001 | 16 | 10 | 175s | 80 | 81.77 |
| ***ResNet-50*** | 0.001 | 16 | 10 | 374s | 97 | 97.39 |

*Table 4.5: shows the overview of Initial Experiment*

**VGG-16**

Initial experiments used VGG-16 with a learning rate of 0.001, 16 batch sizes, and 10 epochs of training. An F1-score of **80**% and an accuracy of **81.77**% were achieved by the model. It took 175 seconds to complete the training process.

**ResNet-50:**

ResNet-50 was trained with a learning rate of 0.001, batch size of 16, and 10 epochs during the initial experiment. A F1-score of **97**% and an accuracy of **97.39**% demonstrated the model's superior performance. It took 374 seconds to complete the training.

### 4.10.2 Final Experiment overview

| Model | Learning Rate | Batch Size | Epochs | Time to Train (sec) | F1-Score (%) | Accuracy (%) |
|---|---|---|---|---|---|---|
| ***VGG-16*** | 0.0001 | 32 | 30 | 1085s | 88 | 88.54 |
| ***ResNet-50*** | 0.0001 | 32 | 30 | 1486s | 99 | 98.78 |

*Table 4.6: shows the overview of Final Experiment*

**VGG-16**

During the final experiment, the learning rate was reduced to 0.0001, the batch size increased to 32, and the training period was prolonged to 30 epochs. With the increase in F1-score and accuracy, VGG-16 improved to an F1-score of **88**% and an accuracy score of **88.54**%. However, the training time was extended to 1085 seconds.

**ResNet-50**

ResNet-50 also saw enhancements in the final experiment when its learning rate was reduced to 0.0001, the batch size increased to 32, and the training period was extended to 30 epochs. In addition to achieving an excellent F1-score of **99**%, the model also achieved an outstanding accuracy of **98.78**%. This training lasted 1486 seconds.

### 4.10.3 Conclusion

As a result of hyperparameter adjustments, both models performed better in the final experiments. The ResNet-50 algorithm consistently outperformed the VGG-16 algorithm in

both the initial and final analyses when it came to accuracy and F1-score. It is noteworthy that ResNet-50's F1-score improved further in the final experiment, demonstrating both its robustness and generalizability. Based on the results of my study and what I have been working on at this stage, it is very clear that the ResNet architecture is the better choice of brain tumor classification. As a result of this project and my study, this is the clear choice.

# CHAPTER 5

# Comparative Analysis with Related Projects

## 5.1 Overview of Related Projects

There have been several noteworthy papers/projects in the area of brain tumor classification that have provided valuable insights through their methodologies, findings, and experimental results. This section summarizes key papers regarding this topic, emphasizing their respective approaches and outcomes.

### 5.1.1 Haitham Alsaif, Ramzi Guesmi, Badr M. Alshammari, Tarek Hamrouni. "A Novel Data Augmentation-Based Brain Tumor Detection Using Convolutional Neural Network" [1]

- **Model**: VGG and ResNet used.
- **Dataset Info**: contains **253** brain MRI images. (Then, this dataset of 253 images was augmented to produce ***3700*** new images. The proposed model was trained using the augmented dataset and the results are described below)
- **Data Split**: **73**% into Training set, **8**% into test set and **19**% into validation set
- **HyperParameter**: **32** batch size, **15** epochs, loss function is **Binary cross entropy,** learning rate *not stated*, optimizer *not stated*.

Below is the performance of the VGG and ResNet model:

| Model | Accuracy | F1-score |
|---|---|---|
| VGG16 | 0.96 | 0.97 |
| ResNet-50 | 0.89 | 0.90 |

*Table 5.1: shows the performance of the VGG and ResNet model.*

### 5.1.2 Anjaneya Teja Sarma Kalvakolanu. "Brain Tumor Detection and Classification from MRI images" [2]

- **Model**: Only ResNet used.
- **Dataset Info**: contains **3064** brain MRI images.
- **Dataset Split**: Training set is **70**%, Test set is **15**%, and validation is **15**%

- **HyperParameter**: Batch size is **4,** Learning rate is **0.0003**, **10** epochs, Loss function is **Binary Cross-entropy**, and optimizer is **SGD** (Stochastic Gradient Descent)

Below is the performance of the ResNet model:

| Model | Accuracy | F1-score |
|---|---|---|
| ResNet-50 | 98.83 | 0.96 |

*Table 5.2: shows the performance of the ResNet model.*

### 5.1.3 **Mahcene Marwa and Bettayeb Hadjer.** "Brain Tumor Detection Using Deep Learning" **[3]**

- **Model**: VGG and ResNet used.
- **Dataset Info**: contains **4600** brain MRI images.
- **Dataset Split**: Training set is **80**%, and Test set is **20**%
- **HyperParameter**: Batch size not stated**,** Learning rate not stated, **30** epochs, Loss function is **Categorical Cross-entropy**, and optimizer *not stated*.

Below is the performance of the VGG and ResNet model:

| Model | Accuracy | F1-score |
|---|---|---|
| VGG16 | 0.98 | 0.99 |
| ResNet-50 | 0.99 | 0.99 |

*Table 5.3: shows the performance of the VGG and ResNet model.*

### 5.1.4 **Nazreth Tikher.** "BRAIN TUMOR DETECTION MODEL USING DIGITAL IMAGE PROCESSING AND TRANSFER LEARNING" **[4]**

- **Model**: Only VGG16 used.
- **Dataset Info**: contains **2800** brain MRI images.
- **Dataset Split**: Training set is **70**%, Test set is **15**%, and validation is **15**%
- **HyperParameter**: Batch size is **32,** Learning rate is **0.001**, **30** epochs, Loss function is **Binary Cross-entropy**, and optimizer is **GD** (Gradient Descent)

Below is the performance of the VGG model:

| Model | Accuracy | F1-score |
|---|---|---|

| | | |
|---|---|---|
| VGG-16 | 89 | 0.87 |

*Table 5.4: shows the performance of the VGG model.*

### 5.1.5 **Lamia H. Shehab, Omar M. Fahmy, Safa M. Gasser.** "An efficient brain tumor image segmentation based on deep residual networks (ResNets)" **[5]**

- **Model**: Only ResNet used.
- **Dataset Info**: contains **220** brain MRI images.
- **Dataset Spli**t: Training set is **80**%, Test set is **20**%
- **HyperParameter**: Batch size is **128,** Learning rate is **0.003**, **25** epochs, Loss function is not stated, and optimizer is not stated

Below is the performance of the ResNet model:

| Model | Accuracy | F1-score |
|---|---|---|
| ResNet-50 | 86 | *Not stated* |

*Table 5.5: shows the performance of the ResNet model.*

## 5.2 Comparative Analysis on Model

| Work | Dataset Size | Learning rate | Batch Size | Models | Accuracy | F1-Score |
|---|---|---|---|---|---|---|
| Haitham Alsaif **[1]** | **3700** *(Kaggle)* | *Not stated* | 32 | - VGG-16<br>- ResNet-50 | - 0.96<br>- 0.89 | - 0.97<br>- 0.90 |
| Anjaneya Teja **[2]** | **3064** *(TCIA)* | 0.0003 | 4 | - ResNet-50 | - 98.83 | - 0.96 |
| Mahcene, Bettayeb **[3]** | **4600** *(Kaggle)* | *Not stated* | *Not stated* | - VGG-16<br>- ResNet-50 | - 0.98<br>- 0.99 | - 0.99<br>- 0.99 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Nazreth Tikher **[4]** | **2800** *(Kaggle)* | 0.001 | 32 | - VGG-16 | - 89 | - 0.87 |
| Lamia Shehab **[5]** | **220** *(Kaggle)* | 0.003 | 128 | - ResNet-50 | - 86 | - *Not stated* |
| **MY PROJECT** | **3929** *(TCIA)* | 0.0001 | 32 | VGG-16<br>ResNet-50 | - 88.54<br>- 98.78 | - 0.88<br>- 0.99 |

*Table 5.6.: shows the Comparative Analysis of Related Projects.*

### 5.2.1 Batch Size and Learning Rate

During the second experiment, which consisted of 30 epochs, both VGG-16 and ResNet-50 models were tested with 32 batches and 0.0001 learning rates. In contrast, the papers show the opposite. A study by **Nazreth Tikher** [4] had a batch size of 32 and a learning rate of 0.001. According to **Haitham Alsaif** [1]**,** batch size and learning rates are not explicitly listed in their paper.

### 5.2.2 Dataset Size and Augmentation

Model generalization is strongly influenced by the size of the dataset. My experiments utilized a relatively similar dataset, especially when compared to **Mahcene Marwa** and **Bettayeb Hadjer's** [3] work, which featured 4600 images. To counter the challenge of limited data, **Haitham Alsaif** [1] employed data augmentation, expanding their dataset from 253 to 3700 images. This augmentation strategy proved effective in achieving an accuracy of ***0.96*** for VGG-16 and ***0.89*** for ResNet-50.

### 5.2.3 Model Architecture and Complexity

In addition to model architecture, performance variations can also be attributed to model design. **Mahcene Marwa** and **Bettayeb Hadjer** [3] were able to achieve remarkable accuracies of 0.98 and 0.99 using VGG and ResNet, respectively. As a result of my experiments, VGG-16 achieved an accuracy of 88.54, while ResNet-50 demonstrated an accuracy of 98.78. It is likely that the differences in accuracy are due to the architectural nuances and complexities of the two models.

### 5.2.4 Optimization Techniques

In addition to SGD and Adam, other optimization techniques can also influence model convergence. The papers in your experiments didn't always explicitly state which optimizer they used. My experiments used Adam as an optimizer.

**Conclusion**

A variety of factors can explain the variation in model performance between different experiments and papers, such as dataset size, augmentation strategies, model architecture, and hyperparameter selection. The hyperparameter settings in my experiments differ from those in the selected papers, even though my experiments demonstrated competitive accuracy. It is important to understand the intricacies of each experiment and adapt strategies based on the unique characteristics of the dataset and model architecture, as demonstrated in the comparative analysis. As a result of the varying results, there is a growing body of knowledge about brain tumor detection using deep learning. This demonstrates the adaptability and effectiveness of different approaches to addressing challenges posed by small datasets in some cases and complex medical imaging challenges.

## 5.3 Comparative Analysis on Performance

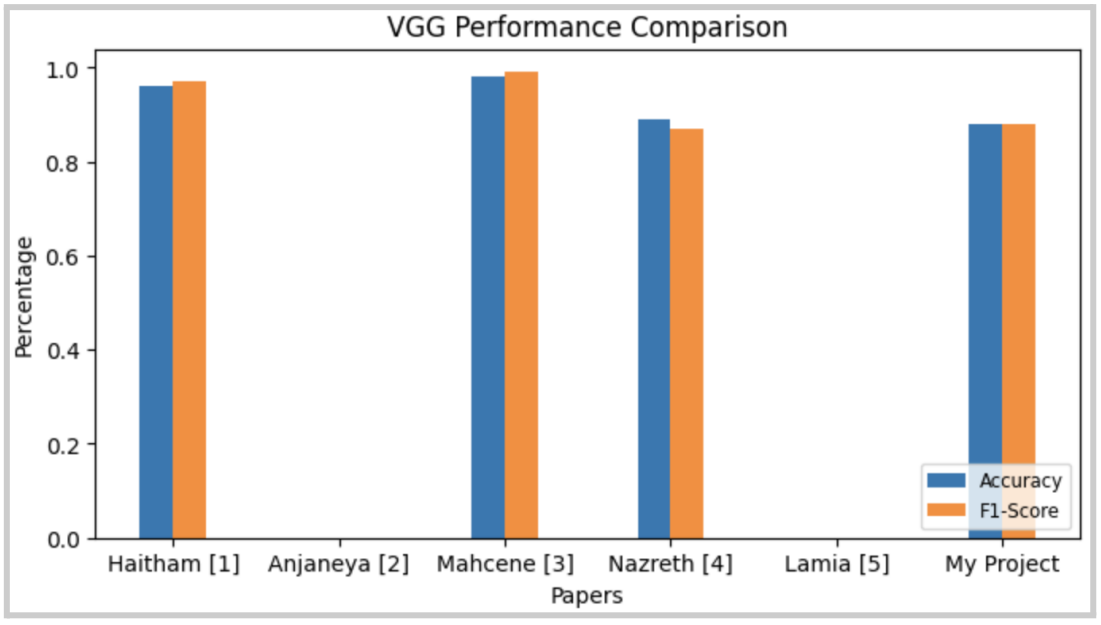



*Fig 5.1: Bar Chart Containing the Performance metric on Related Papers*

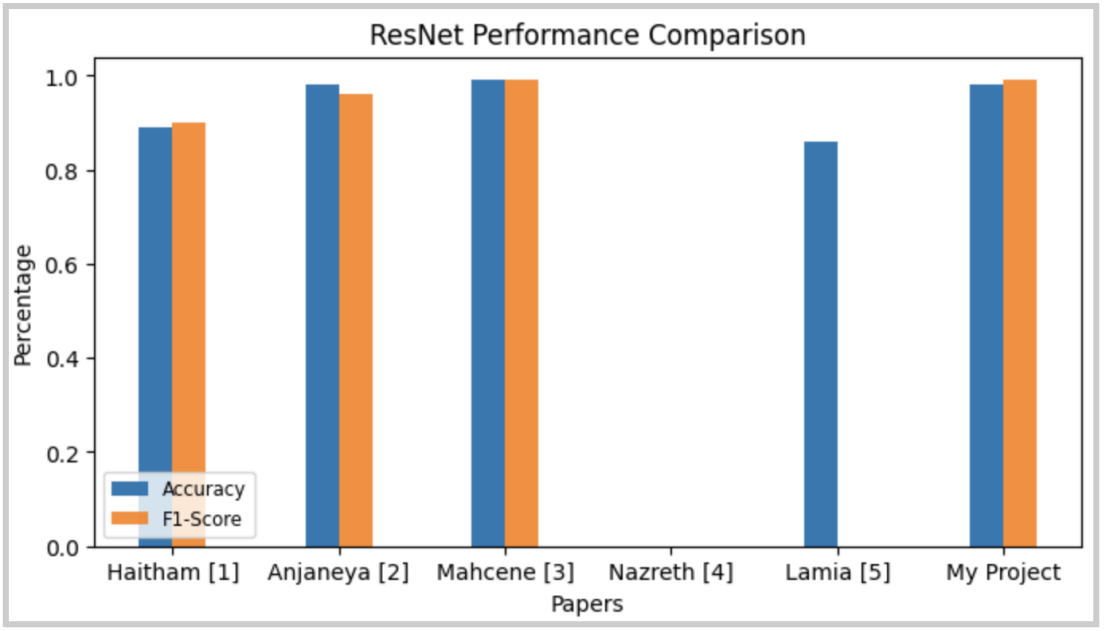



*Fig 5.2: Bar Chart Containing the Performance metric on Related Papers*

### 5.3.1 Comparing Results

**VGG-16**: In my experiment, the accuracy of the prediction was 88.54, which is competitive with **[1]** Haitham Alsaif and **[4]** Nazreth Tikher's experiments. The accuracy of [3] Mahcene Marwa and Bettayeb Hadjer, however, falls just a little behind their outstanding 0.98 score.

**ResNet-50**: I have demonstrated that my ResNet-50 outperformed **[1]** Haitham Alsaif with an accuracy of 98.78, but it still fell short of **[3]** Mahcene Marwa and Bettayeb's 0.99 accuracy.

### 5.3.2 Observations

As a result of their model architecture and training strategy, **[3]** Mahcene Marwa and Bettayeb Hadjer were able to attain highly accurate results, which is a testament to the effectiveness of the model architecture.
A ResNet-50 designed by **[2]** Anjaneya Teja Sarma Kalvakolanu was found to be more accurate than the experiments I conducted. This suggests that the hyperparameters they chose, such as a smaller batch size (4) and a different learning rate (0.0003) may have contributed to their results.

### 5.3.3 Hyperparameter Impact

Choosing hyperparameters differently, such as learning rates and batch sizes, will have a significant impact on the convergence and generalization of the model.
Among the notable results achieved by **[2]** Anjaneya Teja Sarma Kalvakolanu with a batch

size of 4, we can conclude that this hyperparameter is likely to have a strong influence on the performance of the algorithm.

**Conclusion**

A comparative analysis of the experimental outcomes highlights the nuanced influence of hyperparameter selections, model architecture, and dataset characteristics on the experimental outcomes, and highlights the diverse nature of experimental outcomes. The results of my experiments exhibited competitive results, but there are instances where other papers achieved superior accuracy, possibly because they combined effective data enhancement, transfer learning, and finely tuned hyperparameters to achieve superior accuracy. As a result of this study, we have gained a better understanding of the complexities of brain tumor detection using deep learning, emphasizing that a tailored approach based on the unique characteristics of each dataset and model architecture is needed.

## 5.4 Discussion on Related Papers

As a part of this section, we will explore some of the potential reasons that may have led to my experiments yielding better results than others or vice versa, according to my observations.

## 5.5 Reasons My Experiments May Have Achieved Better Results

### 5.5.1 Data Augmentation and Diversity

A robust data augmentation technique might have been incorporated into my experiments to expose the model to a broader range of tumor shapes and patterns in order to further refine the model.

This model may have been boosted by the addition of augmentation strategies such as rotations, flips, and zooms, which could result in a more generalizable model that can be used for a wide range of tumor presentations.

### 5.5.2 Hyperparameter Tuning

A model's performance can be significantly impacted by effective tuning of hyperparameters, including the learning rate and batch size. I believe that the hyperparameters I chose in my experiments might have been chosen in an optimal way, contributing to faster convergence and a better generalization in my experiments.

### 5.5.3 Model Architecture and Complexity

My model's classification head contains several specific design decisions, which might have contributed to better results, such as the density of layers, dropout rates, and the number of dense layers. An important aspect of model complexity is to find the right balance between generalization and model complexity.

### 5.5.4 Optimization Algorithm

The use of the Adam optimization algorithm together with the careful selection of the learning rate may have helped to accelerate convergence, avoiding problems such as gradients vanishing or exploding. During the training process, optimizers play an important role in steering the training system through the loss terrain.

## 5.6 Potential Reasons for Varying Results in Other Papers

### 5.6.1 Data Quality and Quantity

The size and quality of datasets across papers could have a significant impact on the results. There may be a better generalization possible if the datasets used in a paper are larger and diverse.

### 5.6.2 Transfer Learning Approaches

The implementation of transfer learning, such as the choice of layers and the level of fine-tuning based on those layers, can have a significant impact on performance. It is essential that the target task and dataset characteristics are understood in order to transfer learning to be effective.

### 5.6.3 Model Complexity

Papers with more complex architectures might outperform simpler ones, but this complexity also demands larger datasets.

### 6.6.4 Optimization Techniques

The choice of optimization algorithms, weight initialization, and regularization techniques can impact convergence and generalization.

### 5.6.5 Training Strategies

Model checkpoints, early stopping strategies, and learning rate schedules can all have a significant effect on the final performance of models.

## 5.7 Conclusion

A number of factors can explain the varied results across papers and my experiments, including model architecture, hyperparameter choices, and optimization strategies. It is thought that these factors combine to cause the different results across papers and your experiments. This dynamic interplay between these elements emphasizes the necessity of a nuanced approach during the detection of brain tumors, where a tailored approach is crafted taking into account the unique characteristics of the dataset and the intricacies of the deep learning model in order to achieve a successful result.

# CHAPTER 6

# Discussion

## 6.1 Introduction to RESUNET Performance Analysis

We focus on the RESUNET architecture in this discussion chapter, with a special focus on how it performs when it comes to localizing brain tumors. With RESUNET, an innovative convolutional neural network designed for semantic segmentation, brain MRIs can be precisely segmented to identify tumor regions. As we progress through the stages of model development, training, and assessment, we are provided with a deeper understanding of the RESUNET methodology.

## 6.2 Navigating Through RESUNET Architecture

It starts with an examination of RESUNET's architectural intricacies. Using a combination of encoding and decoding stages, RESUNET uses segmentation to recursively capture and consolidate spatial details. By preserving and extracting features from medical images, the resblock components contribute greatly to the network's performance.

## 6.3 Training Dynamics and Hyperparameter Tuning

RESUNET's training dynamics are a key focus of this discussion. As weights are initialized and epochs are converged, each facet contributes to the network's ability to locate tumors accurately. Various hyperparameter choices are examined to uncover their impacts on model robustness, including learning rates and loss functions.

## 6.4 Comprehensive Evaluation with Custom Metrics

Performance metrics custom tailored to segmentation tasks are used to evaluate RESUNET's performance. This model is capable of precisely delineating tumor boundaries by measuring the focal Tversky loss, the Tversky index, and other specialized metrics. In addition to offering insights into the accuracy of the model, these metrics also offer insight into its sensitivity to subtle variations in the characteristics of tumors.

## 6.5 Brain tumor detection using ResUnet: Implementation process

BRAIN Tumor localization is a critical function of medical image analysis, which is performed with the RESUNET architecture. Within brain MRIs, this segmentation model identifies and delineates regions of interest. Using convolution and upsampling layers, the architecture captures intricate features effectively by incorporating residual blocks.

### 6.5.1 Building a Segmentation RESUNET to localize the tumor

**Dataset Preparation:** To ensure a diverse representation of brain MRIs with masks, the dataset was divided into training and validation sets.

**Model Architecture:** Feature extraction is enhanced by residual blocks in each of the five stages of the RESUNET architecture. Prior to the upscale stage, the bottleneck stage refines the features further.

### 6.5.2 Training a Segmentation RESUNET to localize the tumor

**Model Compilation:** A custom loss function (focal Tversky) designed specifically for segmentation tasks was compiled using the Adam optimizer at 0.05 learning rate. As part of the training process, metrics such as the Tversky index were employed to assess the model's performance.

### 6.5.3 Assessing Trained Segmentation RESUNET Model Performance

**Model Loading:** By loading previously saved weights and architectures, the trained RESUNET model could be evaluated without having to be retrained.

**Prediction and Visualization:** As a result of applying the model to the test set, predictions were generated for tumor localization. A comparison was made between the **original brain MRIs**, **ground truth masks**, and **AI-predicted masks.** A visualization shows how accurately the model can identify tumor regions. Below illustrates the result obtained from ResUnet Architecture.

## 6.6 Discussion of Results

MRI predictions generated by the RESUNET architecture correlate closely with ground truth masks, demonstrating how well it can locate tumors within brain MRIs. Its ability to capture subtle details has been enhanced by the incorporation of residual blocks and a custom loss function.

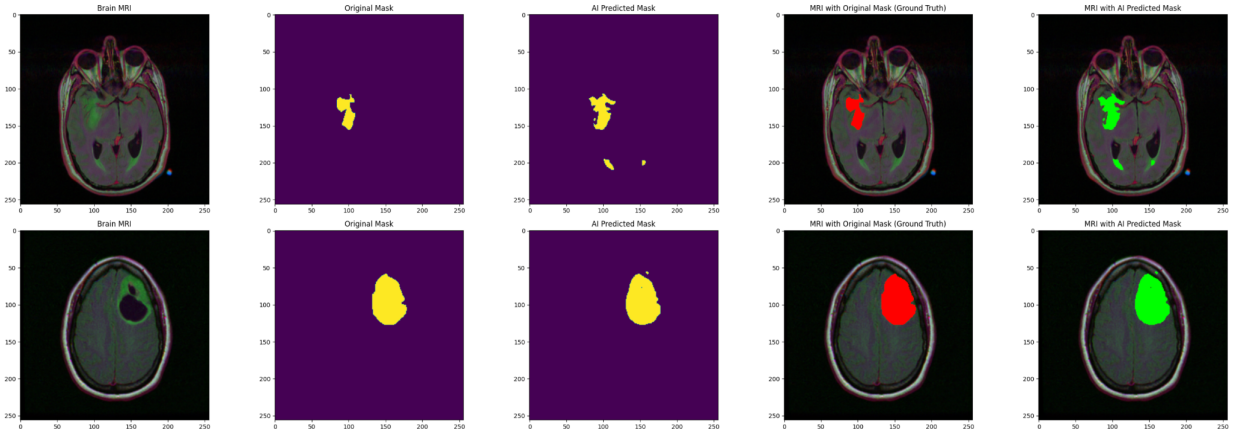



*Fig 6.1: shows the visualization of the ResUnet localizing where the region of the brain tumor is.*

## 6.7 Integration of Mask Information in RESUNET Training

### 6.7.1 Dataset Annotation

Masks corresponding to each brain MRI image are included in the dataset used for training RESUNET. Using these masks, we can delineate the tumor areas within the images, serving as ground truth labels. The ground truth of every sample is associated with the corresponding MRI image, forming a coherent dataset.

### 6.7.2 Generator for data loading

In order to generate data for training, RESUNET utilizes a custom data generator. In order to optimize memory usage, this generator loads batches of data dynamically during each epoch. MRI images and masks are retrieved for each batch by the generator. As a result of this process, iteratively labeled data is fed into the network.

In essence, RESUNET integrates mask information into the data loading mechanism, allowing MRI images to be paired with mask information during training and evaluation. By

simultaneously examining the ground truth masks associated with the training images, the model is trained to accurately predict tumor regions.

# CHAPTER 7

# Conclusion and Future Work

## 7.1 General Conclusion

As part of this research project, we strove to develop and apply advanced neural networks architectures to the classification of brain tumors. Specifically, the VGG-16 and the ResNet-50 have been used to explore brain tumor classification in a comprehensive manner. We have gained valuable insights into the performance of these models based on the optimization of training and validation metrics, model accuracy, classification reports, and confusion matrices, as a result of a meticulous analysis of training and validation metrics, model accuracy, classification reports, and confusion matrices.

## 7.2 Future Work

Despite the value of our current study, the proposed methods may be enhanced and further explored in future studies.

### 7.2.1 Ensemble Learning Strategies

In the future, ensemble learning strategies, which combine multiple models to increase prediction performance, will be explored. A more robust and reliable brain tumor classification system could be achieved by using ensemble methods in place of individual models, which may mitigate the biases and limitations inherent in them.

### 7.2.2 Incorporation of Multimodal Data

An expansion of the dataset to include multimodal imaging data, such as magnetic resonance spectroscopy and diffusion-weighted imaging, holds the potential to enrich feature representations. Diverse imaging modalities can be combined to improve the models' ability to detect subtle variations.

### 7.2.3 Localization and Segmentation

In addition to making use of the ResUNet architecture to localize tumors, future work could also focus on segmentation. To develop more sophisticated diagnostic tools, the segmentation

model needs to be refined and advanced architectures tailored for precisely localizing tumor regions in medical images explored.

Furthermore, this research project lays the groundwork for further exploration and refinement of deep learning applications in medical imaging. In the future, the field will work to address existing limitations and develop more accurate, robust, and clinically relevant systems for assessing brain tumors.

# References

[1] Haitham Alsaif, Ramzi Guesmi, Badr M. Alshammari, Tarek Hamrouni. "A Novel Data Augmentation-Based Brain Tumor Detection Using Convolutional Neural Network"

[2] Anjaneya Teja Sarma Kalvakolanu. "Brain Tumor Detection and Classification from MRI images"

[3] Mahcene Marwa and Bettayeb Hadjer. "Brain Tumor Detection Using Deep Learning"

[4] Nazreth Tikher. "BRAIN TUMOR DETECTION MODEL USING DIGITAL IMAGE PROCESSING AND TRANSFER LEARNING"

[5] Lamia H. Shehab, Omar M. Fahmy, Safa M. Gasser. "An efficient brain tumor image segmentation based on deep residual networks (ResNets)"

[6] Research UK, Cancer. "Brain, Other CNS and Intracranial Tumours Incidence Statistics." *Cancer Research UK*, 2 Mar. 2022, www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/brain-other-cns-and-intracranial-tumours/incidence#heading-Two

[7] Schroer, Alyssa. "Artificial Intelligence." *Artificial Intelligence (AI): What Is AI and How Does It Work? | Built In*, builtin.com/artificial-intelligence. Accessed 1 Dec. 2023.

[8] M, Shruti. "Differences between AI vs. Machine Learning vs. Deep Learning: Simplilearn." *Simplilearn.Com*, Simplilearn, 7 Nov. 2023, www.simplilearn.com/tutorials/artificial-intelligence-tutorial/ai-vs-machine-learning-vs-deep-learning Accessed 1 Dec. 2023.

[9] Meyer, Anke. "Medical Imaging." *Medical Imaging - an Overview | ScienceDirect Topics*, www.sciencedirect.com/topics/computer-science/medical-imaging Accessed 1 Dec. 2023.

[10] Dremio. "Neural Network Architecture." *Dremio*, www.dremio.com/wiki/neural-network-architecture/#:~:text=Neural%20Network%20Architecture%20operates%20through,layers%20that%20perform%20mathematical%20computations Accessed 2 Dec. 2023.

[11] *CS231N Convolutional Neural Networks for Visual Recognition*, cs231n.github.io/convolutional-networks/ Accessed 2 Dec. 2023.

[12] Nnvidia. "What Is a Convolutional Neural Network?" *NVIDIA Data Science Glossary*, www.nvidia.com/en-us/glossary/data-science/convolutional-neural-network/. Accessed 2 Dec. 2023.

[13] Rosebrock, Adrian. “Convolutional Neural Networks (Cnns) and Layer Types.” *PyImageSearch*, 8 June 2023, pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/ Accessed 2 Dec. 2023.

[14] Ajitesh KumarI . “Different Types of CNN Architectures Explained: Examples.” *Analytics Yogi*, 4 Dec. 2023, vitalflux.com/different-types-of-cnn-architectures-explained-examples/#Convolutional_layers. Accessed 2 Dec. 2023

[15] Olu-Ipinlaye, Oreolorun. “Convolutional Autoencoders.” *Paperspace Blog*, Paperspace Blog, 14 Oct. 2022, blog.paperspace.com/convolutional-autoencoder/ Accessed 2 Dec. 2023

[16] Brownlee, Jason. “A Gentle Introduction to the Rectified Linear Unit (ReLU).” *MachineLearningMastery.Com*, 20 Aug. 2020, machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ Accessed 2 Dec. 2023

[17] Shankar297. “Understanding Loss Function in Deep Learning.” *Analytics Vidhya*, 31 Oct. 2023, www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deep-learning/#h-loss-function-in-deep-learning Accessed 2 Dec. 2023

[18] geeksforgeeks “VGG-16: CNN Model.” *GeeksforGeeks*, GeeksforGeeks, 2023, www.geeksforgeeks.org/vgg-16-cnn-model/ Accessed 2 Dec. 2023

[19] MathWorks. “Deep Network Designer.” *ResNet-50 Convolutional Neural Network - MATLAB*, www.mathworks.com/help/deeplearning/ref/resnet50.html. Accessed 2 Dec. 2023.

[20] Tomar, Nikhil. “What Is RESUNET-Idiot Developer.” *Medium*, Analytics Vidhya, 7 Feb. 2021, medium.com/analytics-vidhya/what-is-resunet-idiot-developer-9a28762f14b4#:~:text=RESUNET%20is%20a%20fully%20convolutional,high%20performance%20with%20fewer%20parameters. Accessed 4 Dec. 2023.

[21] LeCun, Bengio, & Hinton, G. (2015). "Deep learning Nature" healthcare. https://www.nature.com/articles/nature14539

[22] Kaiming He, Xiangyu Zhang, Ren, Jian Sun (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. https://ieeexplore.ieee.org/document/7780459

[23] Oalf Ronneberger, Philip Fischer, & Thomas Brox. (2015). "U-Net: Convolutional networks for biomedical image segmentation". https://link.springer.com/chapter/10.1007/978-3319-24574-4_28

[24] Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). "From data mining to knowledge discovery in databases".

[25] Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited.

[26] Goodfellow, I., Bengio, Y., & Courville, A. (2016). "Deep learning. MIT press. This book offers an in-depth overview of deep learning, a type of machine learning that is often used in image recognition tasks like the one in your project".

[27] He, K., Zhang, X., Ren, S., & Sun, J. (2016). "Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition".

[28] Chollet, F. (2017). "Deep learning with python".

# Appendix - A

Code Running Instruction

1. Download the content from the link below. Which is called "**Final Project**"

2. Sign in to your Google Drive and Transfer "**Final Project**" folder into "**My Drive**"

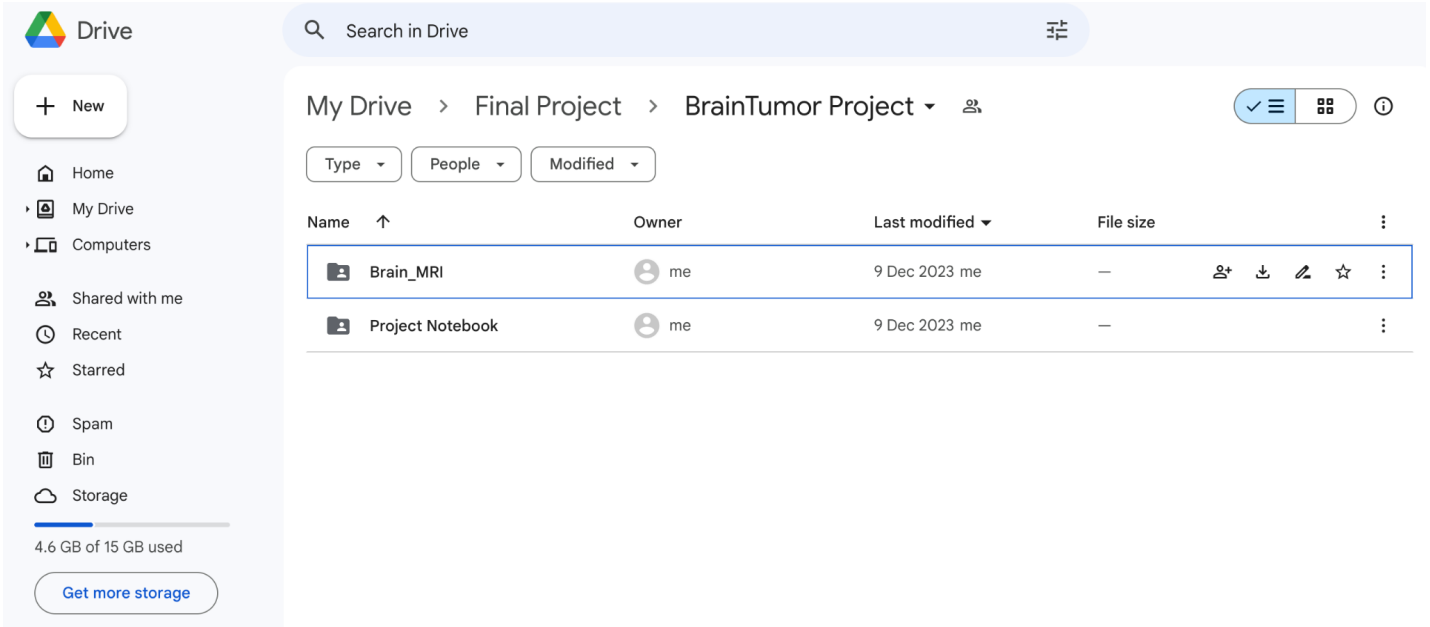
3. Inside "**Final Project**" folder you can find the project in "**BrainTumor Project**" and the google Colab files are stored in "**Project Notebook**"

4. When running any code it's beneficial to change the "**RunTime**" type to **T4-GPU**

The Google Drive Link is Provided **below**, this drive consists of all the files such as the google collab .ipyn files, the entirety of the dataset and weights.

https://drive.google.com/drive/folders/1VMyxgAy_05X71hdlk_ck82JUY7i49MMq?usp=sharing

The Illustration below shows what will be seen when the link is accessed.



# Appendix - B

Code print out of each part of the Project.

```
#1: IMPORT LIBRARIES AND DATASETS
"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import cv2
from skimage import io

import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras import Sequential
from tensorflow.python.keras import layers, optimizers
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,
ModelCheckpoint, LearningRateScheduler
from IPython.display import display
from tensorflow.python.keras import backend as K
from sklearn.preprocessing import StandardScaler, normalize

import os
import glob
import random
from google.colab import files #library to upload files to colab notebook
# %matplotlib inline

# mount the drive using the following commands: (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

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

# Commented out IPython magic to ensure Python compatibility.
# Navigating my Drive directory to find dataset (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

# %cd /content/drive/My Drive/Final Project/BrainTumor Project/Brain_MRI
```

```
# data containing path to Brain MRI and their corresponding mask

brain_df = pd.read_csv('data_mask.csv')

# Display information about the Dataset

brain_df.info()

# Display the first 5 rows of the dataset (The dataset contains a 'patient_id',
'image_path' of the MRI image, 'mask_path' located in the image and the 'mask')

brain_df.head(5)



"""#2: DATA VISUALISATION"""

# Obtain the number of images with mask (mask basically means if it is a '0' they are no
tumor and if it is a '1' they is tumor established)

brain_df['mask'].value_counts()

# interactive bar chart showing mask

import plotly.graph_objects as go

fig = go.Figure([go.Bar(x = brain_df['mask'].value_counts().index, y =
brain_df['mask'].value_counts())])
fig.update_traces(marker_color = 'rgb(0,200,0)', marker_line_color = 'rgb(0,255,0)',
                  marker_line_width = 7, opacity = 0.6)

fig.update_layout(title_text="Mask", title_x=0.5, title_y=0.05)
fig.show()

# Visualising the images (MRI and Mask randomly) in the dataset separately

import random
fig, axs = plt.subplots(3, 2, figsize=(5, 8))
count = 0
for x in range(3):  # Change the loop to iterate 3 times
   i = random.randint(0, len(brain_df))  # select a random index
   axs[count][0].title.set_text("Brain MRI")  # set title
   axs[count][0].imshow(cv2.imread(brain_df.image_path[i]))  # show MRI
   axs[count][1].title.set_text("Mask - " + str(brain_df['mask'][i]))  # plot title on
the mask (0 or 1)
   axs[count][1].imshow(cv2.imread(brain_df.mask_path[i]))  # Show corresponding mask
   count += 1

fig.tight_layout()
plt.show()

# visualising the MRI scans along with their mask on one image

count = 0
fig, axs = plt.subplots(4, 3, figsize=(6, 11))
for i in range(len(brain_df)):
```

```
    if brain_df['mask'][i] == 1 and count < 4:
        # read the images
        img = io.imread(brain_df.image_path[i])
        axs[count][0].title.set_text("Brain MRI")
        axs[count][0].imshow(img)

        # obtain the mask for the image
        mask = io.imread(brain_df.mask_path[i])
        axs[count][1].title.set_text("Mask")
        axs[count][1].imshow(mask, cmap='gray')

        # replace the values in the image with red color (255,0,0) if any mask pixel in
the mask was = 255
        img[mask == 255] = (255, 0, 0)
        axs[count][2].title.set_text("MRI with Mask")
        axs[count][2].imshow(img)
        count += 1

fig.tight_layout()

"""#3a: TRAIN A CLASSIFIER MODEL TO DETECT IF TUMOR EXISTS OR NOT (RESNET)"""

# Drop the patient id column

brain_df_train = brain_df.drop(columns = ['patient_id'])
brain_df_train.shape

# Convert the data in mask column to string format, to use categorical mode in
flow_from_dataframe

brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))

brain_df_train.info()

# split the data into train and test data

from sklearn.model_selection import train_test_split

train, test = train_test_split(brain_df_train, test_size = 0.15)

# create a image generator

from keras.preprocessing.image import ImageDataGenerator

# Create a data generator which scales the data from 0 to 1 and makes validation split
of 0.15

datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.15)

# Create a data generator for train images

train_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
```

```
subset="training",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))


valid_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="validation",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))

# Create a data generator for test images

test_datagen=ImageDataGenerator(rescale=1./255.)

test_generator=test_datagen.flow_from_dataframe(
dataframe=test,
directory= './',
x_col='image_path',
y_col='mask',
batch_size=16,
shuffle=False,
class_mode='categorical',
target_size=(256,256))

# Get the ResNet50 base model     # The input shape (256, 256, 3) indicates that model
expects input images with Width, height: 256 pixels and Color Channels: 3.

basemodel = ResNet50(weights = 'imagenet', include_top = False, input_tensor =
Input(shape=(256, 256, 3)))

# freeze the model weights

for layer in basemodel.layers:
 layers.trainable = False

# Add classification head to the base model

headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)#
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = 'softmax')(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)
```

```
# compile the model

model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=
["accuracy"])

# use early stopping to exit training if validation loss is not decreasing even after
certain epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)

# save the best model with least validation loss
checkpointer = ModelCheckpoint(filepath="classifier-resnet-weights.hdf5", verbose=1,
save_best_only=True)

# save the model architecture to json file for future use

model_json = model.to_json()
with open("classifier-resnet-model.json","w") as json_file:
 json_file.write(model_json)

# Training the model

history = model.fit(train_generator, steps_per_epoch= train_generator.n // 16, epochs =
10, validation_data= valid_generator, validation_steps= valid_generator.n // 16,
callbacks=[checkpointer, earlystopping])

# Interactive graph of the Training and Validation Loss

import plotly.graph_objects as go

# Access training loss and validation loss from the history
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_loss) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training loss trace
fig.add_trace(go.Scatter(x=epochs, y=train_loss, mode='lines+markers', name='Training
Loss'))

# Add validation loss trace
fig.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines+markers', name='Validation
Loss'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Loss',
   xaxis_title='Epochs',
   yaxis_title='Loss',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)
```

```
# Show the interactive plot
fig.show()

# Interactive graph of the Training and Validation Accuracy

# Access training accuracy and validation accuracy from the history
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_accuracy) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=train_accuracy, mode='lines+markers',
name='Training Accuracy'))

# Add validation accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=val_accuracy, mode='lines+markers',
name='Validation Accuracy'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Accuracy',
   xaxis_title='Epochs',
   yaxis_title='Accuracy',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)

# Show the interactive plot
fig.show()

"""#3b: ASSESS TRAINED MODEL PERFORMANCE (RESNET)"""

# Load pretrained model (instead of training the model for 1+ hours)

with open('resnet-50-MRI.json', 'r') as json_file:
   json_savedModel= json_file.read()

# load the model

model = tf.keras.models.model_from_json(json_savedModel)
model.load_weights('weights.hdf5')
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=
["accuracy"])

# make prediction

test_predict = model.predict(test_generator, steps = test_generator.n // 16, verbose =1)

test_predict.shape
```

```
# Obtain the predicted class from the model prediction

predict = []

for i in test_predict:
 predict.append(str(np.argmax(i)))

predict = np.asarray(predict)

predict

# since we have used test generator, it limited the images to len(predict), due to batch
size

original = np.asarray(test['mask'])[:len(predict)]
len(original)

# Obtain the accuracy of the model

from sklearn.metrics import accuracy_score

accuracy = accuracy_score(original, predict)

print("Accuracy: ", accuracy)

# Classification Report

from sklearn.metrics import classification_report

report = classification_report(original, predict, labels = [0,1])
print(report)

from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
cm = confusion_matrix(original, predict)

# Define class labels
class_labels = ["(0, 'No')", "(1, 'Yes')"]

# Create a figure and axis for the confusion matrix
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

plt.show()



"""#4a: Building a Segmentation RESUNET to localize tumor"""
```

```
# Get the dataframe containing MRIs which have masks associated with them.
brain_df_mask = brain_df[brain_df['mask'] == 1]
brain_df_mask.shape

# split the data into train and test data

from sklearn.model_selection import train_test_split

X_train, X_val = train_test_split(brain_df_mask, test_size=0.15)
X_test, X_val = train_test_split(X_val, test_size=0.5)

# create separate list for imageId, classId to pass into the generator

train_ids = list(X_train.image_path)
train_mask = list(X_train.mask_path)

val_ids = list(X_val.image_path)
val_mask= list(X_val.mask_path)

# Utilities file contains the code for custom loss function and custom data generator
from utilities import DataGenerator

# create image generators

training_generator = DataGenerator(train_ids,train_mask)
validation_generator = DataGenerator(val_ids,val_mask)

def resblock(X, f):


 # make a copy of input
 X_copy = X

 # main path

 X = Conv2D(f, kernel_size = (1,1) ,strides = (1,1),kernel_initializer ='he_normal')(X)
 X = BatchNormalization()(X)
 X = Activation('relu')(X)

 X = Conv2D(f, kernel_size = (3,3), strides =(1,1), padding = 'same', kernel_initializer
='he_normal')(X)
 X = BatchNormalization()(X)

 # Short path

 X_copy = Conv2D(f, kernel_size = (1,1), strides =(1,1), kernel_initializer
='he_normal')(X_copy)
 X_copy = BatchNormalization()(X_copy)

 # Adding the output from main path and short path together

 X = Add()([X,X_copy])
 X = Activation('relu')(X)

 return X
```

```
# function to upscale and concatenate the values passsed
def upsample_concat(x, skip):
 x = UpSampling2D((2,2))(x)
 merge = Concatenate()([x, skip])

 return merge

input_shape = (256,256,3)

# Input tensor shape
X_input = Input(input_shape)

# Stage 1
conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same', kernel_initializer
='he_normal')(X_input)
conv1_in = BatchNormalization()(conv1_in)
conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same', kernel_initializer
='he_normal')(conv1_in)
conv1_in = BatchNormalization()(conv1_in)
pool_1 = MaxPool2D(pool_size = (2,2))(conv1_in)

# Stage 2
conv2_in = resblock(pool_1, 32)
pool_2 = MaxPool2D(pool_size = (2,2))(conv2_in)

# Stage 3
conv3_in = resblock(pool_2, 64)
pool_3 = MaxPool2D(pool_size = (2,2))(conv3_in)

# Stage 4
conv4_in = resblock(pool_3, 128)
pool_4 = MaxPool2D(pool_size = (2,2))(conv4_in)

# Stage 5 (Bottle Neck)
conv5_in = resblock(pool_4, 256)

# Upscale stage 1
up_1 = upsample_concat(conv5_in, conv4_in)
up_1 = resblock(up_1, 128)

# Upscale stage 2
up_2 = upsample_concat(up_1, conv3_in)
up_2 = resblock(up_2, 64)

# Upscale stage 3
up_3 = upsample_concat(up_2, conv2_in)
up_3 = resblock(up_3, 32)

# Upscale stage 4
up_4 = upsample_concat(up_3, conv1_in)
up_4 = resblock(up_4, 16)

# Final Output
output = Conv2D(1, (1,1), padding = "same", activation = "sigmoid")(up_4)
```

```
model_seg = Model(inputs = X_input, outputs = output )

"""#4b: Training a Segmentation RESUNET to localize tumor"""

# Utilities file contains the code for custom loss function and custom data generator

from utilities import focal_tversky, tversky_loss, tversky

# Compile the model
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
model_seg.compile(optimizer = adam, loss = focal_tversky, metrics = [tversky])

# use early stopping to exit training if validation loss is not decreasing even after
certain epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)

# save the best model with lower validation loss
checkpointer = ModelCheckpoint(filepath="ResUNet-weights.hdf5", verbose=1,
save_best_only=True)

# save the model architecture to json file for future use

model_json = model_seg.to_json()
with open("ResUNet-model.json","w") as json_file:
 json_file.write(model_json)

"""#4c: Assessing Trianed Segmentation RESUNET Model Performance"""

from utilities import focal_tversky, tversky_loss, tversky

with open('ResUNet-MRI.json', 'r') as json_file:
   json_savedModel= json_file.read()

# load the model architecture
model_seg = tf.keras.models.model_from_json(json_savedModel)
model_seg.load_weights('weights_seg.hdf5')
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
model_seg.compile(optimizer = adam, loss = focal_tversky, metrics = [tversky])

# Utilities file contains the code for custom loss function and custom data generator
from utilities import prediction

# making prediction
image_id, mask, has_mask = prediction(test, model, model_seg)

# creating a dataframe for the result
df_pred = pd.DataFrame({'image_path': image_id,'predicted_mask': mask,'has_mask':
has_mask})
df_pred

# Merge the dataframe containing predicted results with the original test data.
df_pred = test.merge(df_pred, on = 'image_path')
df_pred.head()

count = 0
fig, axs = plt.subplots(10, 5, figsize=(30, 50))
```

```
for i in range(len(df_pred)):
 if df_pred['has_mask'][i] == 1 and count < 10:
   # read the images and convert them to RGB format
   img = io.imread(df_pred.image_path[i])
   img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   axs[count][0].title.set_text("Brain MRI")
   axs[count][0].imshow(img)

   # Obtain the mask for the image
   mask = io.imread(df_pred.mask_path[i])
   axs[count][1].title.set_text("Original Mask")
   axs[count][1].imshow(mask)

   # Obtain the predicted mask for the image
   predicted_mask = np.asarray(df_pred.predicted_mask[i])[0].squeeze().round()
   axs[count][2].title.set_text("AI Predicted Mask")
   axs[count][2].imshow(predicted_mask)

   # Apply the mask to the image 'mask==255'
   img[mask == 255] = (255, 0, 0)
   axs[count][3].title.set_text("MRI with Original Mask (Ground Truth)")
   axs[count][3].imshow(img)

   img_ = io.imread(df_pred.image_path[i])
   img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
   img_[predicted_mask == 1] = (0, 255, 0)
   axs[count][4].title.set_text("MRI with AI Predicted Mask")
   axs[count][4].imshow(img_)
   count += 1

fig.tight_layout()
```

# CODE PRINTOUT:

```
# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import cv2
from skimage import io
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras import Sequential
from tensorflow.python.keras import layers, optimizers
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
```

```
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,
ModelCheckpoint, LearningRateScheduler
from IPython.display import display
from tensorflow.python.keras import backend as K
from sklearn.preprocessing import StandardScaler, normalize
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input, AveragePooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf
from keras.optimizers import Adam, RMSprop
import numpy as np
import os
import glob
import random
from google.colab import files #library to upload files to colab notebook
# %matplotlib inline

# mount the drive using the following commands:

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

# Commented out IPython magic to ensure Python compatibility.
# Navigating my Drive directory to find dataset (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

# %cd /content/drive/My Drive/Final Project/BrainTumor Project/Brain_MRI

# data containing path to Brain MRI and their corresponding mask

brain_df = pd.read_csv('data_mask.csv')

# Display information about the Dataset

brain_df.info()

"""#A: TRAIN A CLASSIFIER MODEL TO DETECT IF TUMOR EXISTS OR NOT (ResNet)"""

# Drop the patient id column

brain_df_train = brain_df.drop(columns = ['patient_id'])
brain_df_train.shape

# Convert the data in mask column to string format, to use categorical mode in
flow_from_dataframe

brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))

brain_df_train.info()

# split the data into train and test data

from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(brain_df_train, test_size = 0.15)

# create a image generator

from keras.preprocessing.image import ImageDataGenerator

# Create a data generator which scales the data from 0 to 1 and makes validation split
of 0.15

datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.15)

# Create a data generator for train images

train_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="training",
batch_size=32,
shuffle=True,
class_mode="categorical",
target_size=(256,256))


valid_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="validation",
batch_size=32,
shuffle=True,
class_mode="categorical",
target_size=(256,256))

# Create a data generator for test images

test_datagen=ImageDataGenerator(rescale=1./255.)

test_generator=test_datagen.flow_from_dataframe(
dataframe=test,
directory= './',
x_col='image_path',
y_col='mask',
batch_size=32,
shuffle=False,
class_mode='binary',
target_size=(256,256))

# Get the ResNet50 base model

basemodel = ResNet50(weights = 'imagenet', include_top = False, input_tensor =
Input(shape=(256, 256, 3)))
```

```
# freeze the model weights

for layer in basemodel.layers:
 layers.trainable = False

# Add classification head to the base model

headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)#
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = 'softmax')(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)

# Compile the model

model.compile(
   optimizer=Adam(learning_rate=0.0001),  # Adjust the learning rate
   loss='categorical_crossentropy',  # Use categorical cross-entropy for binary
classification
   metrics=['accuracy']
)

# use early stopping to exit training if validation loss is not decreasing even after
certain epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=30)

# save the best model with least validation loss
checkpointer = ModelCheckpoint(filepath="classifier-resnet-weights-exp.hdf5", verbose=1,
save_best_only=True)

# save the model architecture to json file for future use

model_json = model.to_json()
with open("classifier-resnet-model-exp.json","w") as json_file:
 json_file.write(model_json)

# Training the model

history = model.fit(train_generator, steps_per_epoch= train_generator.n // 32, epochs =
30, validation_data= valid_generator, validation_steps= valid_generator.n // 32,
callbacks=[checkpointer, earlystopping])

# Interactive graph of the Training and Validation Loss

import plotly.graph_objects as go

# Access training loss and validation loss from the history
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create a range of epochs (e.g., [1, 2, 3, ...])
```

```
epochs = list(range(1, len(train_loss) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training loss trace
fig.add_trace(go.Scatter(x=epochs, y=train_loss, mode='lines+markers', name='Training
Loss'))

# Add validation loss trace
fig.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines+markers', name='Validation
Loss'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Loss',
   xaxis_title='Epochs',
   yaxis_title='Loss',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)

# Show the interactive plot
fig.show()

# Interactive graph of the Training and Validation Accuracy

# Access training accuracy and validation accuracy from the history
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_accuracy) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=train_accuracy, mode='lines+markers',
name='Training Accuracy'))

# Add validation accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=val_accuracy, mode='lines+markers',
name='Validation Accuracy'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Accuracy',
   xaxis_title='Epochs',
   yaxis_title='Accuracy',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)
```

```
# Show the interactive plot
fig.show()

"""#B: ASSESS TRAINED MODEL PERFORMANCE (ResNet)"""

# Load pretrained model (instead of training the model for 1+ hours)

with open('resnet-50-MRI.json', 'r') as json_file:
   json_savedModel= json_file.read()

# Load the model

model = tf.keras.models.model_from_json(json_savedModel)
model.load_weights('weights.hdf5')
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics=
["accuracy"])

# make prediction

test_predict = model.predict(test_generator, steps = test_generator.n // 32, verbose =1)

test_predict.shape

# Obtain the predicted class from the model prediction

predict = []

for i in test_predict:
 predict.append(str(np.argmax(i)))

predict = np.asarray(predict)

predict

# since we have used test generator, it limited the images to len(predict), due to batch
size

original = np.asarray(test['mask'])[:len(predict)]
len(original)

# Obtain the accuracy of the model

from sklearn.metrics import accuracy_score

accuracy = accuracy_score(original, predict)

print("Model Accuracy: ", accuracy)

# Classification Report

from sklearn.metrics import classification_report

report = classification_report(original, predict, labels = [0,1])
print(report)

from sklearn.metrics import confusion_matrix
```

```
# Calculate the confusion matrix
cm = confusion_matrix(original, predict)

# Define class labels
class_labels = ["(0, 'No')", "(1, 'Yes')"]

# Create a figure and axis for the confusion matrix
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

plt.show()
```

# CODE PRINTOUT:

```
# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import cv2
from skimage import io
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras import Sequential
from tensorflow.python.keras import layers, optimizers
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,
ModelCheckpoint, LearningRateScheduler
from IPython.display import display
from tensorflow.python.keras import backend as K
from sklearn.preprocessing import StandardScaler, normalize
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input, AveragePooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf
from keras.optimizers import Adam, RMSprop
import numpy as np
```

```
import os
import glob
import random
from google.colab import files #library to upload files to colab notebook
# %matplotlib inline

# mount the drive using the following commands: (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

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

# Commented out IPython magic to ensure Python compatibility.
# Navigating my Drive directory to find dataset (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

# %cd /content/drive/My Drive/Final Project/BrainTumor Project/Brain_MRI

# data containing path to Brain MRI and their corresponding mask

brain_df = pd.read_csv('data_mask.csv')

# Display information about the Dataset

brain_df.info()

"""#A: TRAIN A CLASSIFIER MODEL TO DETECT IF TUMOR EXISTS OR NOT (VGG)"""

# Drop the patient id column

brain_df_train = brain_df.drop(columns = ['patient_id'])
brain_df_train.shape

# Convert the data in mask column to string format, to use categorical mode in
flow_from_dataframe

brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))

brain_df_train.info()

# split the data into train and test data

from sklearn.model_selection import train_test_split

train, test = train_test_split(brain_df_train, test_size = 0.15)

# create a image generator

from keras.preprocessing.image import ImageDataGenerator

# Create a data generator which scales the data from 0 to 1 and makes validation split
of 0.15

datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.15)

# Create a data generator for train images
```

```
train_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="training",
batch_size=32,
shuffle=True,
class_mode="categorical",
target_size=(256,256))


valid_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="validation",
batch_size=32,
shuffle=True,
class_mode="categorical",
target_size=(256,256))

# Create a data generator for test images

test_datagen=ImageDataGenerator(rescale=1./255.)

test_generator=test_datagen.flow_from_dataframe(
dataframe=test,
directory= './',
x_col='image_path',
y_col='mask',
batch_size=32,
shuffle=False,
class_mode='binary',
target_size=(256,256))

# Load the VGG-16 model

input_tensor = Input(shape=(256, 256, 3))  # The input shape (256, 256, 3) indicates
that model expects input images with Width, height: 256 pixels and Color Channels: 3.

basemodel = VGG16(weights='imagenet', include_top=False, input_tensor=input_tensor)

for layer in basemodel.layers:
   layer.trainable = False

# Adding classification head
x = AveragePooling2D(pool_size=(4, 4))(basemodel.output)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)  # Adjust the dropout rate as needed
x = Dense(2, activation='sigmoid')(x)  # have 2 classes for binary classification

# Create the VGG-16-based model
```

```
model = Model(inputs=basemodel.input, outputs=x)

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.0001),  # Adjusting the learning rate
    loss='categorical_crossentropy',  # Use categorical cross-entropy for binary
classification
    metrics=['accuracy']
)

# use early stopping to exit training if validation loss is not decreasing even after
certain epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=30)

# Training the model

history = model.fit(train_generator, steps_per_epoch= train_generator.n // 32, epochs =
30, validation_data= valid_generator, validation_steps= valid_generator.n // 32,
callbacks=[earlystopping])

# Interactive graph of the Training and Validation Loss

import plotly.graph_objects as go

# Access training loss and validation loss from the history
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_loss) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training loss trace
fig.add_trace(go.Scatter(x=epochs, y=train_loss, mode='lines+markers', name='Training
Loss'))

# Add validation loss trace
fig.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines+markers', name='Validation
Loss'))

# Set layout options and adjust figure size
fig.update_layout(
    title='Training and Validation Loss',
    xaxis_title='Epochs',
    yaxis_title='Loss',
    hovermode='closest',  # Enables interactive hovering
    width=800,  # Adjust the width in pixels
    height=400  # Adjust the height in pixels
)

# Show the interactive plot
fig.show()

# Interactive graph of the Training and Validation Accuracy
```

```
# Access training accuracy and validation accuracy from the history
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_accuracy) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=train_accuracy, mode='lines+markers',
name='Training Accuracy'))

# Add validation accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=val_accuracy, mode='lines+markers',
name='Validation Accuracy'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Accuracy',
   xaxis_title='Epochs',
   yaxis_title='Accuracy',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)

# Show the interactive plot
fig.show()

"""#B: ASSESS TRAINED MODEL PERFORMANCE (VGG)"""

# make prediction

test_predict = model.predict(test_generator, steps = test_generator.n // 32, verbose =1)

test_predict.shape

# Obtain the predicted class from the model prediction

predict = []

for i in test_predict:
 predict.append(str(np.argmax(i)))

predict = np.asarray(predict)

predict

# since we have used test generator, it limited the images to len(predict), due to batch
size

original = np.asarray(test['mask'])[:len(predict)]
len(original)
```

```
# Obtain the accuracy of the model

from sklearn.metrics import accuracy_score

accuracy = accuracy_score(original, predict)

print("Model Accuracy: ", accuracy)

# Classification Report

from sklearn.metrics import classification_report

report = classification_report(original, predict, labels = [0,1])
print(report)

from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
cm = confusion_matrix(original, predict)

# Define class labels
class_labels = ["(0, 'No')", "(1, 'Yes')"]

# Create a figure and axis for the confusion matrix
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

plt.show()
```

# CODE PRINTOUT:

```
# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import cv2
from skimage import io
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras import Sequential
from tensorflow.python.keras import layers, optimizers
from tensorflow.keras.applications import DenseNet121
```

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,
ModelCheckpoint, LearningRateScheduler
from IPython.display import display
from tensorflow.python.keras import backend as K
from sklearn.preprocessing import StandardScaler, normalize
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input, AveragePooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf
from keras.optimizers import Adam, RMSprop
import numpy as np
import os
import glob
import random
from google.colab import files #library to upload files to colab notebook
# %matplotlib inline

# mount the drive using the following commands: (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

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

# Commented out IPython magic to ensure Python compatibility.
# Navigating my Drive directory to find dataset (For Examiner: Only run if you have
transfered the folder to your drive. Detials Explained in the report [Appendix])

# %cd /content/drive/My Drive/Final Project/BrainTumor Project/Brain_MRI

# data containing path to Brain MRI and their corresponding mask

brain_df = pd.read_csv('data_mask.csv')

# Display information about the Dataset

brain_df.info()

"""#A: TRAIN A CLASSIFIER MODEL TO DETECT IF TUMOR EXISTS OR NOT (VGG)"""

# Drop the patient id column

brain_df_train = brain_df.drop(columns = ['patient_id'])
brain_df_train.shape

# Convert the data in mask column to string format, to use categorical mode in
flow_from_dataframe

brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))

brain_df_train.info()
```

```
# split the data into train and test data

from sklearn.model_selection import train_test_split

train, test = train_test_split(brain_df_train, test_size = 0.15)

# create a image generator

from keras.preprocessing.image import ImageDataGenerator

# Create a data generator which scales the data from 0 to 1 and makes validation split
of 0.15

datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.15)

# Create a data generator for train images

train_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="training",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))


valid_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="validation",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))

# Create a data generator for test images

test_datagen=ImageDataGenerator(rescale=1./255.)

test_generator=test_datagen.flow_from_dataframe(
dataframe=test,
directory= './',
x_col='image_path',
y_col='mask',
batch_size=16,
shuffle=False,
class_mode='binary',
target_size=(256,256))

# Load the VGG-16 model
```

```
input_tensor = Input(shape=(256, 256, 3))  # The input shape (256, 256, 3) indicates
that model expects input images with Width, height: 256 pixels and Color Channels: 3.

basemodel = VGG16(weights='imagenet', include_top=False, input_tensor=input_tensor)

for layer in basemodel.layers:
   layer.trainable = False

# Adding classification head
x = AveragePooling2D(pool_size=(4, 4))(basemodel.output)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)  # Adjust the dropout rate as needed
x = Dense(2, activation='sigmoid')(x)  # have 2 classes for binary classification

# Create the VGG-16-based model
model = Model(inputs=basemodel.input, outputs=x)

# Compile the model
model.compile(
   optimizer=Adam(learning_rate=0.001),  # Adjust the learning rate
   loss='categorical_crossentropy',  # Use categorical cross-entropy for binary
classification
   metrics=['accuracy']
)

# use early stopping to exit training if validation loss is not decreasing even after
certain epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)

# Training the model

history = model.fit(train_generator, steps_per_epoch= train_generator.n // 16, epochs =
10, validation_data= valid_generator, validation_steps= valid_generator.n // 16,
callbacks=[earlystopping])

# Interactive graph of the Training and Validation Loss

import plotly.graph_objects as go

# Access training loss and validation loss from the history
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_loss) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training loss trace
fig.add_trace(go.Scatter(x=epochs, y=train_loss, mode='lines+markers', name='Training
Loss'))

# Add validation loss trace
```

```
fig.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines+markers', name='Validation
Loss'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Loss',
   xaxis_title='Epochs',
   yaxis_title='Loss',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)

# Show the interactive plot
fig.show()

# Interactive graph of the Training and Validation Accuracy

# Access training accuracy and validation accuracy from the history
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Create a range of epochs (e.g., [1, 2, 3, ...])
epochs = list(range(1, len(train_accuracy) + 1))  # Convert range to a list

# Create a Plotly figure
fig = go.Figure()

# Add training accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=train_accuracy, mode='lines+markers',
name='Training Accuracy'))

# Add validation accuracy trace
fig.add_trace(go.Scatter(x=epochs, y=val_accuracy, mode='lines+markers',
name='Validation Accuracy'))

# Set layout options and adjust figure size
fig.update_layout(
   title='Training and Validation Accuracy',
   xaxis_title='Epochs',
   yaxis_title='Accuracy',
   hovermode='closest',  # Enables interactive hovering
   width=800,  # Adjust the width in pixels
   height=400  # Adjust the height in pixels
)

# Show the interactive plot
fig.show()

"""#B: ASSESS TRAINED MODEL PERFORMANCE (VGG)"""

# make prediction

test_predict = model.predict(test_generator, steps = test_generator.n // 16, verbose =1)

test_predict.shape
```

```
# Obtain the predicted class from the model prediction

predict = []

for i in test_predict:
 predict.append(str(np.argmax(i)))

predict = np.asarray(predict)

predict

# since we have used test generator, it limited the images to len(predict), due to batch
size

original = np.asarray(test['mask'])[:len(predict)]
len(original)

# Obtain the accuracy of the model

from sklearn.metrics import accuracy_score

accuracy = accuracy_score(original, predict)

print("Model Accuracy: ", accuracy)

# Classification Report

from sklearn.metrics import classification_report

report = classification_report(original, predict, labels = [0,1])
print(report)

from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
cm = confusion_matrix(original, predict)

# Define class labels
class_labels = ["(0, 'No')", "(1, 'Yes')"]

# Create a figure and axis for the confusion matrix
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)

# Add labels and title
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

plt.show()



import matplotlib.pyplot as plt
```

```
import numpy as np

# Results for VGG
vgg_paper_accuracy = [0.96, 0, 0.98, 0.89, 0]  # Replace with the actual values from the
papers
vgg_paper_f1 = [0.97, 0, 0.99, 0.87, 0]  # Replace with the actual values from the
papers
your_vgg_accuracy = 0.88  # Replace with the actual value from your project
your_vgg_f1 = 0.88  # Replace with the actual value from your project

# Results for ResNet
resnet_paper_accuracy = [0.89, 0.98, 0.99, 0, 0.86]  # Replace with the actual values
from the papers
resnet_paper_f1 = [0.90, 0.96, 0.99, 0, 0]  # Replace with the actual values from the
papers
your_resnet_accuracy = 0.98  # Replace with the actual value from your project
your_resnet_f1 = 0.99  # Replace with the actual value from your project

models = ['Haitham [1]', 'Anjaneya [2]', 'Mahcene [3]', 'Nazreth [4]', 'Lamia [5]', 'My
Project']

# Bar chart for VGG
plt.figure(figsize=(8, 4))
bar_width = 0.2
index = np.arange(len(models))

plt.bar(index, vgg_paper_accuracy + [your_vgg_accuracy], width=bar_width,
label='Accuracy')
plt.bar(index + bar_width, vgg_paper_f1 + [your_vgg_f1], width=bar_width,
label='F1-Score', alpha=0.9)

plt.xlabel('Papers')
plt.ylabel('Percentage')
plt.title('VGG Performance Comparison')
plt.xticks(index + bar_width / 2, models)
plt.legend(loc='lower right', fontsize='small')
plt.show()

# Bar chart for ResNet
plt.figure(figsize=(8, 4))
bar_width = 0.2
index = np.arange(len(models))

plt.bar(index, resnet_paper_accuracy + [your_resnet_accuracy], width=bar_width,
label='Accuracy')
plt.bar(index + bar_width, resnet_paper_f1 + [your_resnet_f1], width=bar_width,
label='F1-Score', alpha=0.9)

plt.xlabel('Papers')
plt.ylabel('Percentage')
plt.title('ResNet Performance Comparison')
plt.xticks(index + bar_width / 2, models)
plt.legend(loc='lower left', fontsize='small')
plt.show()
```