5-2024

AUTOMATED BRAIN TUMOR CLASSIFIER WITH DEEP LEARNING

venkata sai krishna chaitanya kandula
California State University – San Bernardino

AUTOMATED BRAIN TUMOR CLASSIFIER WITH DEEP LEARNING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Venkata Sai Krishna Chaitanya, Kandula

May 2024

AUTOMATED BRAIN TUMOR CLASSIFIER WITH DEEP LEARNING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Venkata Sai Krishna Chaitanya, Kandula

May 2024

Approved by:

Dr. Yan Zhang, Advisor, School of Computer Science

Dr. Jennifer Jin, Committee Member

Dr. Qingquan Sun, Committee Member

ABSTRACT

Brain Tumors are abnormal growth of cells within the brain that can be categorized as benign (non-cancerous) or malignant (cancerous). Accurate and timely classification of brain tumors is crucial for effective treatment planning and patient care. Medical imaging techniques like Magnetic Resonance Imaging (MRI) provide detailed visualizations of brain structures, aiding in diagnosis and tumor classification[8].

In this project, we propose a brain tumor classifier applying deep learning methodologies to automatically classify brain tumor images without any manual intervention. The classifier uses deep learning architectures to extract and classify brain MRI images. Specifically, a Convolutional Neural Network (CNN) is trained on a diverse dataset of brain tumor images. The CNN learns intricate patterns and features within the images, enabling it to classify various tumor types. Transfer learning, utilizing pre-trained models such as Visual Geometry Group (VGG) and EfficientNet (B3), enhances the CNN model's ability to generalize across different datasets. Additionally, a traditional neural network Multi-Layer Perceptron (MLP) is applied to classify brain MRI images.

The performance of the VGG, EfficientNet, and MLP models are evaluated and compared. The metrics of accuracy, precision, recall, and F1 score are used to evaluate the efficacy of each model in brain tumor classification. This project contributes to the advancement of automated brain tumor diagnosis, potentially improving patient outcomes through more efficient diagnosis strategies.

ACKNOWLEDGEMENTS

I’m writing this to express my gratitude to Professor Dr. Yan Zhang for serving as an advisor for this project. I’m thankful to her for recommending the High-performance computing platforms to implement the idea.

Professor Dr. Zhang gave more insights into machine learning concepts for implementation, provided constructive feedback, and directed the project based on progress. I’m fortunate to have the opportunity to work under the mentorship of Dr. Zhang. I’d like to extend my thankfulness to Dr. Jennifer Jin and Dr. Qingquan, Sun for their belief in my abilities to handle the project.

My sincere thanks to Professor Dr. Haiyan, Qiao, and Dr. Zhang who provided an opportunity to tutor the machine learning course. This experience helped me to excel in handling machine learning models w.r.t problem statements.

I’d also like to thank the faculty of the computer science department, at California State University, San Bernardino who taught me courses that are the foundation stones for the project.

DEDICATION

With respect and gratitude, I’m dedicating my master’s project to Professor Dr. Yan Zhang. Without her encouragement, I wouldn’t have been able to implement any project in machine learning.

I thank my son (Sid Naag, Kandula) who waited for me at home during my master’s degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Background

Brain Tumor disease is one of the most dangerous cancers in adults and children. Effective treatment requires early detection and classification of brain tumors. The diagnosis of brain tumors is based on the image data analysis of brain images obtained by magnetic resonance imaging (MRI) [6]. The first step in determining the status of a patient with a brain tumor is the accurate interpretation of the brain tumor pictures.

Motivation

Detection of brain tumors is difficult, and the situation becomes even more difficult when there is no automated detection process. When tumor cells are forming in the human brain there is a higher likelihood of significant mortality. Brain tumors are unstable for about twenty-five days because of the complexity of the tissues. The person’s survival rate is usually less than 12 months [12]. More accurate computer-based and automated tumor detection/diagnosis methods are needed to understand and intervene in this real situation. Recently, several attempts have been made to explore machine-learning techniques to automate this process. Magnetic resonance imaging (MRI) detects abnormal tissues that need to be treated. Because of the complexity and variety of tumor types, diagnosing a brain tumor is difficult [6]. Collecting, organizing, and analyzing images have become standard procedures that enable data-driven techniques to detect brain tumors. With the latest machine learning techniques, the classifier uses a deep learning architecture for feature extraction and classification [6].

Contribution

The project aims to classify the user’s MRI brain image (which type of tumor is present). Besides, the main agenda of our project focuses on demonstrating the strengths and weaknesses of the deep learning models used to classify brain tumors. The proposed Automated Brain Tumor Classifier in this project is an advanced system designed to classify brain tumor images with an MRI uploaded by the user. Selecting the best among the four models based on their performance metrics is also one of the project outcomes.

This project utilizes deep learning models to analyze the input MRI image using different techniques for feature extractions. VGG model is selected for spatial exploitation while EfficientNet is for multi-path compound scaling. MLP doesn’t hold feature extraction of the image. However, the results of MLP are compared with VGG and EfficientNet. VGG-16 and VGG-19 are both convolutional neural network (CNN) architectures known for their simplicity and effectiveness in image classification tasks. They consist of a series of convolutional layers followed by max-pooling layers, with increasingly complex features extracted at deeper layers. EfficientNet, on the other hand, is a neural network architecture designed to achieve better accuracy and efficiency by scaling the model's depth, width, and resolution in a balanced way. It uses compound scaling to improve performance across different resource constraints. Multilayer Perceptron (MLP) is a type of feedforward neural network where information moves in only one direction, from input nodes through hidden layers to output nodes. It's often used in classification tasks, but its effectiveness in image classification might be limited compared to CNNs due to the absence of convolutional and pooling layers for feature extraction. When classifying brain tumors, VGG-16 and VGG-19 would be robust choices given their proven track record in image classification tasks, while EfficientNet could offer a good balance between accuracy and computational efficiency. However, MLP might not be as effective in capturing the intricate features present in medical images like brain scans.

In evaluating and comparing the performance of VGG-16, VGG-19, EfficientNet, and MLP for classifying brain tumors, several factors need consideration. VGG-16 and VGG-19, with their deep convolutional architectures [1], offer strong performance in image classification tasks due to their ability to extract intricate features from images. However, they may suffer from computational inefficiency and high memory requirements. EfficientNet addresses these issues by achieving better accuracy with fewer parameters [6] through compound scaling, making it a compelling choice for resource-constrained environments. On the other hand, while MLP is simple and easy to implement, its performance might lag CNN architectures like VGG-16, VGG-19, and EfficientNet in image classification tasks, particularly when dealing with complex medical images such as brain scans, where capturing fine-grained features is crucial. Therefore, while VGG-16, VGG-19, and EfficientNet offer promising results, MLP may not be as suitable for robust brain tumor classification.

Project Milestones

Implementing this project included reaching milestones like collecting the images, preprocessing them, model implementation, and selecting the apt model concerning the performance metrics.

Data Collection and Preprocessing:

- Define the data requirements: Determine the types of brain tumor images needed for classification.
- Gather data: Collect a representative dataset of brain tumor images.
- Clean the data: Process the dataset by addressing missing values removing information and ensuring consistency.
- Augment the data: Enhance the dataset using techniques, like rotation, flipping, and scaling to improve how well the model generalizes.
- Normalize: Adjust values to ensure uniformity across all images.

Model Implementation:

- Choose architecture: Select and implement models like VGG-16, VGG-19, EfficientNet and MLP for classifying brain tumors.
- Train models: Use the dataset to train each model tweaking hyperparameters as necessary.
- Evaluate models: Assess performance using validation data and metrics such as accuracy, precision, recall, and F1 score.
- Fine-tune hyperparameters: Adjust model settings for performance if needed. Explore combining predictions from models to enhance accuracy.

Findings and Insights:

- Performance: Analyze how VGG-16, VGG-19, EfficientNet, and MLP models perform in classifying brain tumors.
- Identify strengths and weaknesses: Evaluate each model's advantages and limitations within the context of brain tumor classification.
- Selecting the Best Model: Choose the model for the task based on its performance metrics.
- Analyzing Feature Importance (for MLP): If you are using MLP, examine which characteristics have a contribution to the classification by assessing their importance.

By following these milestones, the project aims to progress from collecting and preprocessing data to implementing models analyzing findings, and presenting data visualizations that effectively communicate the results.

CHAPTER TWO

DATASET COLLECTION AND PREPROCESSING

Dataset Information

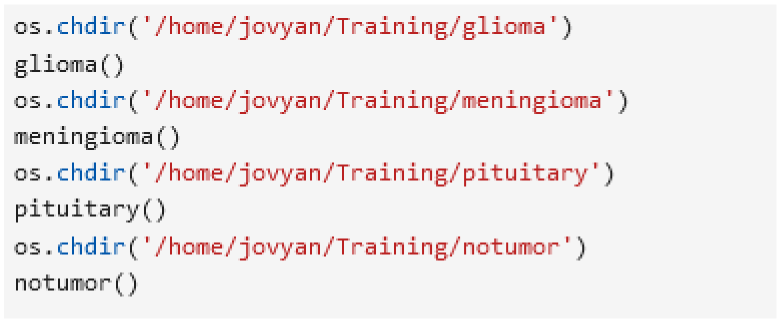
The dataset used in this project is published in Kaggle [11]. This dataset contains 7023 images of human brain MRI images which are classified into 4 classes: glioma, meningioma, pituitary, and healthy brain. Inputs are provided in the form of images. Output is expected to be a multi-class label. This means that output can fall into one of the four categories. The dataset is balanced with inputs in all kinds of categories. We considered 5140 images for training, 1311 for testing, and 572 for validation. Further inside the dataset, to prove that this is a balanced dataset Table 1 supports the assumption.

Table 1: Dataset Segregation

Category	**Training**	**Testing**
Glioma	1321	300
Meningioma	1339	306
Pituitary	1457	405
No Tumor	1595	300

Data Importing and Preprocessing

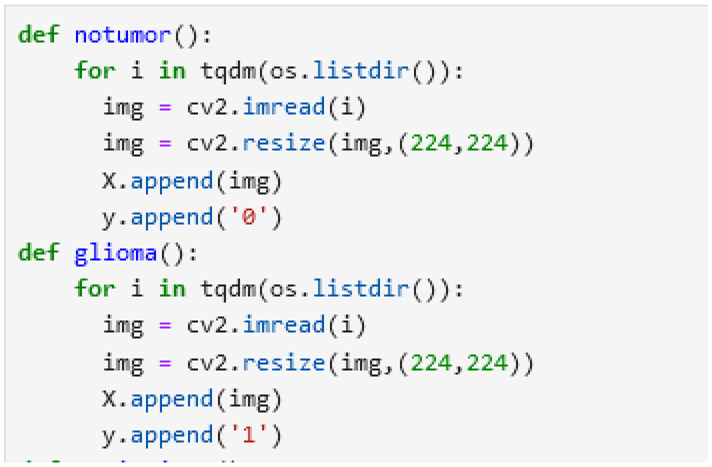
Data importing in this project refers to the reading of the input image as a file from a directory and labeling as per folder structure. Figure 1 provides a piece of code on how to read the images from local directories via Python.



```
os.chdir('/home/jovyan/Training/glioma')
glioma()
os.chdir('/home/jovyan/Training/meningioma')
meningioma()
os.chdir('/home/jovyan/Training/pituitary')
pituitary()
os.chdir('/home/jovyan/Training/notumor')
notumor()
```

Figure 1. Importing of Images

Upon importing the following methods will start labeling each image. For classification at the model level, we represented classes as No Tumor (0), Glioma(1), Meningioma(2), and pituitary(3). Figure 2 provides the methods that help to resize each image and label them with respective numbers as below:



```
def notumor():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X.append(img)
      y.append('0')
def glioma():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X.append(img)
      y.append('1')
```

Figure 2. Labeling of Images

The cv2.imread function is used to read an image from a file. It loads an image from the specified file path and returns it as a NumPy array. The idea is to load the images initially and do preprocessing. After the image is read, we apply 224*224 dimension preprocessing. After resizing the image, the flatten method is applied for the Multi-Layer Perceptron model explicitly. This method converts the 2D array in the resized image to a 1D array.

The flatten method is not required for VGG and EfficientNet as these deep learning models can handle a two-dimensional array. The image data between these pixels will be considered as an input feature. Output/predictor is dependent on the image data fed in the NumPy array. For a model to train, we need the training data which in this case the image files of all kinds of tumors are read and preprocessed.

In the same way, to predict the test images, we need to preprocess before sending the image to the model. Preprocessing w.r.t text data will have formatting, cleaning, removing undefined/null values, etc. For example: we will not consider unique IDs as input features (say medical ID). This unique input feature will not be able to predict the output. So, cleaning these kinds of fields is required beforehand.

Tools

To implement this project, we used libraries, modules, and software. Starting from Visual Studio code, by using Python programming language in VS code with the libraries sklearn and keras the data has been extracted and processed. With the use of libraries – TensorFlow images have been classified accordingly. Further, we used VS code for UI development using streamlit libraries. Integration with the back end is done via VS code.

Jupyter Notebook:

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text [15]. It is widely used in data science, machine learning, scientific research, and education. Jupyter Notebooks support multiple programming languages, but it is particularly popular in the Python community. Users can write and execute code interactively in a step-by-step fashion, making it great for exploratory data analysis and prototyping [15]. Notebooks can be easily shared with others, facilitating collaboration. They can be exported in various formats, including HTML, PDF, and slideshows, making them convenient for presentations and documentation. In this project, we used the 6.4.5 version of Jupyter Notebook on CSUSB high performance computers to achieve.

Python:

Python is a programming language that is widely used for tasks such as data analysis, machine learning, web development, and automation. It has a user readable syntax [4]. This project specifically utilizes Python version 3.10.9. One of the advantages of Python is its absence of braces and the use of indentation for code organization, which enhances readability and reduces errors. Additionally, Python offers a collection of libraries and frameworks to support functionalities. Another notable feature is that Python does not require compilation before execution since it is an interpreted language [4]. This flexibility enables developers to easily develop and troubleshoot their code. Moreover, Python being an open-source language ensures accessibility for everyone to use and contributes towards its advancement. Furthermore, Python comes with a library that encompasses a wide range of functions and modules catering to various tasks minimizing the reliance on external dependencies [4].

VS Code:

One of the most popular pieces of software development is Visual Studio Code, or VS Code, an open-source code editor with a reputation for being extensible, adaptable, and developer-friendly. The version used in this project is

1.85.1. Visual Studio Code is a cross-platform software that works with Windows, Linux, and Mac OS. It is compatible with an enormous community-contributed extension library. By providing developers with enormous options and extensions, it supports a wide range of programming languages and file types. Developers no longer have to switch between an editor and a terminal to execute command line tools and scripts within Visual Studio code. The setup is mentioned in user docs section "https://code.visualstudio.com/docs"

Streamlit:

Streamlit is a Python library that makes it a breeze to create web applications, for data science and machine learning [18]. It simplifies the process of transforming your data scripts into web apps. What's great about Streamlit is that it prioritizes simplicity, meaning you can create web apps with a few lines of Python code. But don't let its simplicity fool you – Streamlit also offers customization options. You have control over the layout, appearance, and behavior of your app by tweaking parameters and using custom CSS styles. One of the features of Streamlit is its ability to deploy and showcase machine learning models seamlessly. It seamlessly integrates with popular machine learning libraries, like TensorFlow and PyTorch allowing you to incorporate your models into your apps effortlessly [18]. When it comes to deploying your Streamlit apps you have plenty of options at your disposal. Whether you choose Streamlit Sharing, Heroku, AWS, or other platforms you'll find that the process is straightforward and hassle-free. Streamlit Sharing deserves a mention as it provides a hosting solution that won't cost you anything.

Keras:

Keras, a Python-based open-source application programming interface (API) is widely used for constructing and training learning models [7]. Initially designed to be a user interface built on top of deep learning frameworks. Keras has now evolved into an essential component of TensorFlow since the release of TensorFlow 2.0. It serves as the high-level API, for TensorFlow [7]. Preprocessing the images is done using Keras. Keras provides pre-defined models for common tasks, such as image classification (e.g., VGG-16, ResNet), text generation (e.g., LSTM), and more [7]. These models can be easily utilized or adapted for specific purposes.

TensorFlow:

TensorFlow, a machine learning framework developed by the Google Brain team is highly popular, for constructing and training machine learning models involving learning [1]. It offers a range of tools, libraries, and community resources to support the development of machine learning applications [1]. This specific project utilizes version 2.9.1 of TensorFlow. Notably TensorFlow 2.0 and subsequent versions have integrated Keras as their high-level API, which greatly enhances TensorFlow ease of use and provides users with an interface for creating and training neural networks. Moreover, TensorFlow supports programming languages such as Python, C++, and Java; however, its primary

API remains in Python due to its accessibility and widespread adoption, within the machine learning community [1].

NumPy:

NumPy, also known as Numerical Python is an open-source Python library that plays a role, in numerical computation [13]. It offers support for handling arrays and matrices with dimensions accompanied by a range of mathematical functions to perform operations on these arrays [13]. NumPy serves as the foundation for Python libraries used in scientific computing enabling efficient and speedy numerical calculations. Its notable features encompass robust array manipulation capabilities, and functions, the ability to broadcast array operations, and seamless integration with other tools used in data science and machine learning. NumPy finds application in tasks such as linear algebra, statistical analysis, signal processing, and more [13]. Due to its indispensability in computing and data manipulation, within the Python ecosystem, it has become a tool.

CHAPTER THREE

METHODOLOGIES

Deep learning is a branch of machine learning that emulates the workings of the brain [9]. It relies on networks, which are designed to process data and learn from it to recognize patterns and make predictions, for new data sets. Deep learning models have applications with one prominent use being, in solving classification problems.

Conventional Neural Networks

A Convolutional Neural Network (CNN) also known as a ConvNet is a type of network architecture specifically designed for tasks related to image recognition and computer vision [9]. It incorporates convolutional and pooling layers to process data. The key components of a Convolutional Neural Network are as follows:

Convolutional Layers

These layers serve as the building blocks of CNNs. They apply convolution operations to input images using filters called kernels. By performing convolutions, the network can capture patterns and features in the images [5].

Activation Function

Typically Rectified Linear Unit (ReLU) activation functions are utilized after layers. ReLU introduces nonlinearity allowing the network to learn relationships, between features [5].

Pooling Layers

Pooling layers often implemented as max pooling play a role in downsampling the dimensions of feature maps. Pooling aids in reducing load and parameter count while preserving information.

Connected Layers

Connected layers appear at the end of the network, for classification purposes. These layers take high-level features extracted by convolutional and pooling layers. Map them to class scores.

Flattening

Before feeding the results of pooling layers into connected layers it's common practice to convert the data into a one-dimensional vector.

Dropout

To prevent overfitting, during training dropout is a technique that randomly disconnects a fraction of connections.

SoftMax Activation

When dealing with class classification tasks the SoftMax activation function is frequently employed in the output layer. Its purpose is to transform the network scores into probability distributions across classes.

Convolutional Neural Networks (CNNs) are highly effective in tasks such as image classification, object detection, and image segmentation [9]. This is due to their ability to autonomously learn features from input data. CNNs has gained usage. Have demonstrated state-of-the-art performance, in various computer vision challenges [9]. Figure 3 provides the simplified diagram of CNN which includes from input to output layer. Convolutional layers are where features of input images are extracted. Furthermore, the Pooling layer is responsible for reducing the spatial dimensions of feature maps and reducing the computational complexity.

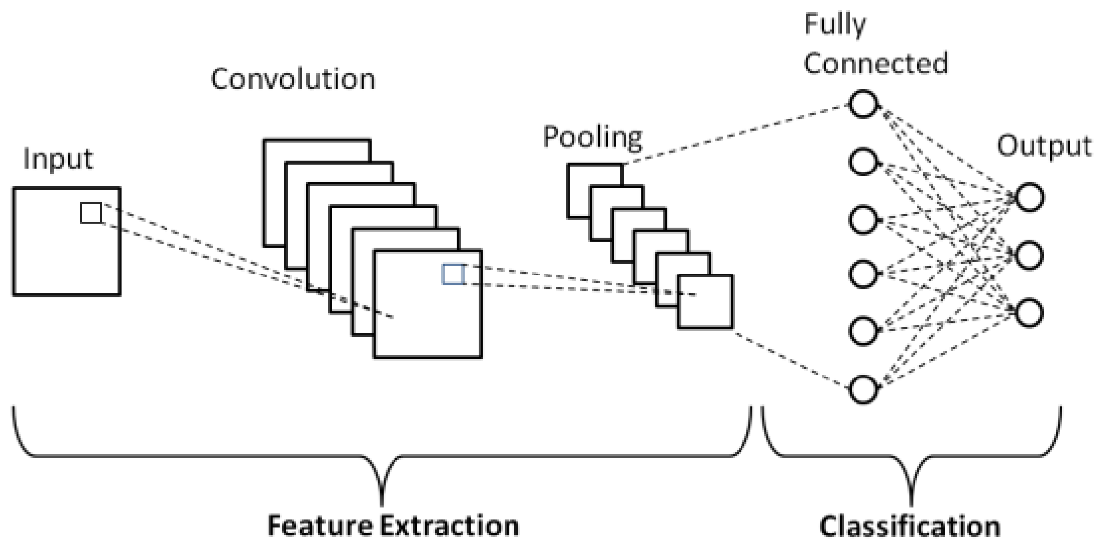


Figure 3. Schematic Diagram of CNN

Figure 4 and Figure 5 illustrate a sample image classification using CNN. D. Bhatt explains the CNN architecture and works on a sample image [12]. Considering the outputs and features, we have chosen CNN architecture to implement the MRI classification problem. Feature extraction function starts from the input layer till flattening the image. Classification is dependent on a fully connected layer. This layer contains a traditional neural network connecting neurons from the previous layer to the next layer. Combining the convolutional and pooling layers will help for prediction [12]. Zebra and computer images are considered in the below examples to classify.

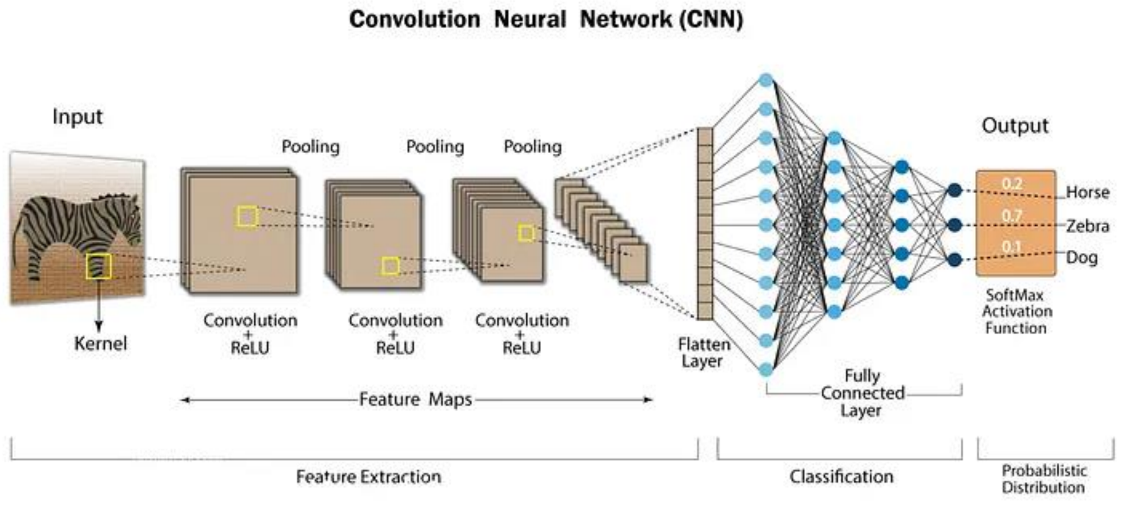


Figure 4. Sample Image Classification using CNN [12]

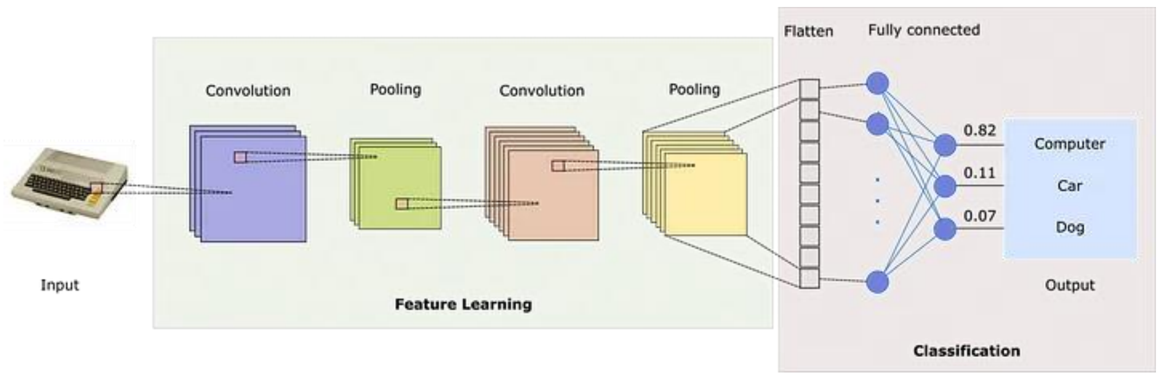


Figure 5. Feature Learning and Classification using CNN [12]

Visual Geometry Group (VGG-16)

The VGG-16 model is a type of deep Convolutional Neural Network (CNN) architecture developed by the Visual Graphics Group (VGG) at the University of Oxford [12]. It belongs to the VGG family of models and is well known for its consistent structure. VGG-16 has gained popularity, for its performance in tasks related to image classification [9].

Architecture

VGG-16 comprises a total of 16 layers, including both fully connected layers. The architecture follows a pattern consisting of repeated layers with small 3x3 filters, followed by max pooling layers. Three connected layers are placed after the layers [19].

Filter Size and Stride

In VGG-16 all convolutional layers utilize filters with dimensions of 3x3. The stride (the amount by which the filter moves) for these layers is set to 1 pixel.

Max Pooling

After each set of layers, max pooling is applied. Max pooling involves using filters sized at 2x2 and moving them with a stride of 2 pixels.

Connected Layers

The final stages of VGG-16 consist entirely of layers that possess respective neuron counts of 4096 and finally,1000. The last layer containing precisely 1000 neurons corresponding to the output classes within the ImageNet dataset was initially trained with VGG-16 [19].

<u>Activation Function</u>

Throughout the network except, for the output layer. Rectified Linear Unit (ReLU) activation functions are uniformly employed. The final layer of a network, known as the output layer, often utilizes the softmax activation function when dealing with tasks involving class classification.

VGG-16, a network model was initially trained on the ImageNet dataset. This dataset consists of a collection of labeled images, across categories. The pre-trained weights of VGG-16 have proven to be quite useful as a starting point for transfer learning in computer vision tasks. In applications, VGG-16 can be employed for tasks like image classification, object detection, and feature extraction.

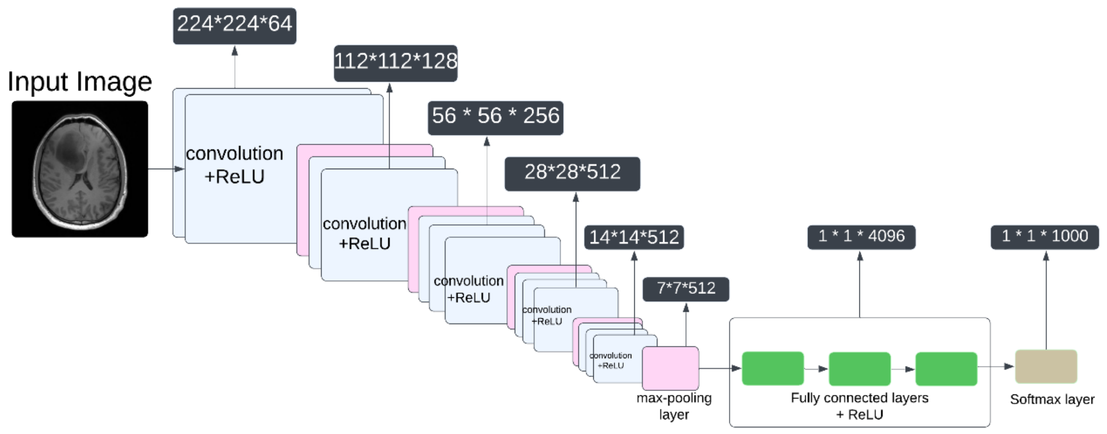


Figure 6. VGG-16 224*224 8 Layer Schematic Diagram

Figure 7 explains the architecture of VGG of 16 layers. Input and output layers remain the same. 4 blocks of convolutional layers in which each block contains 3 layers. CNN architecture in Figure 4 overlaps with Figure 7. Further flavors of VGG increments by 3 in three blocks. Example: VGG-16, VGG-19, VGG21 etc.

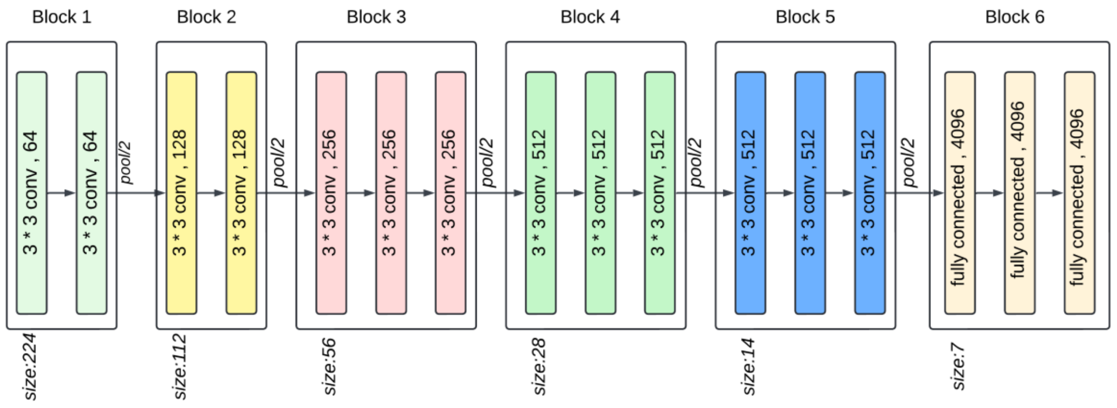
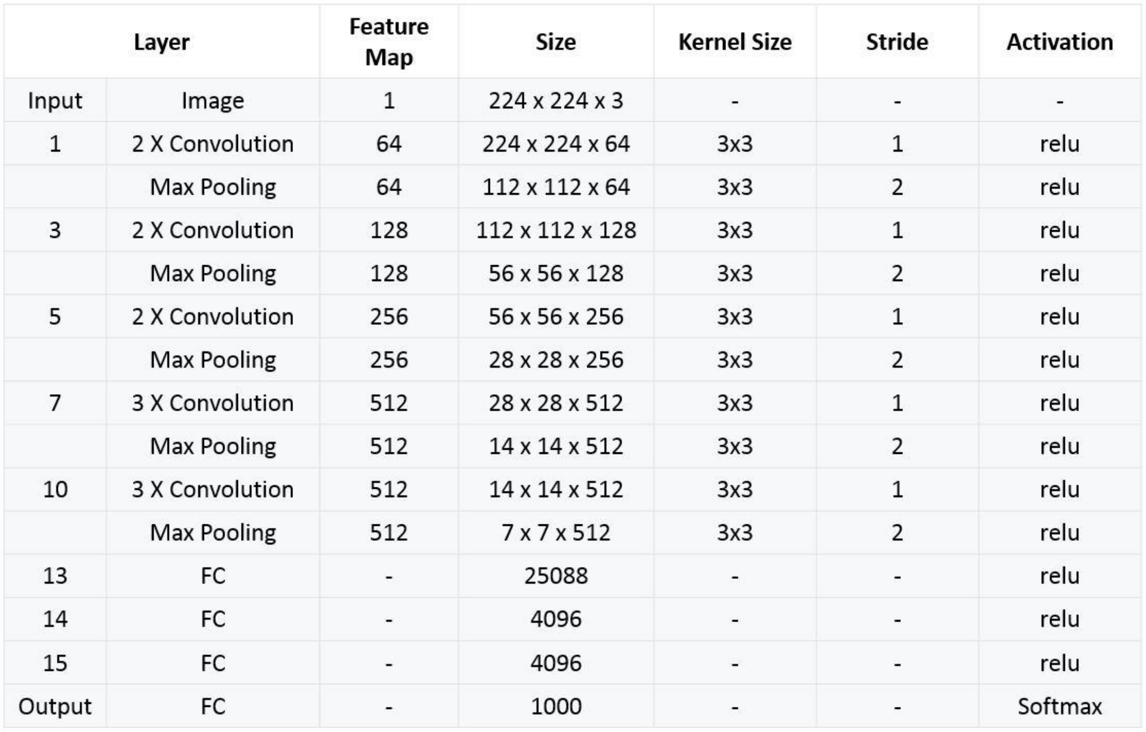


Figure 7. VGG-16 Architecture Diagram

Figure 8 shows a VGG-16 layer with size, kernel, and activation function. A series of VGGs are exactly the same in the last three fully connected layers. The overall structure includes 5 sets of convolutional layers, followed by a MaxPool. The difference is that more cascaded convolutional layers are included in the five sets of convolutional layers.



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Figure 8. VGG-16 Features and Components

Visual Geometry Group (VGG-19)

VGG-19 (Visual Geometry Group - 19 layers) is a deep convolutional neural network architecture that gained prominence for its simplicity and effectiveness in image classification tasks [3]. The model was introduced in the paper titled "Very Deep Convolutional Networks for Large-Scale Image Recognition," authored by Karen Simonyan and Andrew Zisserman.

The VGG-19 model is part of the VGG family, which includes variants with different numbers of layers (VGG-16, VGG-19). The key characteristics of VGG networks are their uniform architecture with small 3x3 convolutional filters and the stacking of multiple convolutional layers. The authors explored the impact of increasing network depth on image classification performance, demonstrating that deeper models tend to perform better on large-scale visual recognition tasks.

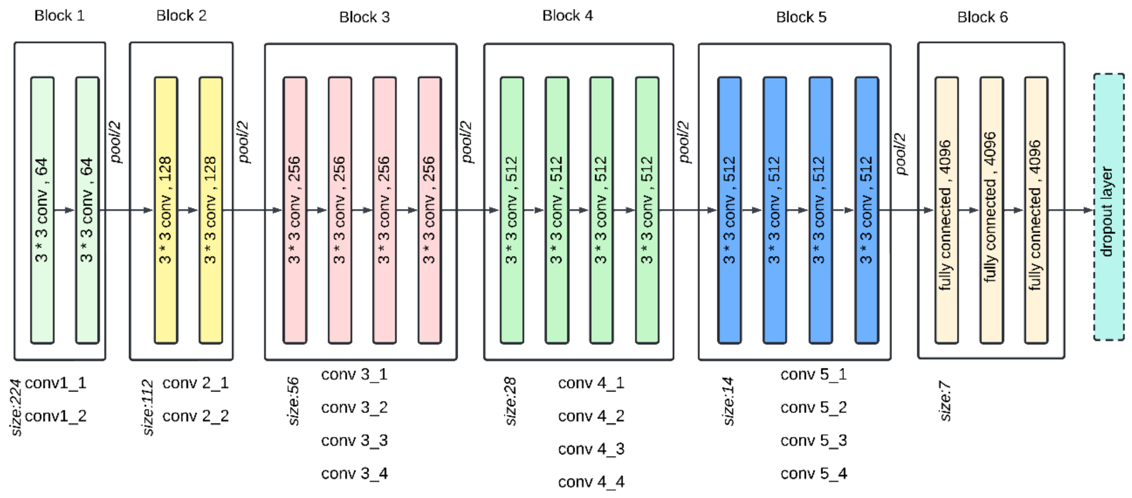


Figure 9. VGG-19 Architecture

Figure 9 illustrates the architecture of VGG-19. Feature extraction functionality consists of 16 convolutional layers. This is divided into 5 blocks. The first two blocks are identical compared to VGG-16. 3 more layers are added from block 3 to block 5. During the training process, our ConvNets receive a fixed size 224 × 224 RGB image as input [21]. We simply adjust the RGB value calculated from the training set by subtracting it from each pixel. The image then goes through a series of convolutional layers that utilize filters with a receptive field size of 3 × 3 to capture left/right, up/down, and center concepts. Additionally, we incorporate 1 × 1 convolution filters, in one configuration to perform transformations on input channels followed by non-linearity. The convolution stride remains at 1 pixel and spatial padding ensures that the spatial resolution is maintained post convolution with a padding of 1 pixel, for every 3 × 3 conv Layer. Spatial pooling involves five max pooling layers following some conv. Layers (not all) where max pooling is done over a window of size 2 × 2 pixels with a stride of 2.

A series of layers precedes three Connected (FC) layers, in various architectures; the initial two consist of 4096 channels each while the third is responsible for conducting 1000-way ImageNet Large Scale Visual Recognition Challenge (ILSVRC) classification accommodating 1000 channels corresponding to each class [9]. The ultimate layer is the max layer. The setup of the connected layers remains consistent, across all networks.

For the experimental purpose, we have implemented it in a sequential manner of layers instead of making use of the existing Keras inline function.

EfficientNet (B3)

EfficientNet is a group of convolutional neural network architectures created to strike a balance between accuracy, efficiency, and model size. It aims to optimize resources while maintaining performance levels.

Compound Scaling

EfficientNet introduces a method called compound scaling that uniformly scales the network's width, depth, and resolution. This scaling factor is determined by a user-defined compound coefficient (φ) which considers the resources [10].

Depth-wise Separable Convolutions

Like MobileNet, EfficientNet utilizes depth convolutions to reduce parameters and computational requirements. These convolutions consist of two steps; depth convolution followed by point convolution [10].

Inverted Residuals with Linear Bottlenecks

To effectively capture low-level features the model employs inverted residuals in its architecture. Linear bottlenecks are also introduced to enhance information flow throughout the network.

Efficient Blocks

The fundamental building block of EfficientNet is a block that incorporates wise separable convolutions and linear bottlenecks.

Scaling Resolution

EfficientNet adjusts the input resolution, in conjunction with the network architecture to capture spatial information [5]. This means that using resolution images leads to the extraction of features.

Variations in Model Architecture

EfficientNet offers multiple model variants (B0 to B7) with different scaling coefficients. These variants are designed to accommodate varying resource limitations catering to both devices and larger cloud-based models.

EfficientNet has proven its excellence in image classification tasks by delivering top-notch performance while maintaining efficiency in terms of model size and computational requirements [11]. It has gained adoption across computer vision applications and serves as a benchmark architecture for tasks like image recognition and object detection, through transfer learning and fine-tuning. Figure 10 represents the model scaling, a) baseline network example b) represents the image width scaling, c) depth scaling d) resolution scaling e) compound scaling scales 3 dimensions with a fixed ratio [10]. Figure 11 is an application of scaling w.r.t a macarons image. Baseline model scaling could not produce an outline. Similar is the case with width, height, depth, and resolution. Applying a compound scaling provides a clear picture of macarons. Red color represents the lower depth, yellow with medium, and blue with higher depth in the image. Compound scaling together with height, and width at a fixed ratio is provided.

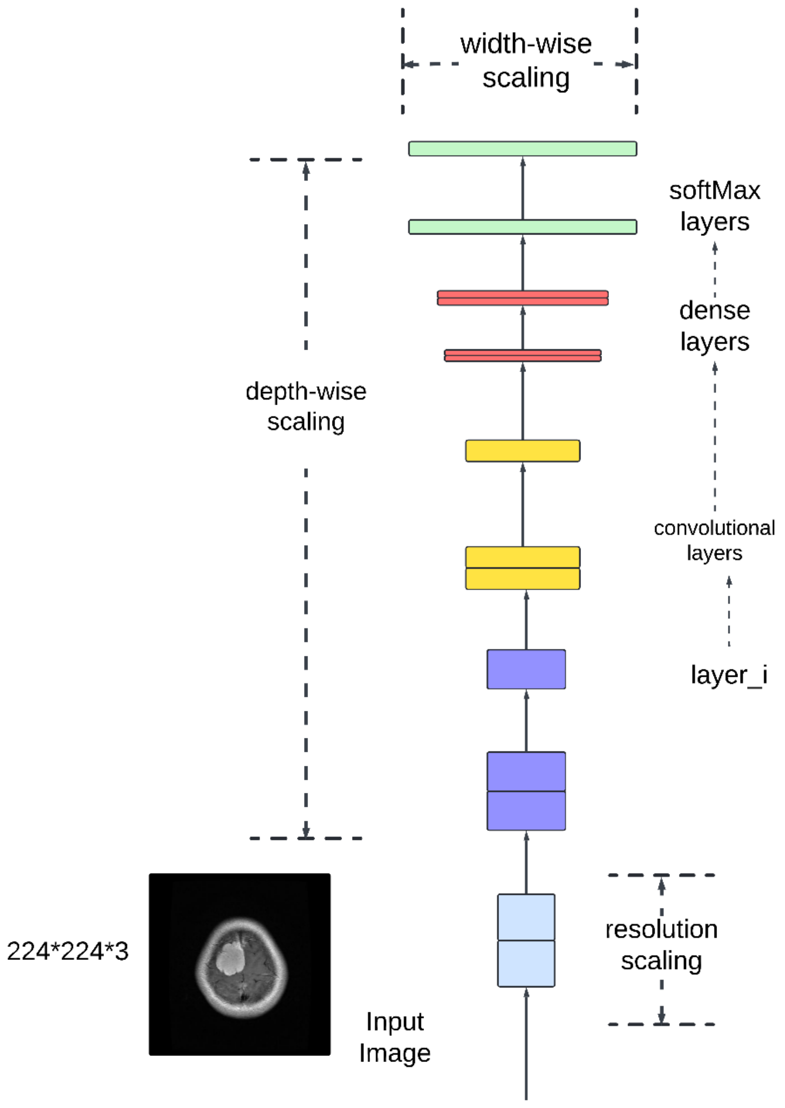


Figure 10. EfficientNet Scaling

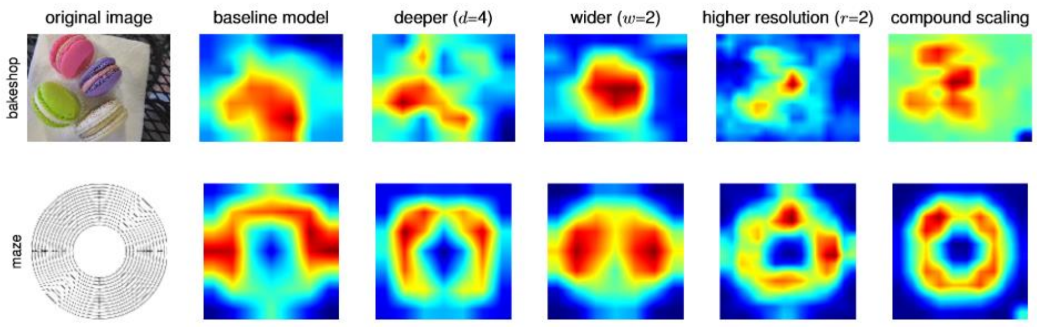


Figure 11. Image Classification with EfficientNet

Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a type of network that consists of several layers of nodes also known as neuro ns or perceptrons [16]. It operates as a feedforward network, where information flows in a direction, from the input layer through hidden layers to the output layer. Important characteristics of an MLP model include [16]:

Input Layer

The input layer represents the features of the input data. Each node in the input layer corresponds to a feature.

Hidden Layers

Between the input and output layers, there can be one or more layers. Each node in a layer applies a sum of inputs followed by an activation function.

Weights and Bias

The connections between nodes in layers are defined by weights. Each node has its bias term. During training these weights and biases are adjusted to minimize errors between predicted and actual outputs.

Activation Function

Nonlinear activation functions are applied to the sum of inputs at each node, in layers. Used activation functions include sigmoid, tangent (tanh), and Rectified Linear Unit (ReLU).

Output Layer

The outcome of the model is generated by the output layer. The number of nodes, in this layer depends on the task, such as classification, multi-class classification, or regression.

To evaluate how well the model performs a loss function is used to measure the difference between the predicted output and the true target values. During training the objective is to minimize this loss. For training Multilayer Perceptron (MLPs) backpropagation and gradient descent are commonly employed. Backpropagation calculates the gradient of the loss with respect to the model's parameters while gradient descent updates these parameters. MLPs are versatile. Have proven successful in tasks like classification, regression, and pattern recognition. They can effectively approximate linear relationships between inputs and outputs. However, they may face challenges with overfitting when dealing with data. Techniques, like dropout and weight decay can be utilized to address this issue. Figure 12 illustrates the architecture diagram of

MLP diagram where hidden layers contain neurons or perceptron. The input block is mapped to every neuron and in the same way it is propagated forward till output.

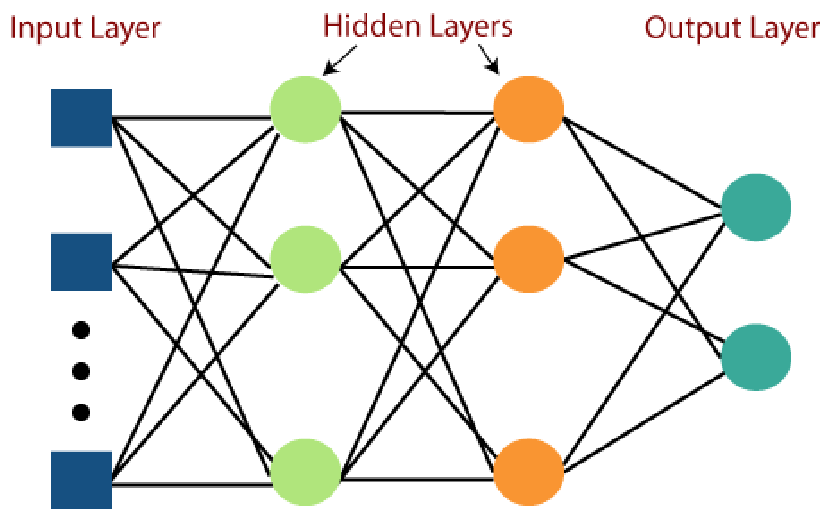


Figure 12. MLP Architecture Diagram

There are reasons why CNNs are preferred over MLPs when working with image data. These reasons should inspire you to delve into the world of CNNs. To use MLPs with images we must flatten the image, which results in losing information or the relationships between neighboring pixels. This loss significantly impacts accuracy. On the other hand, CNNs can preserve information by taking images in their original format. CNNs excel at reducing the number of parameters in a network making them highly efficient in terms of parameters.

CNNs are versatile. Can manage both grayscale and RGB images. In learning, we represent images as arrays of values since an image is composed of pixels. A grayscale image has one color channel and is typically visualized as (height, width, 1) or simply (height, width). We can ignore the dimension since color is always one. Therefore, a grayscale image is a 2D array or tensor.

An RGB image contains three color channels: Red, Green, and Blue. RGB image is represented as (height width, 3) where the third dimension indicates the number of color channels in the image. An RGB image is typically a three-dimensional array or tensor.

CHAPTER FOUR

EVALUATION AND COMPARISON

Metrics and evaluation play a role in assessing the performance and effectiveness of machine learning models. These metrics help us measure how well a model is doing in tasks like classification, regression, clustering, or other types of predictions. The choice of metrics depends on the problem's nature and our goals.

Evaluation Metrics

Evaluation metrics are quantitative measures used to assess the performance of a machine learning model. These metrics provide insights into how well the model is performing in terms of various aspects such as accuracy, precision, recall, F1-score, etc.

Confusion Matrix

In the realm of machine learning a confusion matrix, also referred to as an error matrix, is a table that provides details, on the labels (ground truth) and the predicted class [6]. This matrix not only assesses the model’s performance but also offers a comprehensive analysis of how well it generalizes across different classes [4]. The true labels are represented on the y-axis while the predicted class labels are displayed on the x-axis within the confusion matrix. The confusion matrix outlines the number of correct predictions (TP, TN) as well as incorrect positive and negative predictions (FP, FN) produced by the model.

True Positive (TP): Instances that are actually positive and are correctly classified as positive. True Negative (TN): Instances that are actually negative and are correctly classified as negative. False Positive (FP): Instances that are actually negative but are incorrectly classified as positive (Type I error). False Negative (FN): Instances that are actually positive but are incorrectly classified as negative (Type II error) [6].

TP, TN: Number of Correct Predictions

TP, TN, FP, FN: Total number of predictions.

Classification Report

A classification report is a summary of various performance metrics calculated from the confusion matrix. Parameters to consider for model evaluation are provided below with equations [4]. These metrics collectively provide a comprehensive view of a model's performance on a classification task. When you see a classification report, it helps you understand how well the model is performing for each class and whether it is biased towards a particular class or not. The classification report contains the list of outputs that are classified, accuracy, sensitivity, f1-score, weighted accuracy, and macro accuracy.

Accuracy

The accuracy of the model is defined as the number of correctly predicted outputs out of all ground truths. It gives the percentage of how accurate the proposed model will be on testing [6].

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \qquad (1)$$

Precision

Precision for a particular class c is calculated as the ratio of the number of true positives (correctly predicted instances of class c) to the sum of true positives and false positives (instances incorrectly predicted as class c Precision measures the ratio of predicted observations to all predicted positives [6].

$$Precision = \frac{(TP)}{(TP+FP)} \qquad (2)$$

Recall (Sensitivity or True Positive Rate)

Recall for a particular class c measures the proportion of correctly predicted positive instances of class c among all actual positive instances of class c. It focuses on the ability of the classifier to find all positive instances of class c Recall represents the ratio of predicted observations to all actual positives[6].

$$Recall = \frac{(TP)}{(TP+FN)} \qquad (3)$$

F1 Score

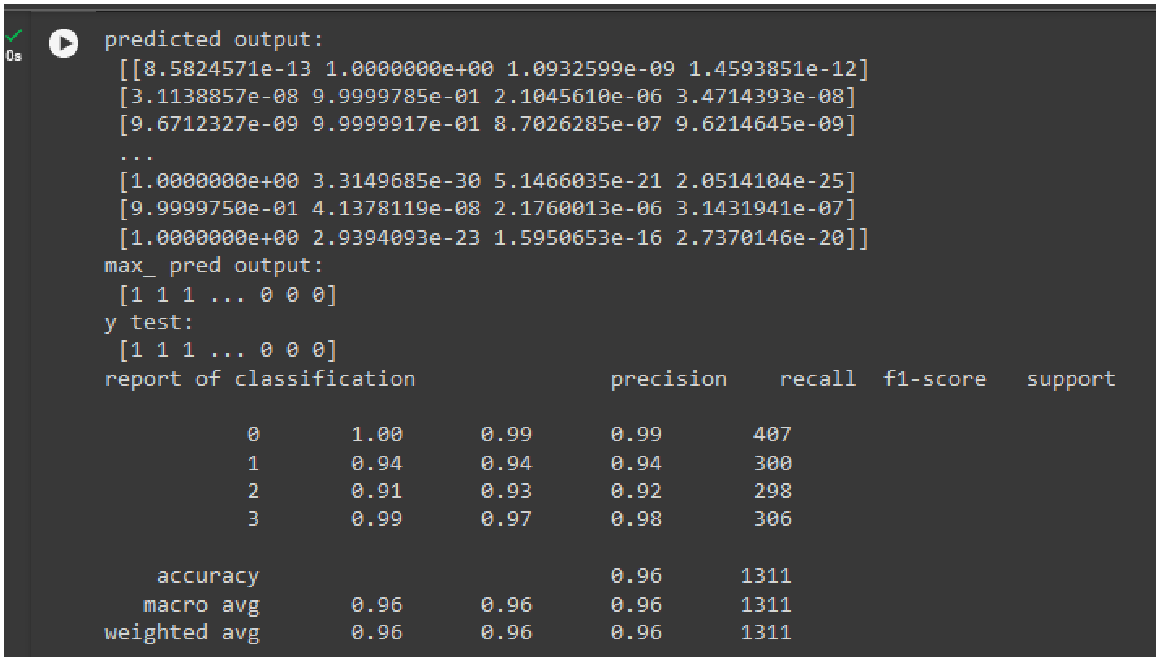
F1-Score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall, making it useful for imbalanced datasets [6].

$$F1\ Score = \frac{(2*Precision*Recall)}{(Precision+Recall)} \qquad (4)$$

Model Evaluation

VGG-16 Evaluation

We will apply each image to the VGG-16 model. The classification report is as below in Figure 13. The classification report contains precision, recall/sensitivity, f1- score, and accuracy. VGG-16 is compared in an experiment done by MDPI [11] “Medical Diagnosis of Brain Tumor with an Effective Hybrid Transfer Learning Model” and performed below AlexNet, and equivalent to Mobile Net V2. From the research article [11] hybrid model GN-Alex Net outperformed with respect to VGG-16.

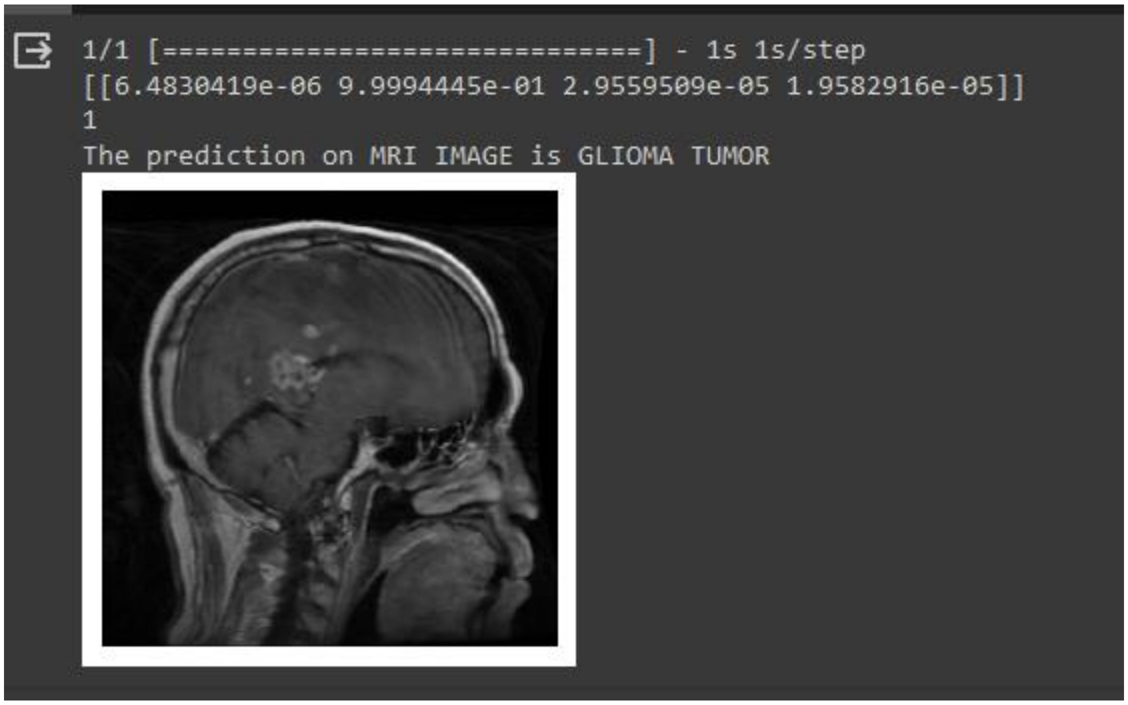


```
predicted output:
 [[8.5824571e-13 1.0000000e+00 1.0932599e-09 1.4593851e-12]
 [3.1138857e-08 9.9999785e-01 2.1045610e-06 3.4714393e-08]
 [9.6712327e-09 9.9999917e-01 8.7026285e-07 9.6214645e-09]
 ...
 [1.0000000e+00 3.3149685e-30 5.1466035e-21 2.0514104e-25]
 [9.9999750e-01 4.1378119e-08 2.1760013e-06 3.1431941e-07]
 [1.0000000e+00 2.9394093e-23 1.5950653e-16 2.7370146e-20]]
max_ pred output:
 [1 1 1 ... 0 0 0]
y test:
 [1 1 1 ... 0 0 0]
report of classification               precision    recall  f1-score   support

           0       1.00      0.99      0.99       407
           1       0.94      0.94      0.94       300
           2       0.91      0.93      0.92       298
           3       0.99      0.97      0.98       306

    accuracy                           0.96      1311
   macro avg       0.96      0.96      0.96      1311
weighted avg       0.96      0.96      0.96      1311
```

Figure 13. VGG-16 Classification Report

An epoch is completed when the model has iterated over the entire training dataset once. Using 20 epochs on training dataset to predict on test data. The number of epochs is a hyperparameter that needs to be set before training the model. The appropriate number of epochs can depend on factors such as the complexity of the task, the size of the dataset, and the convergence behavior of the model.

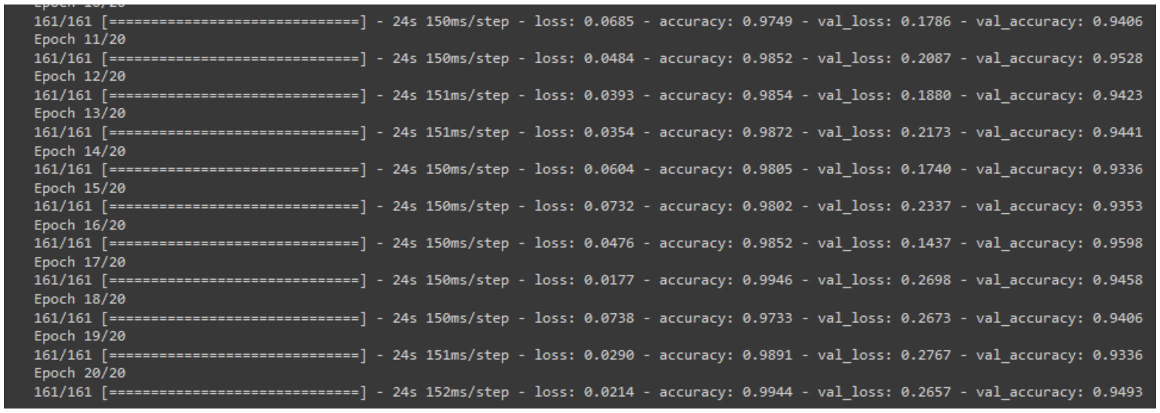
The accuracy obtained by VGG-16 is 96% on test data. Figure 14 represents predicting a single image after fitting and compiling the model on training data.



```
1/1 [==============================] - 1s 1s/step
[[6.4830419e-06 9.9994445e-01 2.9559509e-05 1.9582916e-05]]
1
The prediction on MRI IMAGE is GLIOMA TUMOR
```

Figure 14. Test Image Classification as Glioma

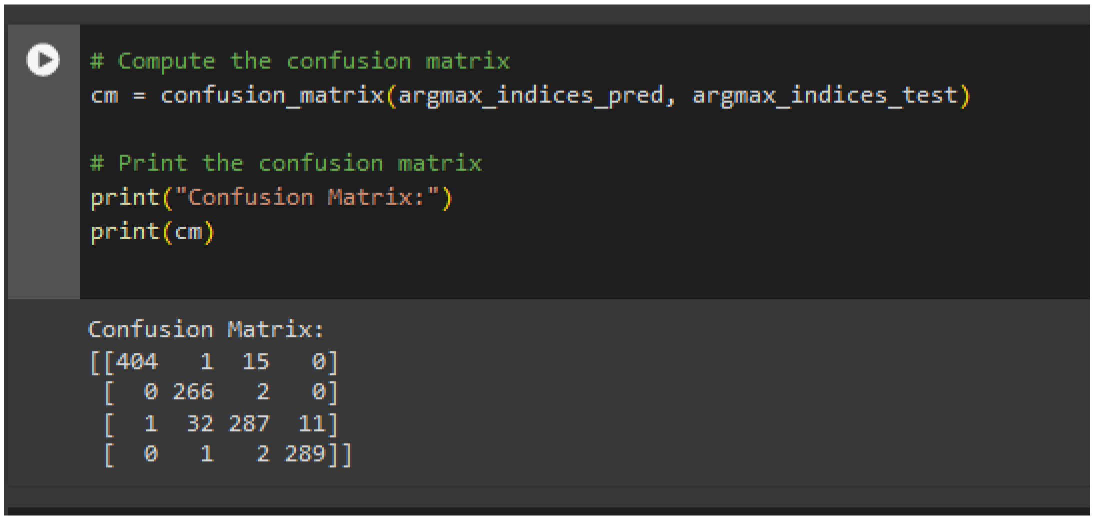
Continuing with the computation time of model compilation with each epoch is 150ms/step approx. and 24 sec in Figure 15. Therefore 20 epochs would run in time of 24 sec*20 =480secs (8 mins) as below.



```
161/161 [==============================] - 24s 150ms/step - loss: 0.0685 - accuracy: 0.9749 - val_loss: 0.1786 - val_accuracy: 0.9406
Epoch 11/20
161/161 [==============================] - 24s 150ms/step - loss: 0.0484 - accuracy: 0.9852 - val_loss: 0.2087 - val_accuracy: 0.9528
Epoch 12/20
161/161 [==============================] - 24s 151ms/step - loss: 0.0393 - accuracy: 0.9854 - val_loss: 0.1880 - val_accuracy: 0.9423
Epoch 13/20
161/161 [==============================] - 24s 151ms/step - loss: 0.0354 - accuracy: 0.9872 - val_loss: 0.2173 - val_accuracy: 0.9441
Epoch 14/20
161/161 [==============================] - 24s 150ms/step - loss: 0.0604 - accuracy: 0.9805 - val_loss: 0.1740 - val_accuracy: 0.9336
Epoch 15/20
161/161 [==============================] - 24s 150ms/step - loss: 0.0732 - accuracy: 0.9802 - val_loss: 0.2337 - val_accuracy: 0.9353
Epoch 16/20
161/161 [==============================] - 24s 150ms/step - loss: 0.0476 - accuracy: 0.9852 - val_loss: 0.1437 - val_accuracy: 0.9598
Epoch 17/20
161/161 [==============================] - 24s 150ms/step - loss: 0.0177 - accuracy: 0.9946 - val_loss: 0.2698 - val_accuracy: 0.9458
Epoch 18/20
161/161 [==============================] - 24s 150ms/step - loss: 0.0738 - accuracy: 0.9733 - val_loss: 0.2673 - val_accuracy: 0.9406
Epoch 19/20
161/161 [==============================] - 24s 151ms/step - loss: 0.0290 - accuracy: 0.9891 - val_loss: 0.2767 - val_accuracy: 0.9336
Epoch 20/20
161/161 [==============================] - 24s 152ms/step - loss: 0.0214 - accuracy: 0.9944 - val_loss: 0.2657 - val_accuracy: 0.9493
```

Figure 15. Computation Time for VGG-16

Figure 16 represents the confusion Matrix of VGG-16 applied to test data. The left-to-right diagonal number in the matrix represents the True positives in a multi-class classification. The same is represented in Table 2, correct predictions are 1246 out of 1311 test images.



```python
# Compute the confusion matrix
cm = confusion_matrix(argmax_indices_pred, argmax_indices_test)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[404   1  15   0]
 [  0 266   2   0]
 [  1  32 287  11]
 [  0   1   2 289]]
```

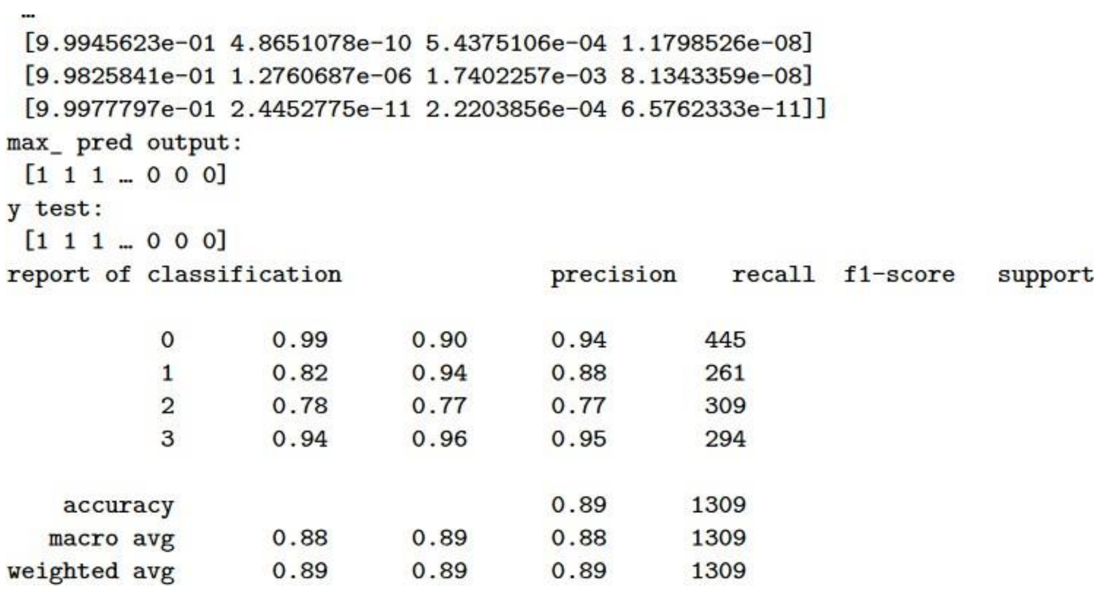
Figure 16. Confusion Matrix of VGG-16

Table 2: Visual Geometry Group (VGG-16) Confusion Matrix

Predicted/ Ground Truth	No Tumor	Glioma	Meningioma	Pituitary
No Tumor	404	1	15	0
Glioma	0	266	2	0
Meningioma	1	32	287	11
Pituitary	0	1	2	289

<u>VGG-19 Evaluation</u>

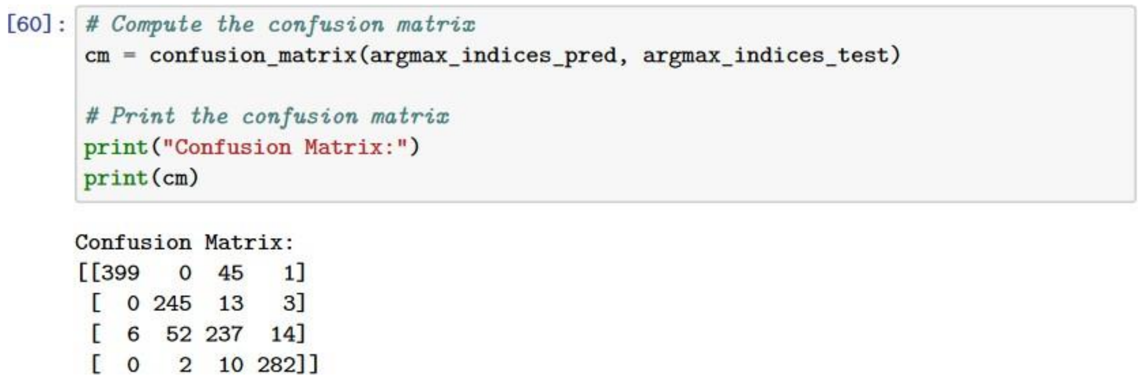
Implementation of VGG-19 in a sequential/manual manner provides an output that is less accurate than Kera's inbuilt methods (VGG-19). We are able to achieve about 89% accuracy. The classification report is shown in Figure 17. Precision is 99%, recall is 90%, f1-score is 94%.



```
 …
 [9.9945623e-01 4.8651078e-10 5.4375106e-04 1.1798526e-08]
 [9.9825841e-01 1.2760687e-06 1.7402257e-03 8.1343359e-08]
 [9.9977797e-01 2.4452775e-11 2.2203856e-04 6.5762333e-11]]
max_ pred output:
 [1 1 1 … 0 0 0]
y test:
 [1 1 1 … 0 0 0]
report of classification              precision    recall  f1-score   support

           0       0.99      0.90      0.94       445
           1       0.82      0.94      0.88       261
           2       0.78      0.77      0.77       309
           3       0.94      0.96      0.95       294

    accuracy                           0.89      1309
   macro avg       0.88      0.89      0.88      1309
weighted avg       0.89      0.89      0.89      1309
```

Figure 17. Classification Report of VGG-19

Figure 18 and Table 3 represent the confusion matrix of applying VGG-19 on the brain tumor dataset. On a total of 1303 test images, the model is able to predict True values as 1163 images. This gives an accuracy of 89%.



```
[60]: # Compute the confusion matrix
      cm = confusion_matrix(argmax_indices_pred, argmax_indices_test)

      # Print the confusion matrix
      print("Confusion Matrix:")
      print(cm)

      Confusion Matrix:
      [[399   0  45   1]
       [  0 245  13   3]
       [  6  52 237  14]
       [  0   2  10 282]]
```

Figure 18. Confusion Matrix of VGG-19

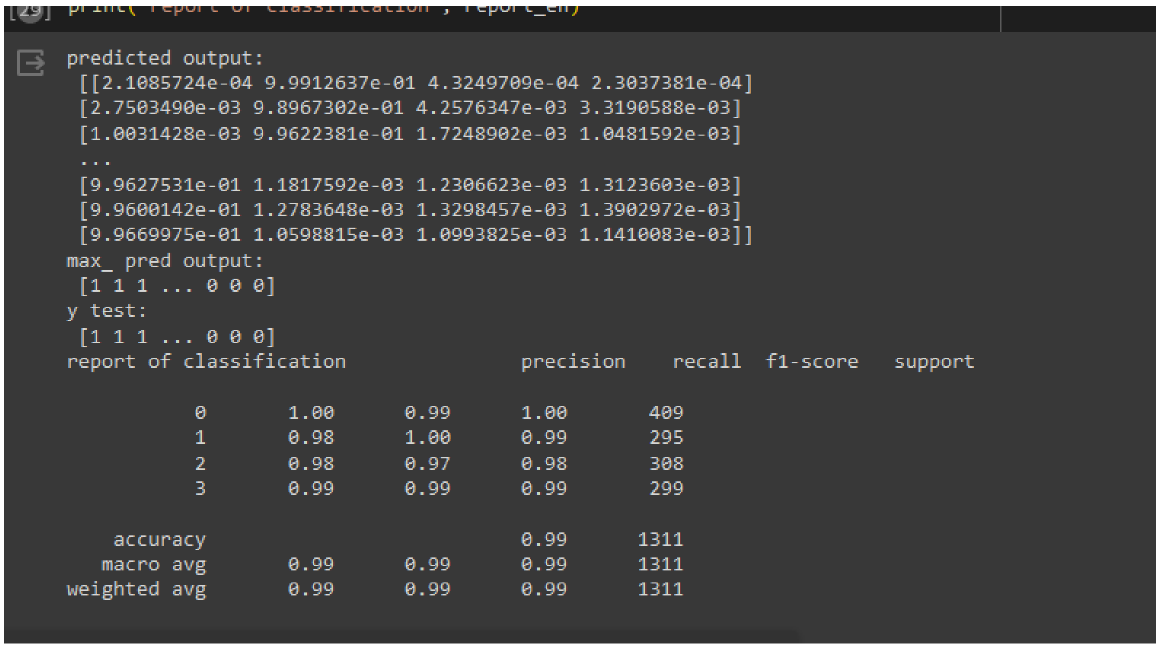
Table 3. Visual Geometry Group (VGG-19) Confusion Matrix

Predicted/ Ground Truth	No Tumor	Glioma	Meningioma	Pituitary
No Tumor	399	0	45	1
Glioma	0	245	13	3
Meningioma	6	52	237	14
Pituitary	0	2	10	282

<u>EfficientNet Evaluation</u>

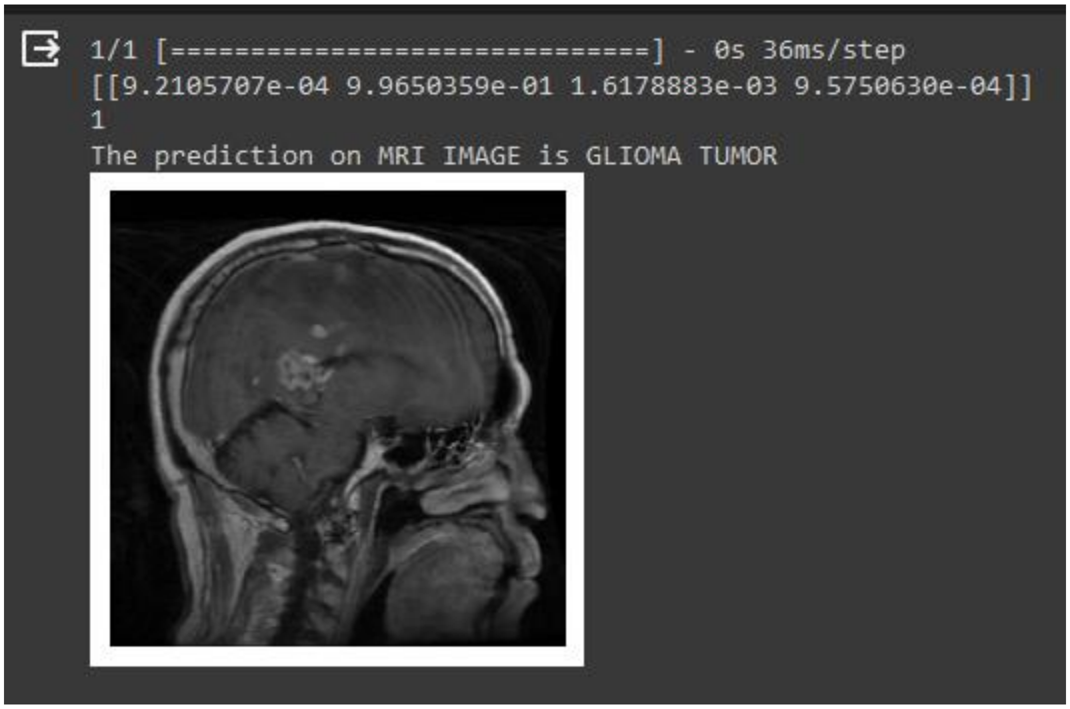
EfficientNet provides an accuracy of 99% over 20 epochs. It consumes a significant amount of computation time at each epoch. Considering the computation time, it took around 82s per epoch which took around 1640 secs.

This is expensive compared to the VGG-16 implementation. Figure 19 represents the classification report and Figure 20 represents the example classification using EfficientNet.



```
[29] print("report of classification", report_en)

predicted output:
 [[2.1085724e-04 9.9912637e-01 4.3249709e-04 2.3037381e-04]
 [2.7503490e-03 9.8967302e-01 4.2576347e-03 3.3190588e-03]
 [1.0031428e-03 9.9622381e-01 1.7248902e-03 1.0481592e-03]
 ...
 [9.9627531e-01 1.1817592e-03 1.2306623e-03 1.3123603e-03]
 [9.9600142e-01 1.2783648e-03 1.3298457e-03 1.3902972e-03]
 [9.9669975e-01 1.0598815e-03 1.0993825e-03 1.1410083e-03]]
max_ pred output:
 [1 1 1 ... 0 0 0]
y test:
 [1 1 1 ... 0 0 0]
report of classification               precision    recall  f1-score   support

           0       1.00      0.99      1.00       409
           1       0.98      1.00      0.99       295
           2       0.98      0.97      0.98       308
           3       0.99      0.99      0.99       299

    accuracy                           0.99      1311
   macro avg       0.99      0.99      0.99      1311
weighted avg       0.99      0.99      0.99      1311
```

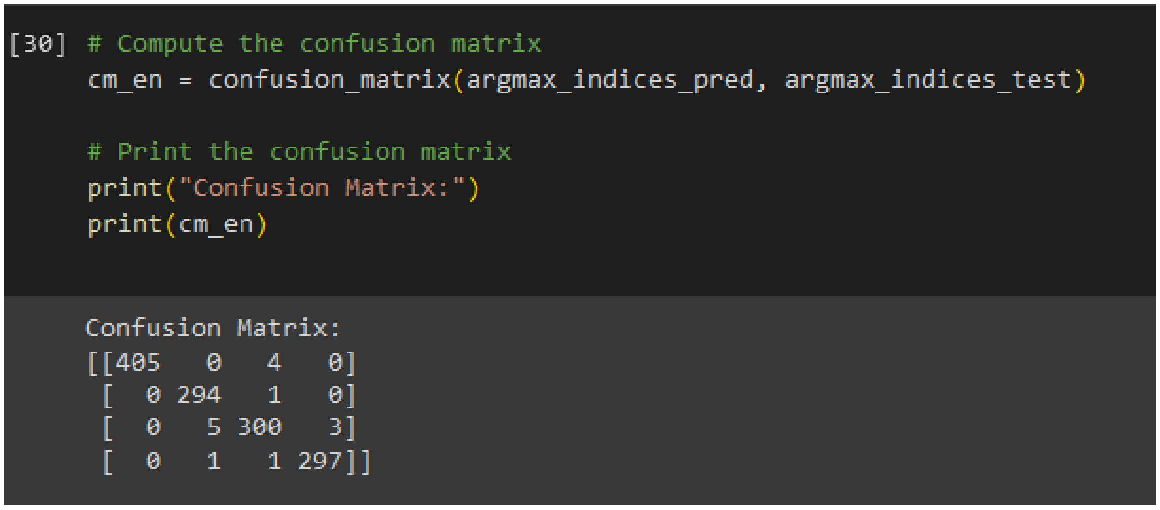
Figure 19. EfficientNet Classification Report



```
1/1 [==============================] - 0s 36ms/step
[[9.2105707e-04 9.9650359e-01 1.6178883e-03 9.5750630e-04]]
1
The prediction on MRI IMAGE is GLIOMA TUMOR
```

Figure 20. Example Classification using EfficientNet

Considering the confusion matrix, this looks as below. The other parameters like accuracy, sensitivity, f1-score, and precision can be derived from the confusion Matrix. Figure 21 and Table 3 represent the confusion matrix of EfficientNet(B3) where 1296 correct predictions out of 1311 test images.



```
[30] # Compute the confusion matrix
     cm_en = confusion_matrix(argmax_indices_pred, argmax_indices_test)

     # Print the confusion matrix
     print("Confusion Matrix:")
     print(cm_en)
```

```
     Confusion Matrix:
     [[405   0   4   0]
      [  0 294   1   0]
      [  0   5 300   3]
      [  0   1   1 297]]
```

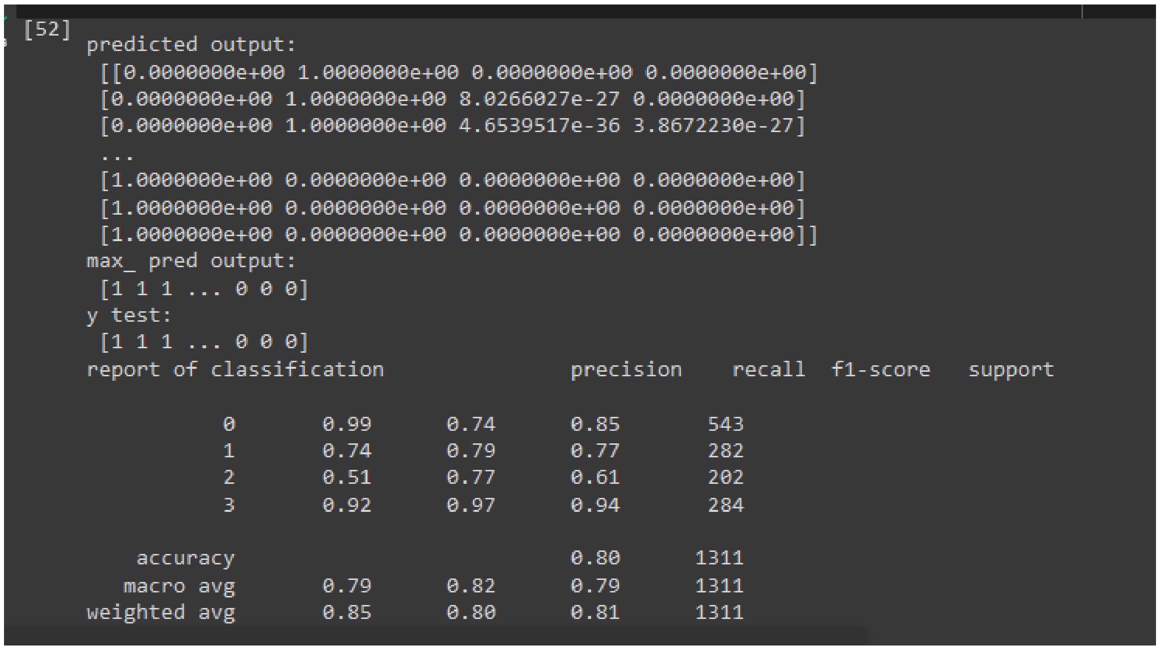
Figure 21. Confusion Matrix for EfficientNet (B3)

Table 4: EfficientNet(B3) Confusion Matrix

Predicted/ Ground Truth	No Tumor	Glioma	Meningioma	Pituitary
No Tumor	405	0	4	0
Glioma	0	294	1	0
Meningioma	0	5	300	3
Pituitary	0	1	1	297

MLP Evaluation

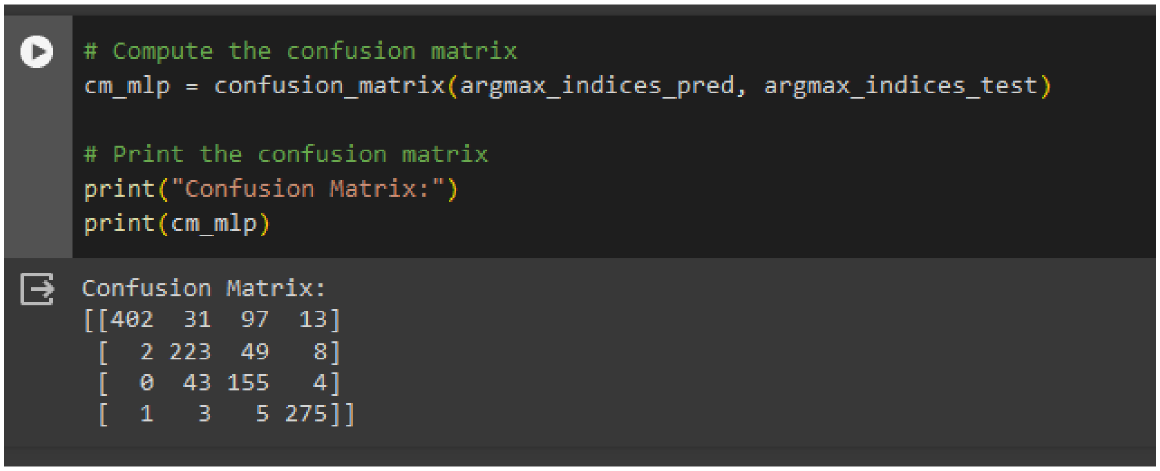
Multi-layer Perceptron (MLP) is versatile and can be applied to various tasks, including classification, regression, and pattern recognition. However, applying it to image classification is not a great idea because it cannot deal with a two-dimensional array of image data. Flattening the image will make it 2D to 1D. Results will be impacted. The observed results are as below in Figure 22. We can achieve 80% accuracy on test data.



```
[52]
     predicted output:
      [[0.0000000e+00 1.0000000e+00 0.0000000e+00 0.0000000e+00]
      [0.0000000e+00 1.0000000e+00 8.0266027e-27 0.0000000e+00]
      [0.0000000e+00 1.0000000e+00 4.6539517e-36 3.8672230e-27]
      ...
      [1.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]
      [1.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]
      [1.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]]
     max_ pred output:
      [1 1 1 ... 0 0 0]
     y test:
      [1 1 1 ... 0 0 0]
     report of classification               precision    recall  f1-score   support

                0       0.99      0.74      0.85       543
                1       0.74      0.79      0.77       282
                2       0.51      0.77      0.61       202
                3       0.92      0.97      0.94       284

         accuracy                           0.80      1311
        macro avg       0.79      0.82      0.79      1311
     weighted avg       0.85      0.80      0.81      1311
```

Figure 22. MLP Classification Report

Figure 23 and Table 5 represent the confusion matrix of MLP where 1055 correct predictions out of 1311 test images. Accuracy went low because of the model's nature of dealing with images.



```
# Compute the confusion matrix
cm_mlp = confusion_matrix(argmax_indices_pred, argmax_indices_test)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm_mlp)
```

```
Confusion Matrix:
[[402  31  97  13]
 [  2 223  49   8]
 [  0  43 155   4]
 [  1   3   5 275]]
```

Figure 23. Confusion Matrix using MLP

Table 5: Multi-Layer Perceptron (MLP) Confusion Matrix

Predicted/ Ground Truth	No Tumor	Glioma	Meningioma	Pituitary
No Tumor	402	31	97	13
Glioma	2	223	49	8
Meningioma	0	43	155	4 ▶
Pituitary	1	3	5	275

Comparison

Performance evaluation task is done by researchers on augmented MRI images. For performance evaluation of the model, statistical methods of evaluation are used like sensitivity, specificity, precision, recall, *f* measure, false positive (FP) ratio, and Receiver Operating Characteristic (ROC) curve [1]. As per research, the performance of a CNN model depends on feature extraction, preprocessing the images, classifying them as positives and negatives, and applying geometrical data augmentation techniques on brain MRI images [1]. Above all of them, we also need to check any unseen patterns of images and compare the results with other conventional models like MLP [16]. Figure 24 illustrates accuracy comparison among four models. Figure 25 represents the computation time in seconds for each model. Time trade off can be chosen w.r.t accuracy of model by the user.

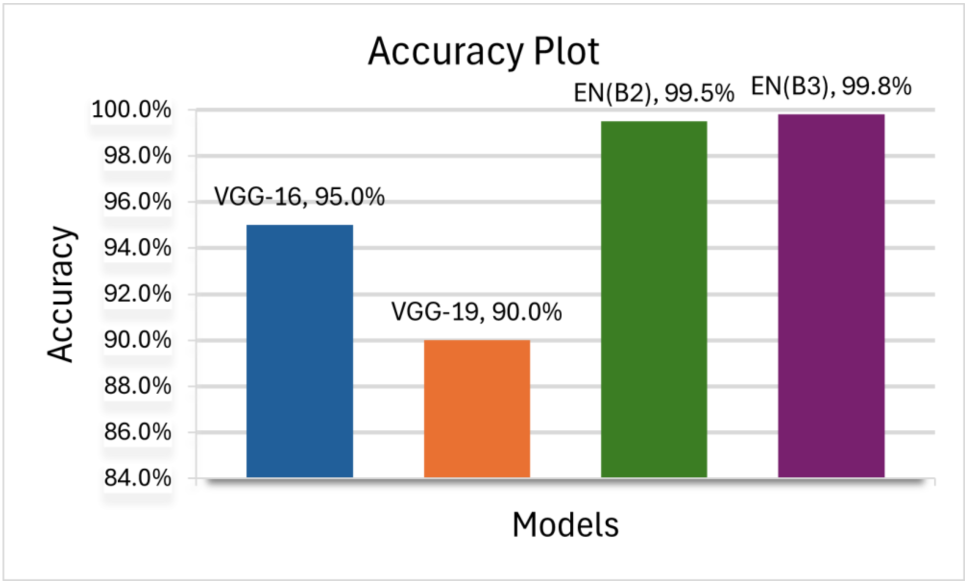


Figure 24. Accuracy Comparison

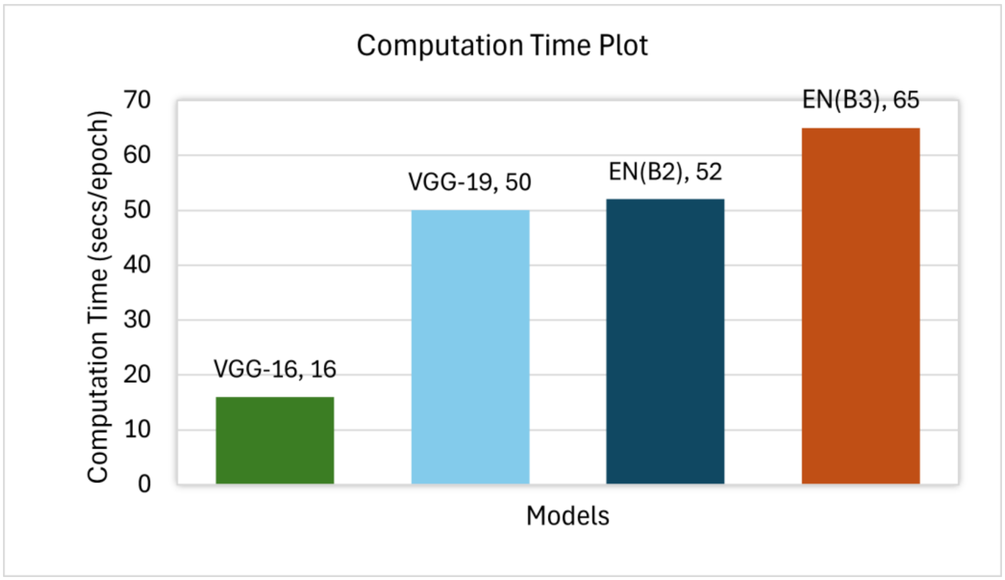


Figure 25. Computation Time Comparison

CNN models outperform with an initial 10 epochs and the gradual addition of epochs will result in greater accuracy trading off with computation cost. As in research on brain MRI [1], 49 epochs are applied on CNN to achieve 100% accuracy which leads to high usage of computation resources and time.

<u>Plots of VGG-16 vs EfficientNet(B3)</u>

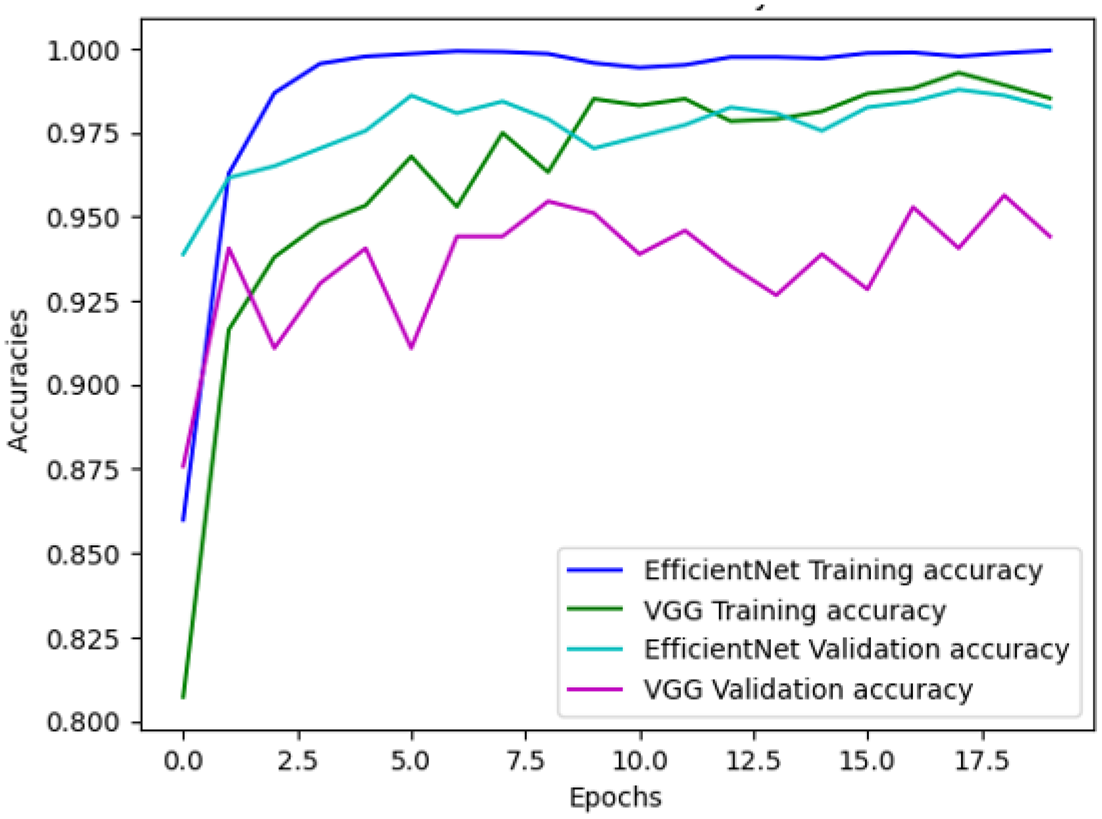


Figure 26. Accuracy Plots

Figure 26 From the implemented project, EfficientNet performs high accuracy with a smaller number of epochs and the tradeoff for the B3 layer is computation time. VGG-16 performs less accuracy at initial epochs and improves over iterations. Increase in variations of VGG and EfficientNet models like (VGG-19, VGG21, B0, B3, B5, B7) will result in higher accuracy at initial epochs due to the convolutional layers processing.

Loss Plots of VGG-16 vs EfficientNet

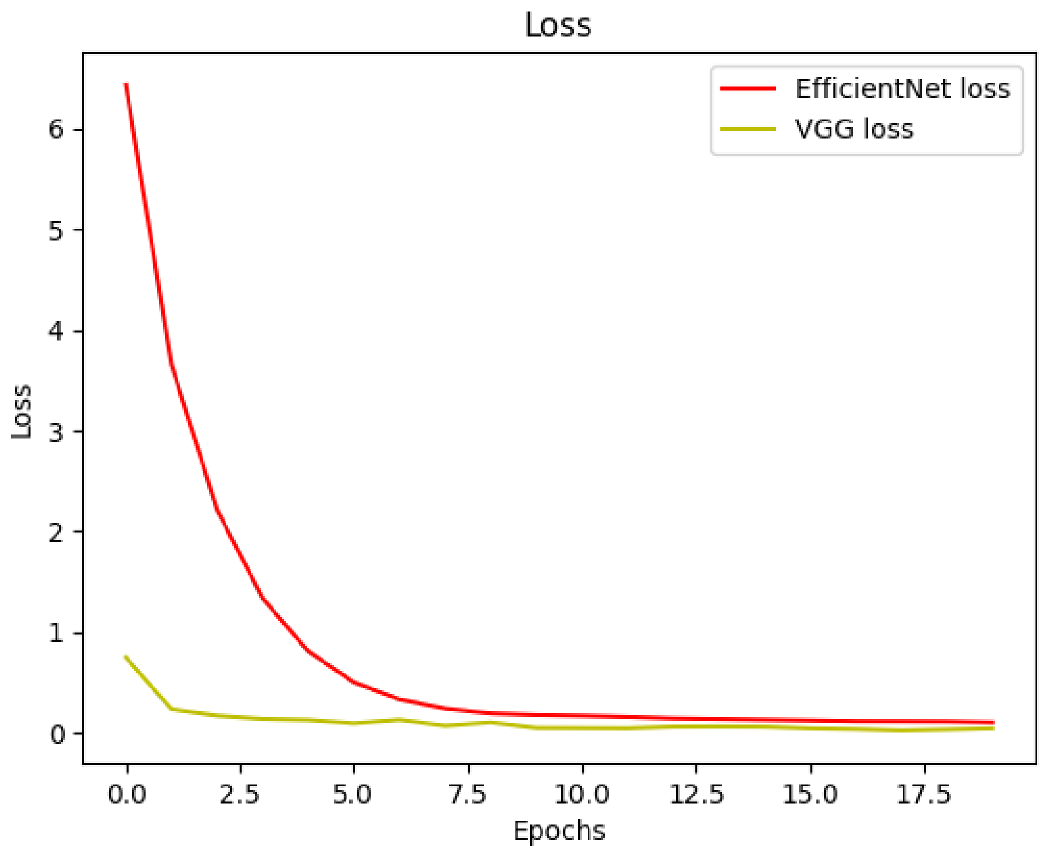


Figure 27. Loss Plots

Figure 27 represents Loss curves that converge after certain epochs which illustrate that both models perform equally after iterations. Both of the loss values converge after 12th epoch.

CHAPTER FIVE

SYSTEM DESIGN

The system design of this application started with the process of defining architecture, components, modules, interfaces, and data for this application. On a high-level representation, the application started with UML diagrams, identifying the components involved in constructing, modifying, and data transformation to fit.

State Diagram

A state diagram represents the various states that a system can exist in and the transitions between those states. States represent different conditions or situations. Transitions indicate how a system moves from one state to another. Events indicate the actions that trigger a system's state. Figure 28 represents different possible states that the proposed application will have like the launch web page, image input, preprocessing, image processing, ML model to classify, and output page. The transition directions are shown below.

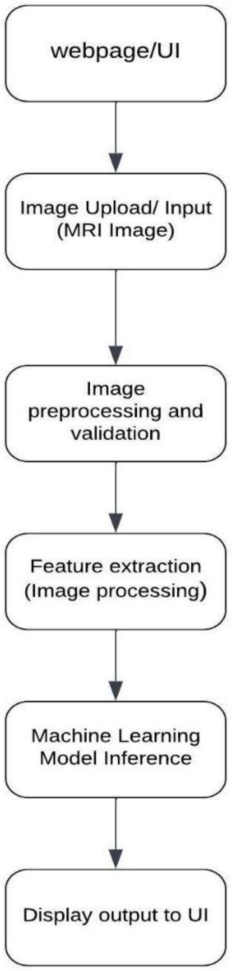

webpage/UI
Image Upload/ Input (MRI Image)
Image preprocessing and validation
Feature extraction (Image processing)
Machine Learning Model Inference
Display output to UI

Figure 28. State Diagram

Use Case Diagram

The use case diagram is shown in Figure 29 representing the interactions between actors (in this case user) and the system to achieve the specific goals. The end user is given access to the system, uploading the MRI image, clicking on predict, and checking the accuracy and classification of the input image.

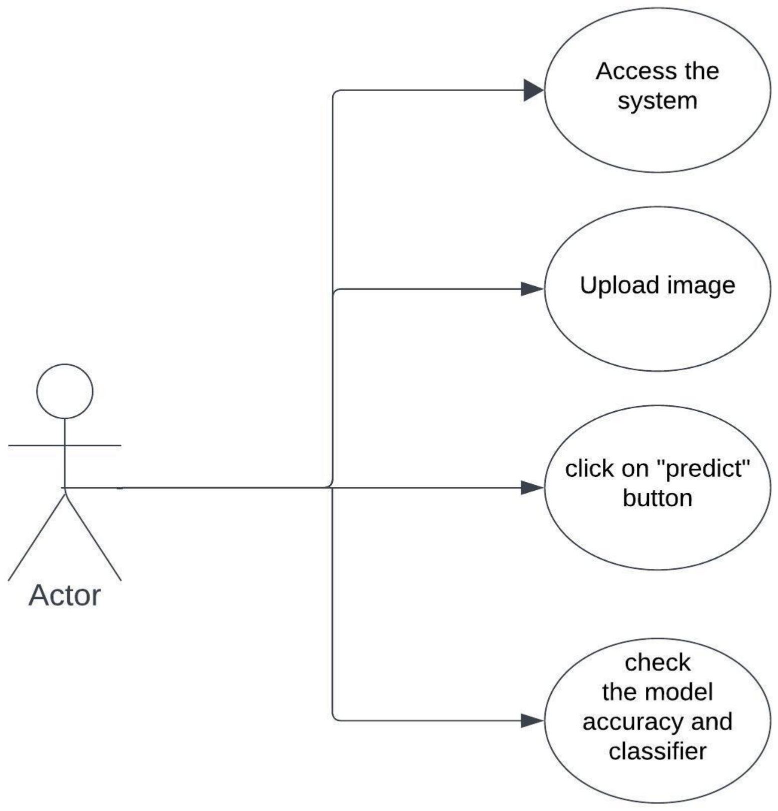

Access the system
Upload image
click on "predict" button
check the model accuracy and classifier
Actor

Figure 29. Use Case Diagram

Sequence Diagram

The sequence diagram in Figure 30 represents interactions between the components involved in the system in chronological order. The actor represents the end user and four different entities namely webapp, ML models, and model evaluation. Each interaction is represented with a message on top describing the communication of objects. Interactions start with the application launch and end with identifying the tumor classification with accuracy.

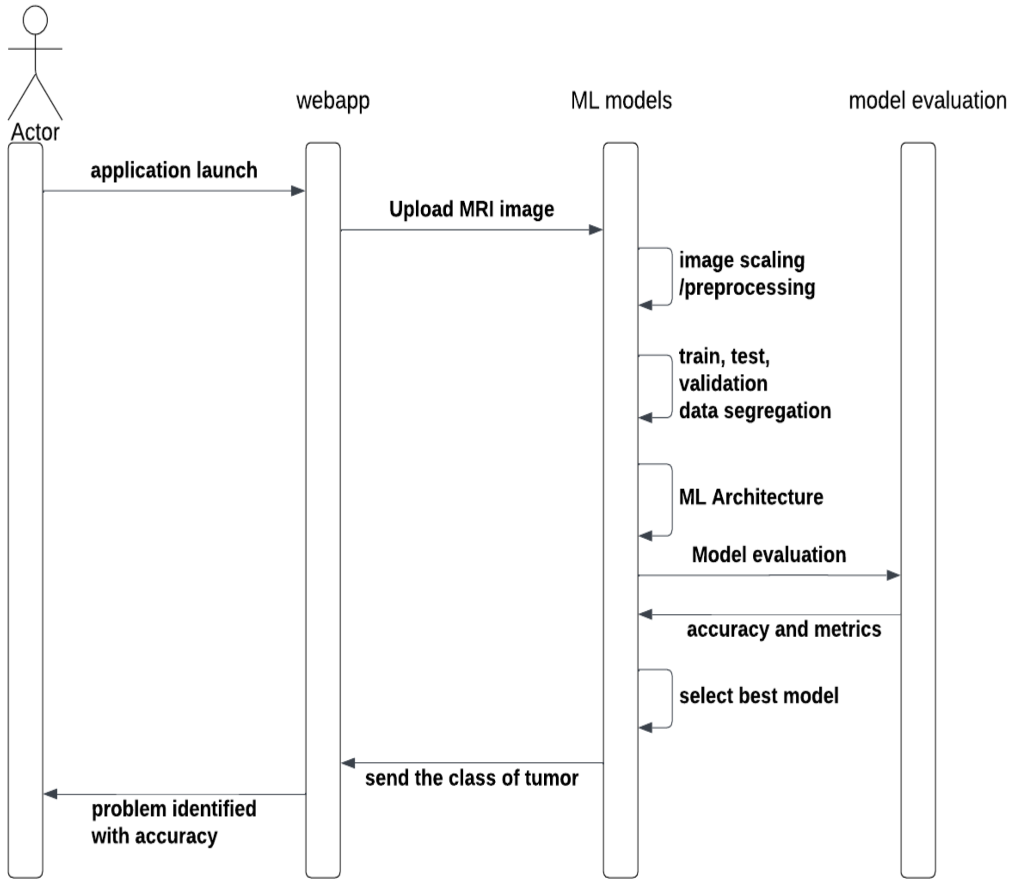


Figure 30. Sequence Diagram

Class Diagram

The class diagram is a static structure diagram that represents the classes in a system, along with their attributes, methods, and relationships. Figure 31 represents the different classes and interfaces. Streamlit, TensorFlow, Keras, Numpy, ImageOps, and sklearn are predefined packages required for implementation.

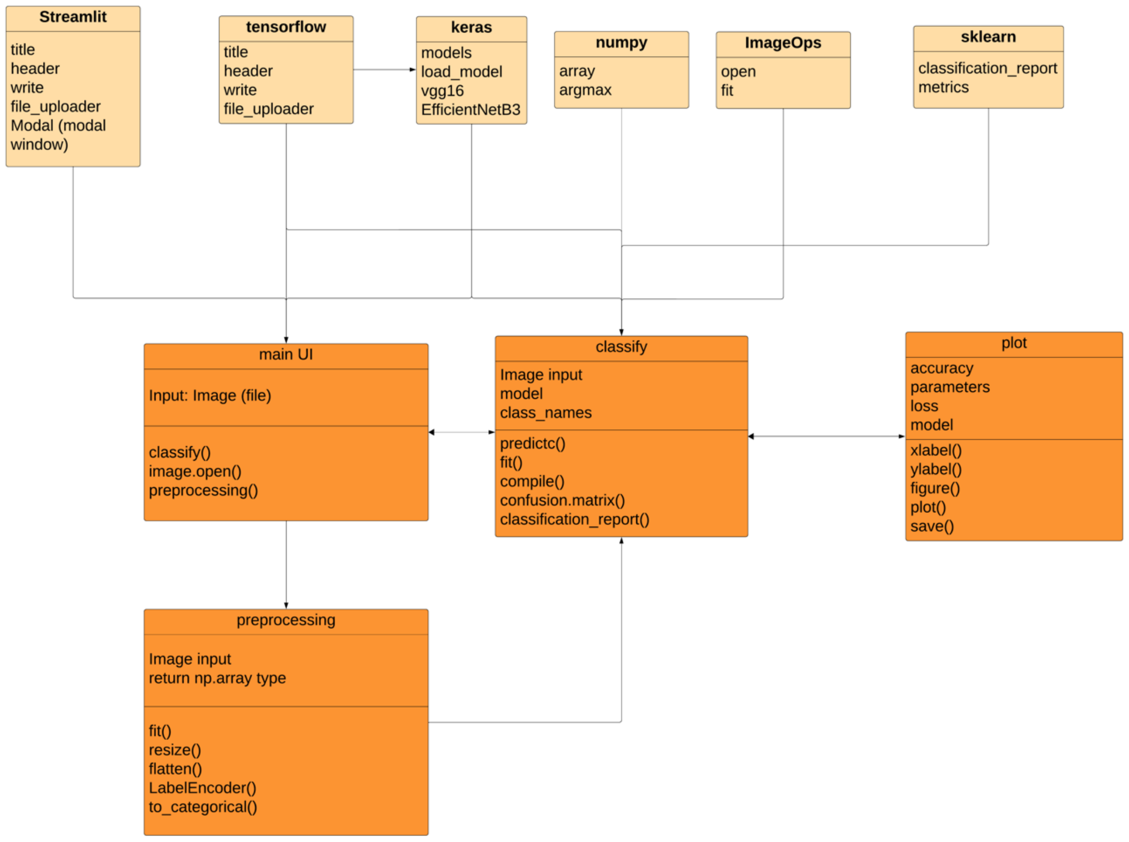
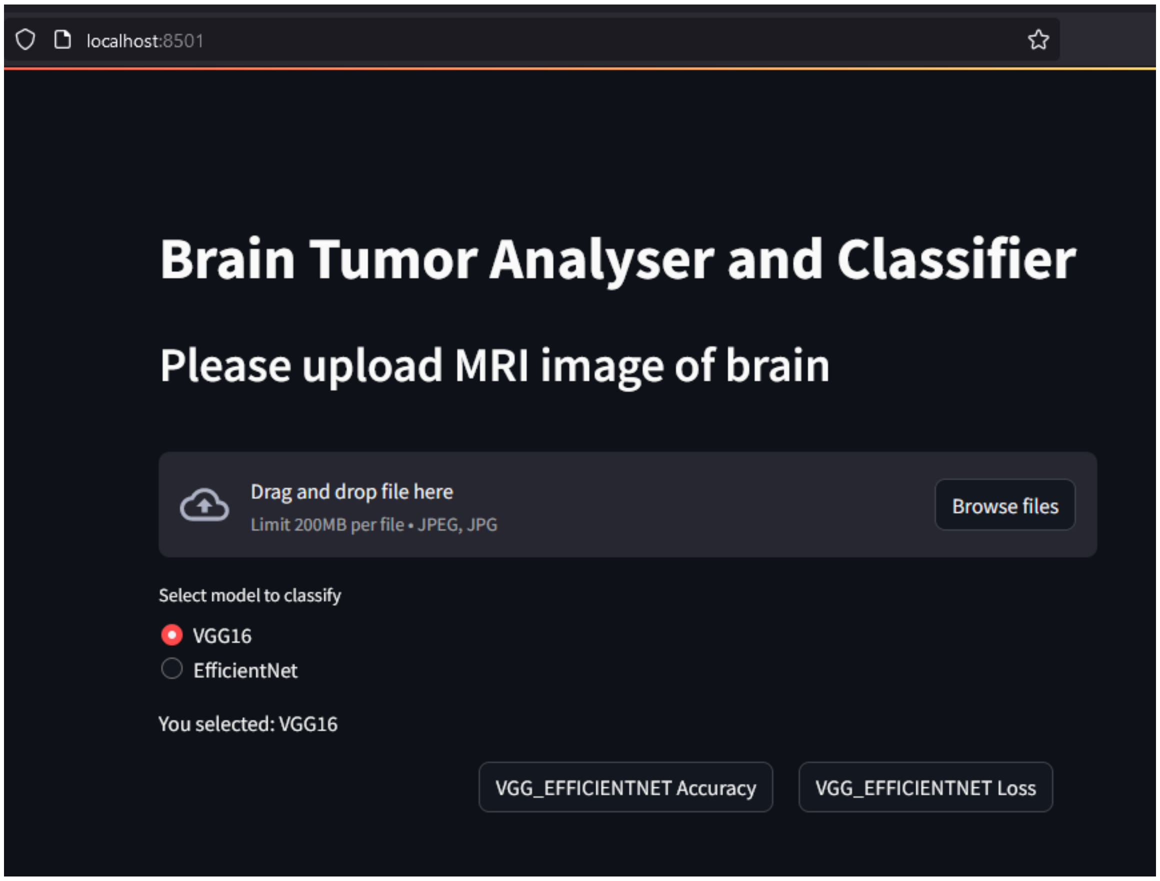


Figure 31. Class Diagram

Unit Test Screens

Application Launch Page

For the ease of the end user and to make it accessible, the front layer in Python is added to hide the back-end layers. The front-end layer is built on the stream-lit library of Python. Front front-end layer interacts with the ML models through the exported h5 file form. Users will be able to input the image. The application launch page is represented in Figure 32.



localhost:8501

Brain Tumor Analyser and Classifier

Please upload MRI image of brain

Drag and drop file here
Limit 200MB per file • JPEG, JPG

Select model to classify

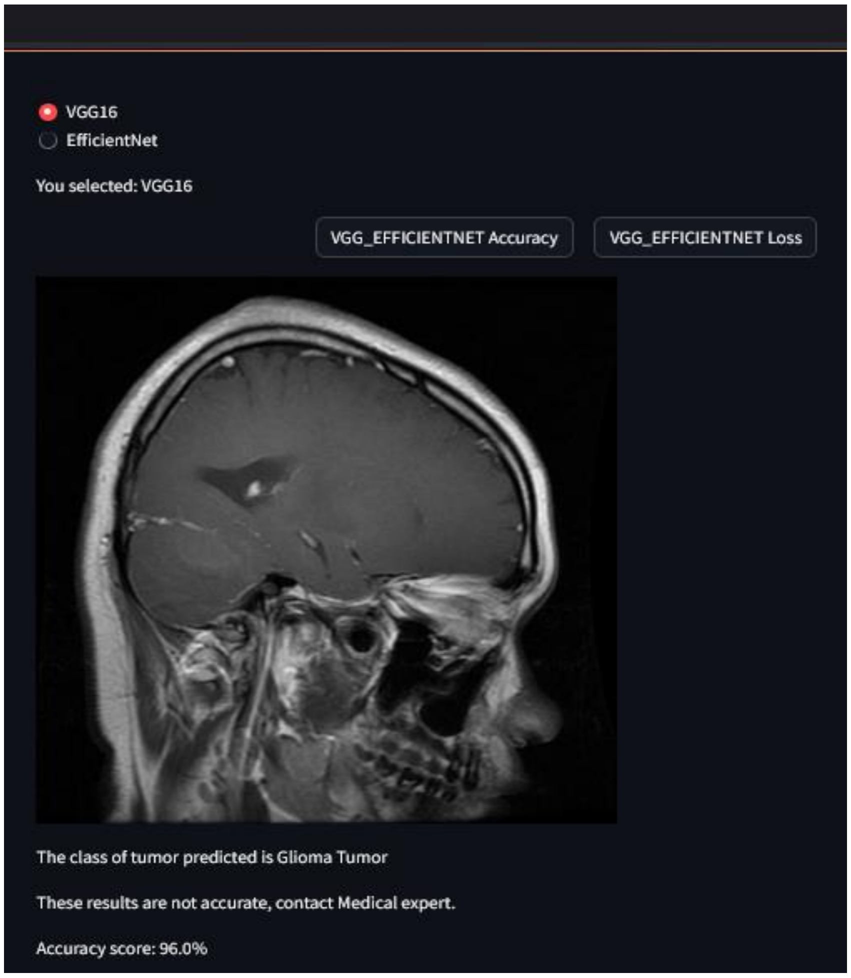
[✓ VGG16]
EfficientNet

You selected: VGG16

Figure 32. Application Launch Screen

VGG-16 Output Screen

User upon selecting the desired ML model to classify, the application will send the parameters as VGG-16 model and classify the input image. Observed classification accuracy as 96%. The prediction matched with the ground truth which is in the dataset. Figure 31 displays the output screen using VGG-16.



You selected: VGG16

The class of tumor predicted is Glioma Tumor

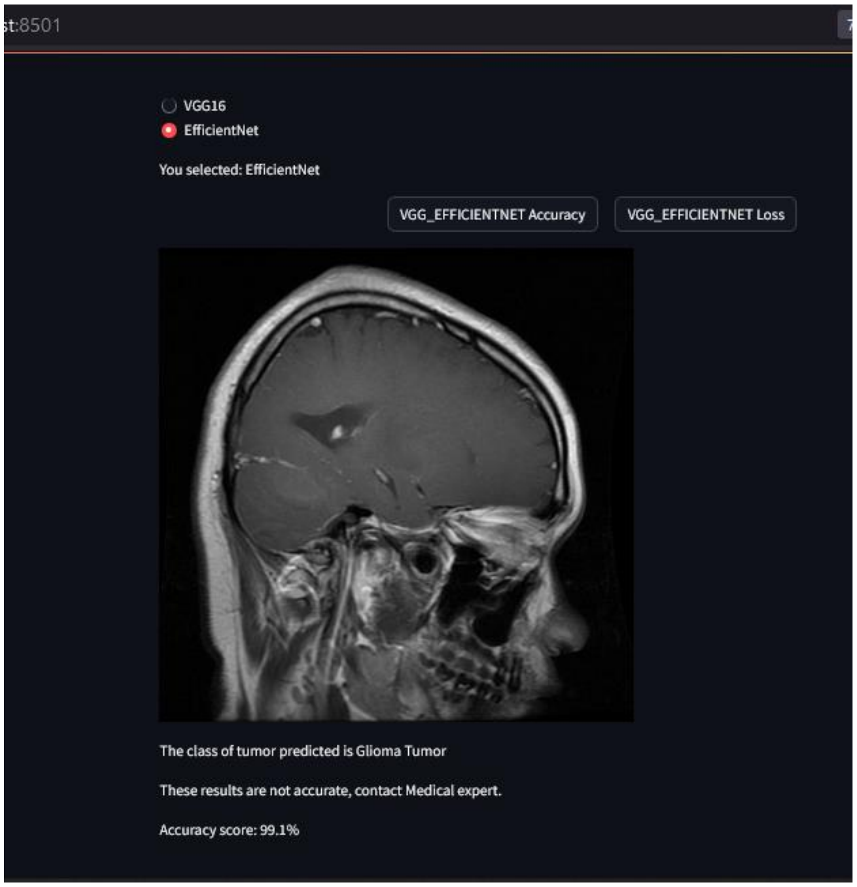
These results are not accurate, contact Medical expert.

Accuracy score: 96.0%

Figure 33. VGG-16 Classification Output

<u>EfficientNet Output Screen</u>

The user chose the EfficientNet model to classify the input image. EfficientNet is known to provide more accuracy compared to the VGG-16 model. Figure 34 represents the output screen to classify using the EfficientNet model. The accuracy is 99.1% as a Glioma Tumor and matches with ground truth.



You selected: EfficientNet

The class of tumor predicted is Glioma Tumor

These results are not accurate, contact Medical expert.

Accuracy score: 99.1%

Figure 34. EfficientNet Classification Results

CHAPTER SIX

CONCLUSION

In summary, the development of a web application that uses Python to predict the classification of brain tumors, in MRI scans are advancement in the field of imaging and diagnostics. By incorporating cutting-edge learning models like VGG-16, EfficientNet as well as a multi-layer perceptron, the accuracy and reliability of the prediction system are greatly improved.

The utilization of VGG-16 and EfficientNet brings networks (CNNs) to the forefront allowing the model to learn intricate patterns and features from the MRI [4] images. These architectures, combined with transfer learning techniques enable the network to utilize existing knowledge from extensive datasets even when there is limited labeled medical data available. This enhances the model's ability to generalize effectively.

The inclusion of layer perceptrons further enhances the predictive capabilities of this web application. These neural networks capture relationships within the data leading to performance in classifying different types of brain tumors. By combining these architectures, a comprehensive approach is taken toward tumor classification that accommodates variations in data distribution and ensures predictions across diverse cases.

The user-friendly design of this web application makes it accessible, for healthcare professionals who can leverage machine learning techniques without requiring technical expertise. The web interfaces real-time prediction abilities make the diagnostic process more efficient potentially allowing for the detection and intervention of patients with brain tumors.

Moving forward future improvements to the web application can focus on model training using updated datasets. This will ensure that the system remains skilled at handling evolving trends in imaging. Additionally, efforts can be directed toward expanding the model’s capabilities to detect and classify conditions. This will have an impact on health care further enhancing its effectiveness.

All the development of this web application represents an advancement in integrating artificial intelligence and medical imaging. It contributes significantly to improving accuracy and ultimately enhancing outcomes when it comes to identifying and classifying brain tumors.

Future Work

There is an extension idea at the top of this project to improve accuracy. Not limited to an existing dataset, if the model makes a wrong prediction user will be given the option to correct the output. All these wrong predictions are consolidated and at the end of the day, a batch job will be set run for including wrong instances to the training dataset. The model will run again on training data. This improves model accuracy day by day. Large organizations like Google today follow the same methodology to train existing models on new data.

APPENDIX A

SOURCE CODE AND DEPENDENCIES

The code below includes the main components of UI layers and backend layers. UI layer uses the .h5 file of the ML model to predict the outputs.

```
import streamlit as st
import tensorflow as tf
import keras
from keras.models import load_model
from PIL import Image
import numpy as np
from sklearn.metrics import classification_report

from util import classify
from streamlit_modal import Modal

#title part of UI
st.title('Brain Tumor Analyser and Classifier')
#header part
st.header('Please upload MRI image of brain')
#upload the file
file = st.file_uploader('',type=['jpeg','jpg'])
vggmodelpath='C:/Users/kvsk_/Downloads/vggmodel_rev1.h5'
efficientnetmodelpath='C:/Users/kvsk_/Downloads/efficientNetmodel_rev1.h5'
sel_model = st.radio(
    "Select model to classify",
    ["VGG-16", "EfficientNet" ],
    index=0,
)
print(sel_model)
st.write("You selected:", sel_model)
# display image
if (sel_model == "VGG-16"):
    modelpath = vggmodelpath
else:
    modelpath = efficientnetmodelpath


plots_container=st.container()
with plots_container:
    col1, col2, col3 = st.columns(3)
    with col2:
        modal_acc_vgg = Modal(key="Acc_plot",title="Plots")
        open_acc_vggmodal = st.button("VGG_EFFICIENTNET Accuracy")
        if open_acc_vggmodal:
            modal_acc_vgg.open()


    with col3:
        vgg_modal_loss = Modal(key="Loss_plot_vgg",title="Plots")
        open_loss_vgg_modal = st.button("VGG_EFFICIENTNET Loss")
        if open_loss_vgg_modal:
            vgg_modal_loss.open()




if modal_acc_vgg.is_open():
    with modal_acc_vgg.container():
        #st.write("testerss")
        vgg_acc_image =
Image.open('C:/Users/kvsk_/Downloads/vggvsen_acc.png')
        st.image(vgg_acc_image,caption='Accuracy plot', width=400)

if vgg_modal_loss.is_open():
        with vgg_modal_loss.container():
            vgg_loss_image =
Image.open('C:/Users/kvsk_/Downloads/vggvsen_loss.png')
            st.image(vgg_loss_image,caption='Loss plot',width=400)


model = tf.keras.models.load_model(modelpath,compile=False)

with open('C:/Users/kvsk_/Downloads/labels/labels.txt','r') as f:
    class_names = [a[:-1].split(' ')[1] for a in f.readlines()]
    f.close()
print(class_names)

if file is not None :
    print("inside classification")
    image = Image.open(file).convert('RGB')
    st.image(image,width = 500)
    #classify image
    class_name, conf_score = classify(image,model,class_names)

    # Generate a classification report
    #report = classification_report(y_test, y_pred,
target_names=iris.target_names)
```

```
    # Print the classification report
    #print("Classification Report:\n", report)
    #write classification
    if (class_name == "Healthy brain"):
        st.write("No tumor found, it is a","{}".format(class_name))
        st.write("These results are not accurate, contact Medical expert.")
    else:
        st.write("The class of tumor predicted is","{}".format(class_name))
        st.write("These results are not accurate, contact Medical expert.")

    st.write("Accuracy score: {}%".format(int(conf_score * 1000) / 10))

UTIL.py:
import base64

import streamlit as st
from PIL import ImageOps, Image
import numpy as np
import matplotlib.pyplot as plt
from keras.utils import img_to_array


def classify(image, model, class_names):
        #convert image to 224,224
    img = ImageOps.fit(image, (224, 224), Image.Resampling.LANCZOS)

    plt.figure(figsize=(10, 10))
    plt.subplot(1,3,1)
    plt.imshow(img, cmap="gray")
    plt.axis('off')
    plt.show
    i = img_to_array(img)
    input_arr = np.array([i])
    predict_x=model.predict(input_arr)
    class_x=np.argmax(predict_x)
    pred_accuracy = predict_x.max()
    print(predict_x)
    print(class_x)
    if(class_x == 1):
        print("The prediction on MRI IMAGE is GLIOMA TUMOR")
        class_label = "Glioma Tumor"
    elif(class_x == 2):
        print("The prediction on MRI IMAGE is MENINGIOMA TUMOR")
        class_label = "Meningioma Tumor"
    elif(class_x == 3):
        print("The prediction on MRI IMAGE is PITUITARY TUMOR")
        class_label = "Pituitary Tumor"
    else:
        print("No Tumor is found in MRI image")
        class_label = "Healthy brain"
    return class_label, pred_accuracy
```

Models implementation:

```
import tensorflow as tf
from zipfile import ZipFile
import os,glob
import cv2
from tqdm.notebook import tqdm_notebook as tqdm
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Convolution2D, Dropout, Dense,MaxPooling2D
from keras.layers import BatchNormalization
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from tensorflow.keras import regularizers

import keras
from keras.preprocessing import image
from keras.utils import load_img,img_to_array
import matplotlib.pyplot as plt
from statistics import mean
from keras.applications import VGG-16
from tensorflow import keras

from keras.optimizers import Adam, Adamax
from keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras


from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,
GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.models import Model
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.neural_network import MLPClassifier

def notumor():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_train_val.append(img)
      y_train_val.append('0')
def glioma():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_train_val.append(img)
      y_train_val.append('1')
def meningioma():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_train_val.append(img)
      y_train_val.append('2')
def pituitary():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_train_val.append(img)
      y_train_val.append('3')
X_train_val = []

y_train_val = []
def notumor_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_test.append(img)
      y_test.append('0')
def glioma_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_test.append(img)
      y_test.append('1')
def meningioma_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_test.append(img)
      y_test.append('2')
def pituitary_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224)).flatten()
      X_test.append(img)
      y_test.append('3')
X_test = []
y_test = []
os.chdir('/home/jovyan/Training/glioma')
glioma()
os.chdir('/home/jovyan/Training/meningioma')
meningioma()
os.chdir('/home/jovyan/Training/pituitary')
pituitary()
os.chdir('/home/jovyan/Training/notumor')
notumor()

os.chdir('/home/jovyan/Testing/glioma')
glioma_test()
os.chdir('/home/jovyan/Testing/meningioma')
meningioma_test()
os.chdir('/home/jovyan/Testing/pituitary')
pituitary_test()
os.chdir('/home/jovyan/Testing/notumor')
notumor_test()
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=70)
# Assume X and y are your features and target variable, respectively
#X_temp, X_test, y_temp, y_test = train_test_split(X_train_val, y_train_val,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,
test_size=0.10, random_state=42)


print ("Shape of an image in X_train: ", X_train[0].shape)
print ("Shape of an image in X_test: ", X_test[0].shape)
print ("Shape of an image in X_val: ", X_val[0].shape)
```

```
le = preprocessing.LabelEncoder()
y_train = le.fit_transform(y_train)
y_val = le.fit_transform(y_val)
y_test = le.fit_transform(y_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_val = tf.keras.utils.to_categorical(y_val, num_classes=4)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=4)
y_train = np.array(y_train)
X_train = np.array(X_train)
y_test = np.array(y_test)
X_test = np.array(X_test)
y_val = np.array(y_val)
X_val = np.array(X_val)
print("X_train Shape: ", X_train.shape)
print("X_test Shape: ", X_test.shape)
print("y_train Shape: ", y_train.shape)
print("y_test Shape: ", y_test.shape)
print("X_val Shape: ", X_val.shape)
print("y_val Shape: ", y_val.shape)
#include validation set in model.fit where 10% of training data


img_rows, img_cols = 224, 224
vgg = VGG-16.VGG-16(weights = 'imagenet',include_top = False,input_shape =
(img_rows, img_cols, 3) )
for layer in vgg.layers:
    layer.trainable = False

for (i,layer) in enumerate(vgg.layers):
    print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
def lw(bottom_model, num_classes):
    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(num_classes,activation='softmax')(top_model)
    return top_model
num_classes = 4
FC_Head = lw(vgg, num_classes)
model = Model(inputs = vgg.input, outputs = FC_Head)
print(model.summary())
```

```
model.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics =
['accuracy'])
history_vgg = model.fit(X_train,y_train,
                 epochs=20,
                 validation_data=(X_val,y_val))

acc_vgg = history_vgg.history['accuracy']
print("Training Accuracy : ",mean(acc_vgg))
val_acc_vgg = history_vgg.history['val_accuracy']
print("Validation Accuracy : ",mean(val_acc_vgg))
loss_vgg = history_vgg.history['loss']
val_loss_vgg = history_vgg.history['val_loss']

epochs = range(20)
plt.xlabel("Epochs")
plt.ylabel("Accuracies ")
#plt.plot(epochs, loss, 'r', label='loss')
plt.plot(epochs, val_acc_vgg, 'b', label='Validation accuracy')
plt.title('Validation accuracy')
plt.legend(loc=0)
# Export the plot to a PNG file
#plt.savefig("Accuracy_plot.png",format="png")
plt.figure()

plt.show()

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(epochs, loss_vgg, 'r', label='loss')
#plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Loss')
plt.legend(loc=0)
# Export the plot to a PNG file
#plt.savefig("loss_plot.png",format="png")
plt.figure()

plt.show()
#uncomment to save the model for front end.
#model.save('vggmodel_web.h5')
img = load_img('/content/drive/MyDrive/ML_Dataset/Testing/glioma/Te-
gl_0273.jpg', target_size=(224,224))
plt.figure(figsize=(10, 10))
plt.subplot(1,3,1)
plt.imshow(img, cmap="gray")
plt.axis('off')
plt.show
i = img_to_array(img)
input_arr = np.array([i])
predict_x=model.predict(input_arr)
class_x=np.argmax(predict_x)
print(predict_x)
print(class_x)
if(class_x == 1):
   print("The prediction on MRI IMAGE is GLIOMA TUMOR")
elif(class_x == 2):
   print("The prediction on MRI IMAGE is MENINGIOMA TUMOR")
elif(class_x == 3):
   print("The prediction on MRI IMAGE is PITUITARY TUMOR")
else:
   print("The MRI Image is Of HEALTHY BRAIN")
#evaluation: accuracy, precision, f1-score, recall.
#print the sklearn classification report
# Make predictions on the test set
y_pred = model.predict(X_test)
#report = classification_report(y_test, y_pred)
# Print the classification report
print("predicted output:\n", y_pred)
argmax_indices_pred = np.argmax(y_pred, axis=1)
argmax_indices_test = np.argmax(y_test, axis=1)
print("max_ pred output:\n", argmax_indices_pred)
print("y test:\n", argmax_indices_test)
report = classification_report(argmax_indices_pred, argmax_indices_test)
print("report of classification", report)
# Compute the confusion matrix
cm = confusion_matrix(argmax_indices_pred, argmax_indices_test)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)

# efficient Net implementation
img_rows = 224
img_cols = 224
base_model = tf.keras.applications.efficientnet.EfficientNetB3(include_top=
False, weights= "imagenet", input_shape = (img_rows,img_cols,3), pooling=
'max')
model = Sequential([
   base_model,
   BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
   Dense(256, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer=
regularizers.l1(0.006),
             bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
   Dropout(rate= 0.45, seed= 123),
   Dense(4, activation= 'softmax')
])

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

epochs = 20  # number of all epochs in training

history_en = model.fit( X_train,y_train, epochs= epochs, verbose= 1,
validation_data= (X_val,y_val),
                 validation_steps= None, shuffle= False)
from statistics import mean
acc_en = history_en.history['accuracy']
print("Training Accuracy : ",mean(acc_en))
val_acc_en = history_en.history['val_accuracy']
print("Validation Accuracy : ",mean(val_acc_en))
loss_en = history_en.history['loss']
val_loss_en = history_en.history['val_loss']

epochs = range(20)
plt.xlabel("Epochs")
plt.ylabel("Accuracies ")
#plt.plot(epochs, loss, 'r', label='loss')
plt.plot(epochs, val_acc_en, 'b', label='Validation accuracy')
plt.title('Validation accuracy')
plt.legend(loc=0)
# Export the plot to a PNG file
#plt.savefig("Accuracy_plot_effc.png",format="png")
plt.figure()

plt.show()
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(epochs, loss_en, 'r', label='loss')
#plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Loss')
plt.legend(loc=0)
# Export the plot to a PNG file
```

```
#plt.savefig("loss_plot_effc.png",format="png")
plt.figure()

plt.show()
#uncomment to save the model for front end;
#model.save('efficientNetmodel.h5')
img = load_img('/content/drive/MyDrive/ML_Dataset/Testing/glioma/Te-
gl_0273.jpg', target_size=(224,224))
plt.figure(figsize=(10, 10))
plt.subplot(1,3,1)
plt.imshow(img, cmap="gray")
plt.axis('off')
plt.show
i = img_to_array(img)
input_arr = np.array([i])
predict_x=model.predict(input_arr)
class_x=np.argmax(predict_x)
print(predict_x)
print(class_x)
if(class_x == 1):
   print("The prediction on MRI IMAGE is GLIOMA TUMOR")
elif(class_x == 2):
   print("The prediction on MRI IMAGE is MENINGIOMA TUMOR")
elif(class_x == 3):
   print("The prediction on MRI IMAGE is PITUITARY TUMOR")
else:
   print("The MRI Image is Of HEALTHY BRAIN")
#evaluation: accuracy, precision, f1-score, recall.
#print the sklearn classification report
# Make predictions on the test set
y_pred = model.predict(X_test)
#report = classification_report(y_test, y_pred)
# Print the classification report
print("predicted output:\n", y_pred)
argmax_indices_pred = np.argmax(y_pred, axis=1)
argmax_indices_test = np.argmax(y_test, axis=1)
print("max_ pred output:\n", argmax_indices_pred)
print("y test:\n", argmax_indices_test)
report_en = classification_report(argmax_indices_pred, argmax_indices_test)
print("report of classification", report_en)
# Compute the confusion matrix
cm_en = confusion_matrix(argmax_indices_pred, argmax_indices_test)

# Print the confusion matrix
```

```
print("Confusion Matrix:")
print(cm_en)

mlp_classifier = MLPClassifier(hidden_layer_sizes=(20,), max_iter=10,
solver='adam', random_state=42,)
# Train the classifier
mlp_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = mlp_classifier.predict(X_test)
#report = classification_report(y_test, y_pred)
# Print the classification report
print("predicted output:\n", y_pred)
argmax_indices_pred = np.argmax(y_pred, axis=1)
argmax_indices_test = np.argmax(y_test, axis=1)
print("max_ pred output:\n", argmax_indices_pred)
print("y test:\n", argmax_indices_test)
report_en = classification_report(argmax_indices_pred, argmax_indices_test)
print("report of classification", report_en)
# Compute the confusion matrix
cm_mlp = confusion_matrix(argmax_indices_pred, argmax_indices_test)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm_mlp)
```

Custom VGG-19 Implementation:

```
import tensorflow as tf
from zipfile import ZipFile
import os,glob
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import cv2
from tqdm.notebook import tqdm_notebook as tqdm
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Convolution2D, Dropout, Dense,MaxPooling2D
from keras.layers import BatchNormalization
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from tensorflow.keras import regularizers

import keras
from keras.preprocessing import image
from keras.utils import load_img,img_to_array
import matplotlib.pyplot as plt
from statistics import mean
from tensorflow import keras
from keras.applications.VGG-19 import VGG-19

from keras.optimizers import Adam, Adamax
from keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras


from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,
GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D

from keras.models import Model
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.neural_network import MLPClassifier

def notumor():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_train_val.append(img)
      y_train_val.append('0')
def glioma():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_train_val.append(img)
      y_train_val.append('1')
def meningioma():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_train_val.append(img)
      y_train_val.append('2')
def pituitary():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_train_val.append(img)
      y_train_val.append('3')
X_train_val = []

y_train_val = []

def notumor_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_test.append(img)
      y_test.append('0')
def glioma_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_test.append(img)
      y_test.append('1')
def meningioma_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_test.append(img)
      y_test.append('2')
def pituitary_test():
    for i in tqdm(os.listdir()):
      img = cv2.imread(i)
      img = cv2.resize(img,(224,224))
      X_test.append(img)
      y_test.append('3')
X_test = []
y_test = []

os.chdir('/content/drive/MyDrive/ML_Dataset/Training/glioma')
glioma()
os.chdir('/content/drive/MyDrive/ML_Dataset/Training/meningioma')
meningioma()
os.chdir('/content/drive/MyDrive/ML_Dataset/Training/pituitary')
pituitary()
os.chdir('/content/drive/MyDrive/ML_Dataset/Training/notumor')
notumor()
```

```
os.chdir('/content/drive/MyDrive/ML_Dataset/Testing/glioma')
glioma_test()
os.chdir('/content/drive/MyDrive/ML_Dataset/Testing/meningioma')
meningioma_test()
os.chdir('/content/drive/MyDrive/ML_Dataset/Testing/pituitary')
pituitary_test()
os.chdir('/content/drive/MyDrive/ML_Dataset/Testing/notumor')
notumor_test()

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=70)
# Assume X and y are your features and target variable, respectively
#X_temp, X_test, y_temp, y_test = train_test_split(X_train_val, y_train_val,
test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,
test_size=0.10, random_state=42)


print ("Shape of an image in X_train: ", X_train[0].shape)
print ("Shape of an image in X_test: ", X_test[0].shape)
print ("Shape of an image in X_val: ", X_val[0].shape)
le = preprocessing.LabelEncoder()
y_train = le.fit_transform(y_train)
y_val = le.fit_transform(y_val)
y_test = le.fit_transform(y_test)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_val = tf.keras.utils.to_categorical(y_val, num_classes=4)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=4)
y_train = np.array(y_train)
X_train = np.array(X_train)
y_test = np.array(y_test)
X_test = np.array(X_test)
y_val = np.array(y_val)
X_val = np.array(X_val)
print("X_train Shape: ", X_train.shape)
print("X_test Shape: ", X_test.shape)
print("y_train Shape: ", y_train.shape)
print("y_test Shape: ", y_test.shape)
print("X_val Shape: ", X_val.shape)
print("y_val Shape: ", y_val.shape)
#include validation set in the model.fit where 10% of training data

# Define the VGG-19 model
```

```
def VGG-19_model(input_shape=(224, 224, 3), num_classes=4):
    model = Sequential()

    # Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    # Fully connected layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(4096, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model

# Instantiate the VGG-19 model
```

```
model = VGG-19_model()

# Display the model summary
model.summary()

def lw(bottom_model, num_classes):
    top_model = bottom_model.output
    #top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(num_classes,activation='softmax')(top_model)
    return top_model

num_classes = 4
FC_Head = lw(model, num_classes)
model = Model(inputs = model.input, outputs = FC_Head)
print(model.summary())

model.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics =
['accuracy'])
history_vgg = model.fit(X_train,y_train,
                    epochs=10,
                    validation_data=(X_val,y_val))

acc_vgg = history_vgg.history['accuracy']
print("Training Accuracy : ",mean(acc_vgg))
val_acc_vgg = history_vgg.history['val_accuracy']
print("Validation Accuracy : ",mean(val_acc_vgg))
loss_vgg = history_vgg.history['loss']
val_loss_vgg = history_vgg.history['val_loss']

epochs = range(20)
plt.xlabel("Epochs")
plt.ylabel("Accuracies ")
#plt.plot(epochs, loss, 'r', label='loss')
plt.plot(epochs, val_acc_vgg, 'b', label='Validation accuracy')
plt.title('Validation accuracy')
plt.legend(loc=0)
# Export the plot to a PNG file
#plt.savefig("Accuracy_plot.png",format="png")
plt.figure()

plt.show()
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

#report = classification_report(y_test, y_pred)
# Print the classification report
print("predicted output:\n", y_pred)
argmax_indices_pred = np.argmax(y_pred, axis=1)
argmax_indices_test = np.argmax(y_test, axis=1)
print("max_ pred output:\n", argmax_indices_pred)
print("y test:\n", argmax_indices_test)
report = classification_report(argmax_indices_pred, argmax_indices_test)
print("report of classification", report)

# Compute the confusion matrix
cm = confusion_matrix(argmax_indices_pred, argmax_indices_test)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)
```

REFERENCES

[1] A. Jain, A. Fandango, and A. Kapoor. “TensorFlow Machine Learning Projects: Build 13 Real-World Projects with Advanced Numerical Computations Using the Python Ecosystem” Packt Publishing, U.K, 2018.

[2] A. Naseer, T. Yasir, A. Azhar, et al. “Computer-Aided Brain Tumor Diagnosis: Performance Evaluation of Deep Learner CNN Using Augmented Brain MRI.” International Journal of Biomedical Imaging, Jun 2021, pp. 1-11, DOI: 10.1155/2021/5513500

[3] A. Zisserman and S. Karen, "Very deep convolutional networks for large-scale image recognition." 2014, arXiv preprint, arXiv:1409.1556

[4] D. Beazley, “Python Cookbook”, O’Reilly Media Inc., CA, USA, 2013.

[5] D. Bhatt, C. Patel, H. Talsania, et al. "CNN variants for computer vision: History, architecture, application, challenges, and future scope." Electronics, vol 20, 2021, DOI: 10.3390/electronics10202470.

[6] F. Zulfiqar, U.I. Bajwa, Y. Mehmood. “Multi-class classification of brain tumor types from MR images using EfficientNets.” Biomedical Signal Processing and Control, vol 84, 2023, DOI: 10.1016/j.bspc.2023.104777.

[7] “Keras Implementation with Python”, Keras, Available: https://keras.io/api/ [Accessed: February 2024]

[8] “Matplotlib: Visualization with Python," Matplotlib, Available: https://matplotlib.org/, [Accessed: February 2024].

[9] Md. Saikat Islam Khan, A. Rahman, et al. “Accurate brain tumor detection using deep convolutional neural network” Computational and Structural Biotechnology , vol 20, 2022, pp. 4733-4745, DOI: 10.1016/j.csbj.2022.08.039

[10] M. Tan and Quoc V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”, 2019, International Conference on Machine Learning (PMLR), 2019, pp. 6105-6114, DOI: 10.48550/arXiv.1905.11946

[11] M. Nickparvar, “Brain Tumor MRI Dataset”, Kaggle.com, 2021, https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset (accessed Feb. 20, 2024).

[12] NA. Samee, NF. Mahmoud, G. Atteia, et al. “Classification Framework for Medical Diagnosis of Brain Tumor with an Effective Hybrid Transfer Learning Model.” Diagnostics (Basel) , vol 12(10), 2022, DOI: 10.3390/diagnostics12102541.

[13] “NumPy Documentation with Python”, NumPy, Available: https://numpy.org/doc/stable/user/index.html, [Accessed: February 2024].

[14] P. Gokila Brindha, M. Kavinraj & P. Manivasakam et al. “Brain tumor detection from MRI images using deep learning techniques”. At IOP Conference Series: Materials Science and Engineering, vol 1055 (1), 2021, pp. 173-180, DOI: 10.1088/1757-899X/1055/1/012115

[15] “Project Jupyter Documentation and lab”, Jupyter Notebook, Available: https://jupyterlab.readthedocs.io/en/latest/ [Accessed: January 2024]

[16] S. Haykin, "Neural Networks and Learning Machines." Pearson Education, NJ, USA, 2009.

[17] S. Saeedi, S. Rezayi, H. Keshavarz et al. “MRI-based brain tumor detection using convolutional deep learning methods and chosen machine learning techniques”, BMC Medical Informatics and Decision Making, vol 23(1), 2023, p.16, DOI: 10.1186/s12911-023-02114-6

[18] “Stream lit library and usage in python”, Streamlit, Available: https://docs.streamlit.io/library/api-reference [Accessed: November 2023]

[19] T. Srikanth, "Transfer learning using vgg-16 with deep convolutional neural network for classifying images." International Journal of Scientific and Research Publications (IJSRP),vol 9(10), 2019, pp. 143-150, DOI: 10.29322/IJSRP.9.10.2019.p9420

[20] VH. Phung and R. Eun Joo, "A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets", Applied Sciences, vol 9 (21), 2019, p.4500, DOI: 10.3390/app9214500

[21] VM. Javier , M. Marc & S. Cha, “Current Clinical Brain Tumor Imaging,” Neurosurgery, vol 81(3), p. 397, DOI:10.1093/neuros/nyx103.