BRAIN TUMOR DETECTION & CLASSIFICATION

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

by

G.Shanmukha kumar (20BCI7309)
G.Keerthi vardhani(20BCE7620)
N.Sai swetha(20BCE7196)
B.Bhargav(20BCD7273)

Under the Guidance of

Dr. Gouranga Mandal

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

DECEMEBR 2023

CERTIFICATE

This is to certify that the Capstone Project work titled "**Brain Tumor Detection &Classification"** that is being submitted by **G. Shanmukha kumar(20BCI7309), G.Keerthi vardhani (20BCE7620), N.Sai swetha (20BCE7196), B.Bhargav (20BCD7273)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. Gouranga Mandal
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner 1 **Internal Examiner 2**

Approved by

HoD, Name of the Department

School of Computer Science and Engineering

ACKNOWLEDGEMENTS

We are extremely grateful to VIT-AP University for their leadership, continual oversight, and provision of the information required for the project as well as for their assistance in seeing it through to completion.

We would like to thank Dr.Gouranga Mandal, our project mentor, for his kind support and cooperation in helping us finish this project. It gives us great pleasure to express our gratitude to all of the faculty members of our institution for their timely encouragement and sincere passion, both of which have helped us acquire the necessary information to successfully complete our course of study.

We want to express our gratitude to our parents for their help. Last but not least, we want to thank and appreciate everyone who has contributed to the successful completion of this project, whether directly or indirectly.

ABSTRACT

The brain tumor is a significant health concern affecting a considerable number of individuals worldwide. It poses significant challenges to both patients and healthcare professionals. Tumor detection, diagnosis, and treatment planning are complex tasks that require accurate and efficient techniques for effective management. This can make it more difficult to identify small or subtle tumors accurately. Comparing the performance of these models to each other like VGG- 16, CNN, InceptionV3, and ResNet-50.

It is an important step in evaluating their effectiveness. CNN, InceptionV3, ResNet-50, and VGG-16 are well-known convolutional neural network (CNN) architectures. These CNN architectures have been extensively studied, fine-tuned, and pre-trained on large-scale datasets like ImageNet, enabling them to learn generic features that can be leveraged for various computer vision tasks, including image classification this approach shows promise in overcoming the limitations of low-resolution MR images for accurate brain tumor classification, potentially improving diagnostic accuracy with data segmentation.

Keywords: Image classification, Image recognition, MRI, Neural networks, ResNet-50, InceptionV3, VGG-16, Convolution Neural Network.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

1.INTODUCTION

The presence of abnormal and uncontrolled synapses, which refers to the communication between neurons in the brain, is not a defining characteristic of brain tumors. Brain tumors are primarily characterized by the uncontrolled growth of abnormal cells within the brain tissue[1].Abnormal development growth in the brain is called brain tumor, and their classification is an essential aspect of diagnosis and treatment planning. The two broad categories used to classify brain tumors are malignant and benign[2]. Non-cancerous growth in the brain is called benign tumors in which sur- rounding tissues are not invaded or spread from one part to another part in the body. They usually arise from brain structures such as meninges, pituitary glands, or cranial nerves[3]. Meningiomas, which originate from the meninges (the protective layers covering the brain and spinal cord), are a common type of benign brain tumor. Although they are considered benign, some benign tumors can still pose serious health risks, especially if they grow in critical areas of the brain or exert pressure on vital structures. In rare cases, benign tumors may also transform into malignant tumors over time[4]. One advantage of benign tumors is that they tend to have well-defined borders and limited invasion into the surrounding brain tissue. These characteristics make them more amenable to surgical removal, and in many cases, complete resection can lead to a cure or long-term remission. However, in rare cases, pituitary tumors can be malignant (cancerous) or have the potential to recur or transform into malignancy[5]. Brain tumors can indeed exert pressure on the brain and lead to various health issues, depending on their location and size. Detecting brain tumors early is crucial to initiate appropriate treatment strategies and increase the likelihood of patient survival. MRI is widely utilized in biomedical imaging research and clinical practice for detailed visualization of brain tumors. It provides high-resolution images that can reveal the location, size, and characteristics of tumors, assisting in their diagnosis and classification[6]. According to data from the National Brain Tumor Foundation(NBTF), the mortality rate associated with brain tumors has recently increase by 200 in many countries[7][8].

However, some challenges can affect the quality and accuracy of MR images, as you

mentioned. Advancements in MRI technology have led to improved image quality, and higher-field MRI scanners can provide clearer and more detailed images[9]. Additionally, various image processing techniques and algorithms, including machine learning, deep learning, and artificial intelligence approaches, are used to detect and classify brain tumors using MRI data. MRI provides detailed images of the brain, allowing doctors to identify the presence, location, and characteristics of tumors.

However, there are indeed challenges that can affect the accuracy of tumor detection and diagnosis. One of the challenges is the quality of MRI devices. The quality of the equipment can vary, and lower-quality machines may produce images with reduced resolution and clarity[9]. By improving the image quality, developing advanced image analysis methods, and integrating AI techniques, brain tumor detection, and classification accuracy can be enhanced, leading to improved patient outcomes and increased survival rates.

1.1RELATED WORK

During the analysis of brain tumor detection and classification we have used, researchers have utilized various deep-learning techniques to automate segmentation and detection [17]. A notable approach proposed by Yanming Sun et al. [11] involves the use of Convolutional Neural Networks (CNN) for reliable and computationally efficient brain tumor segmentation, which yielded promising results.

Parra et al. [12] implemented an artificial neural network (ANN) algorithm for MRI brain image segmentation. Their approach utilized multi-spectral features from MR images, including T1-weighted, T2-weighted, and proton density (P.D.), to segment different brain tissues. The proposed ANN algorithm incorporated a learning vector quantization (LVQ) network.

Havaei et al. [13] proposed a deep learning and machine learning-based method for tumor classification of the brain using a convolutional neural network. Their approach achieved high accuracy in distinguishing between different tumor types based on MRI images.

Pereira et al. [14] employed a machine-learning framework for brain tumor classification. They extracted various radio mic features from MRI images and used support vector machines (SVM) to classify tumors into different subtypes.

Valverde et al. [15] employed a machine learning and deep learning model for glioma and its subtypes classification based on MRI images. They utilized a combination of RNN and Convolutional neural networks to capture both spatial and temporal information from multi-sequence MRI scans.

S. Deepak [16] proposed a deep learning model using transfer learning methodology where it classifies the various brain tumors based on the MRI images. They used various convolutional neural networks to differentiate various tumors that are glioma, meningioma, and pituitary.

Used Technique	Author	Dataset	Accuracy	Advantages	Disadvantages
Multi-classification model	I. Charfi , M.Atri	Three Different publicly datasets	90.27%	Low computational time	Classification accuracy need to be increased
Hybrid deep learning based	Kadry, Rauf	ISLE2015 and BRAT 2015	96%	perform multi-class classification for brain tumor	Classification accuracy need to be increased
Kernel-based SVM	Pareek, Mukherjee	Fig share	97%	Can detect whether brain tumor is benign and maligent	Small dataset. Classification accuracy need to be increased
Deep learning-based automatic multimodal classification	Khan, Ashraf	Bra Ts 2015,BraTs 2017,BraTs 2018	97.8%	Feature extraction improved classification accuracy and reduced processing time	Classification accuracy need to be increased
SVM and k-NN classification	Urhan	Figshare,2017	97.25%	Extended ROI's shallow and deep properties improves classification performance	Accuracy need to be improved

Fig. 1 Literature Survey

2. METHODOLOGY

The paper presents a novel approach called InceptionV3, ResNet-50, VGG-16, and CNN for brain tumor classification using deep learning techniques. The initial goal is to decrease the mortality rate of brain tumors and enhance human longevity. The proposed methodology consists of six stages, the proposed methodology aims to environmental setup, collect the data, segmentation, pre-process, classify, and evaluate the performance of a brain tumor classification system. The ultimate aim is to develop a reliable and accurate system that can assist in early detection and improve the treatment outcomes for brain tumor patients.

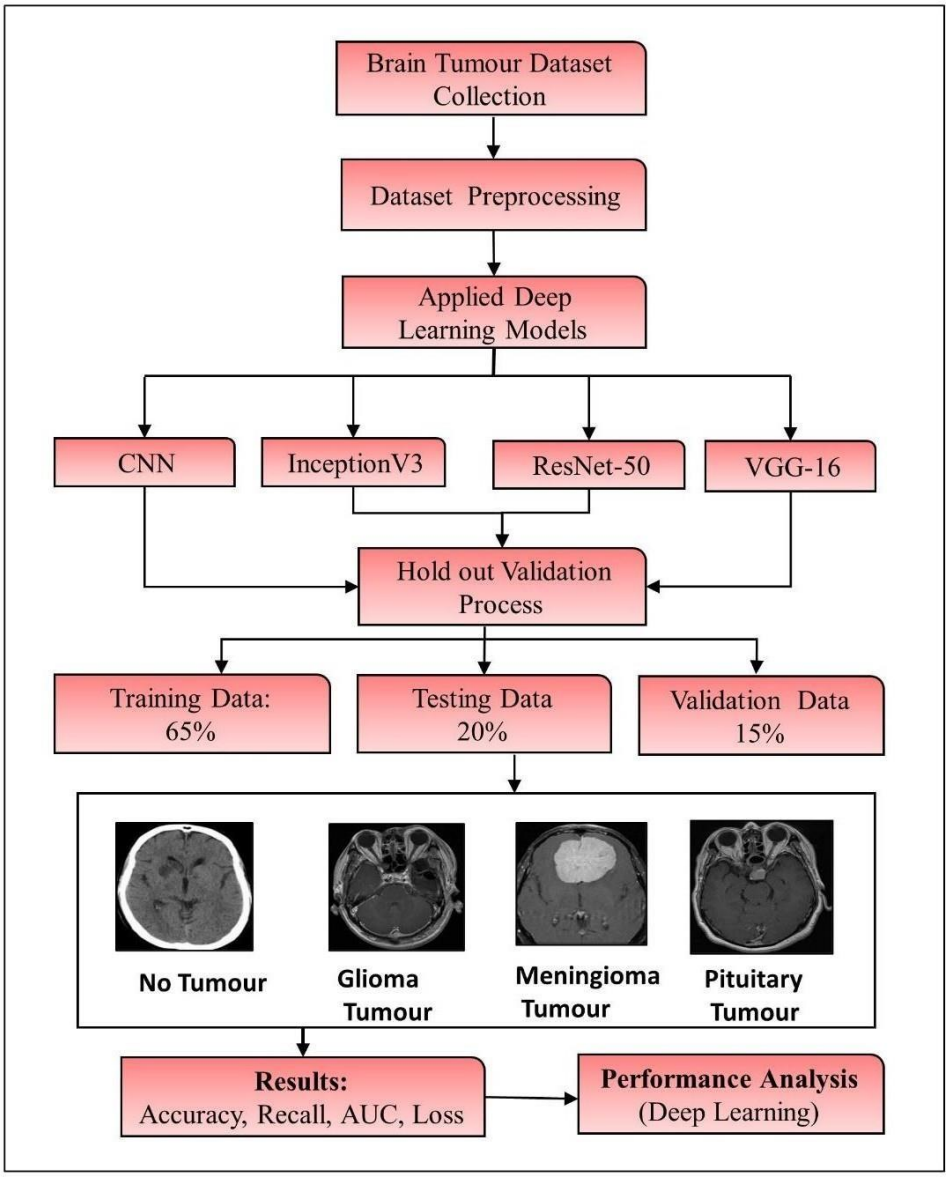


Fig. 2 Complete Workflow

2.1 Environmental Setup:

By leveraging the resources and capabilities offered by Google Colab, the researchers can efficiently train their Deep learning model for brain tumor classification. The platform's computational power and memory capacity enhance the training pro- cess, allowing for faster iterations and potentially improving the model's performance[18].TensorFlow environment for brain tumor classification, you will need to install TensorFlow and ensure that your system meets the requirements, libraries, and modules.

2.2 Dataset Collection:

MRI is a widely used and effective imaging technique for brain tumor detection. The dataset for brain tumor detection that we constructed from publicly accessible data is comprised of magnetic resonance imaging (MRI) images obtained from Kaggle. MRI Dataset of Brain Tumor is a group of four label MR images(Magnetic Resonance Imaging) specifically designed for research and development purposes in the field of brain tumor detection and classification. This dataset plays a significant role in training and evaluating machine learning and deep learning models for brain tumor analysis. The dataset may include images from patients with four different types of brain tumors, they are gliomas, meningiomas, pituitary tumors, and no tumors. It may also contain images from different grades or stages of tumors, ranging from low-grade to high-grade tumors.

Researchers and developers use the Brain Tumor MRI Dataset to train and test their algorithms and models for tasks such as tumor segmentation, classification, and detection. These tasks aim to accurately identify the type, location, and characteristics of tumors in MRI. Access to the Brain Tumor MRI Dataset can typically be obtained by requesting access from the dataset creators or through publicly available repositories or platforms that host medical imaging datasets. Researchers should ensure compliance with data usage agreements and ethical considerations when working with medical imaging datasets.

The dataset includes four different types of brain tumor data:

1. Meningioma: This category contains 1339 MRI images of meningioma tumors depicting patients. Most of the slowly growing tumors are Meningiomas and mostly developed in the meninges, which are the protective membranes around the brain and spinal cord[20].

2. No tumor: The dataset includes 2096 MRI images that do not exhibit any brain tumors. This category serves as a reference for healthy brain images or images without any detectable tumors.

3. Pituitary tumor: This category comprises 1457 MRI images representing patients with pituitary tumors, that developed in the small gland that is located at the base of the brain in the pituitary gland[10].

4. Glioma tumor: The dataset incorporates 1321 MRI images showing patients with glioma tumors. Gliomas are a type of brain tumor that originates from glial cells.

Table 1 Dataset Details

Type	Original Dataset	Training Dataset	Testing Dataset
Glioma Tumor	1321	826	100
Meningioma Tumor	1339	822	115
Pitutary Tumor	1457	827	74
No Tumor	2096	395	105

Overall, your dataset consists of 7023 MRI images, covering the four afore mentioned types of brain tumor data. This diverse collection of images can provide a valuable resource for training and evaluating brain tumor detection models using deep learning or other machine learning techniques [23].

2.3 Segmentation

The segmentation stage is crucial for separating the images into meaningful regions. Manual segmentation of a large number of medical images is impractical, so automated segmentation techniques are employed. This technique helps identify relevant regions of interest within the image. Steps involved in segmentation:

2.3.1 Threshold the image:

Thresholding the image is based on the particular threshold value where it is used to convert the image into a binary image. This can be done using techniques like Otsu's thresholding or adaptive thresholding.

2.3.2 Perform a series of erosions and dilations:

Erosion is a morphological operation that shrinks the white regions in the image, while dilation expands them. By applying a series of erosions followed by dilations, small regions of noise can be eliminated while preserving the larger connected regions.

2.3.3 Find contours in the thresholder image:

Contours are the boundaries of the connected components in the binary image. They can be found using contour detection algorithms such as the chain approximation algorithm.

2.3.4 Identify the largest contour:

Iterate through all the contours and find the contour with the largest area. This can be done by calculating the area of each contour using the 'cv2.contourArea()' function.

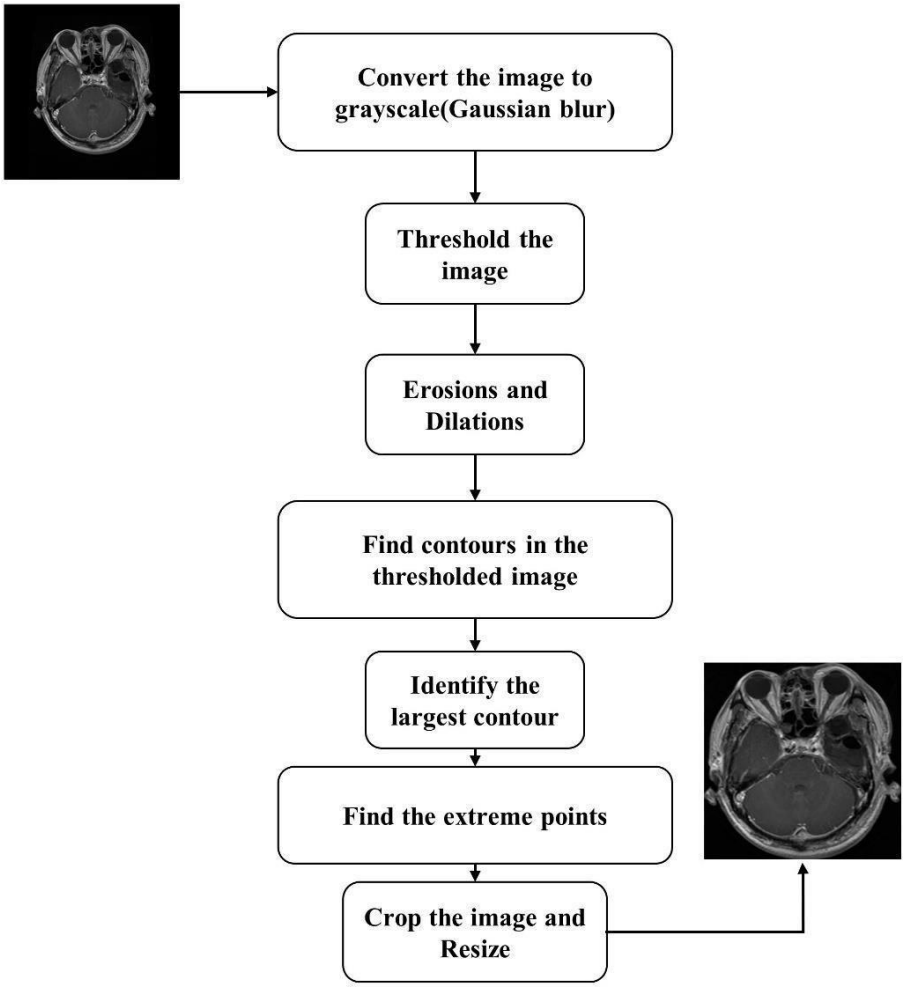


Fig. 3 Segmentation

2.3.5 Find the extreme points:

Once the largest contour is identified, you can find its extreme points. This can be done by finding the minimum and maximum x and y coordinates among all the points in the contour. These extreme points will define the rectangular bounding box around the region of interest.

2.3.6 Crop the image:

Finally, you can use the extreme points to crop the rectangular region of interest from the original image.

2.4 Pre-Processing

Pre-processing the image is one of the crucial steps in the classification of brain tumors. Pre-processing is mainly used for improving model performance.

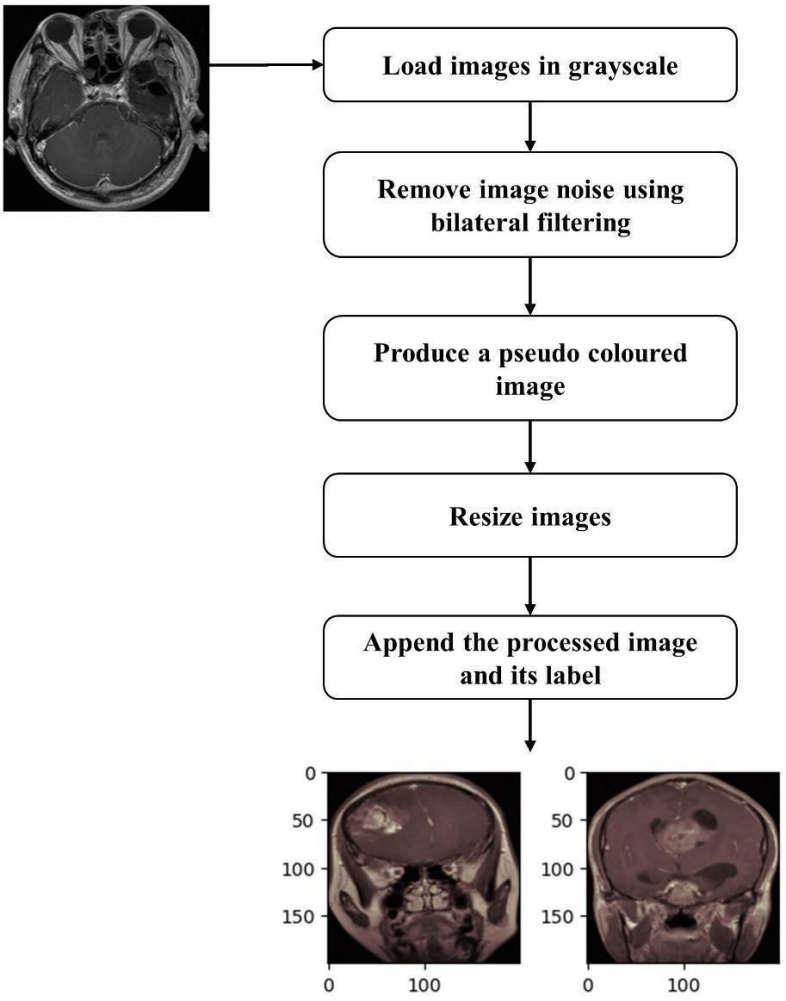


Fig. 4 Pre-processing

There are various steps involved in pre-processing:

2.4.1 Resizing the Image

Resizing the input images to a consistent size is important to ensure compatibility with the pre-trained model. Most pre-trained models have specific input size requirements. Here in resizing images firstly, loading the images in Gray colour then removing images noise. After removing noise it produces a pseudo coloured image. Finally, resizing the image to 150*150.

2.4.2 Normalization

Normalizing the input image is one of the steps in pre-processing. Normalization means scaling the image values into the range from 0 to 1. Normalization is used in data having a similar distribution on pre-trained models which reduces the impact of vari- ations in pixel intensity across various images. Here, our model normalized the images of training and testing data into the range of 0 and 1.

2.4.3 Data Augmentation

Data Augmentation is used to enlarge the size of the training dataset. Here, we used techniques like width shift, horizontal flip, rotation range, and height shift that can be used to create additional training samples, reducing overfitting and improving the model's generalization ability. Data augmentation should be applied consistently during both the training and evaluation phases.

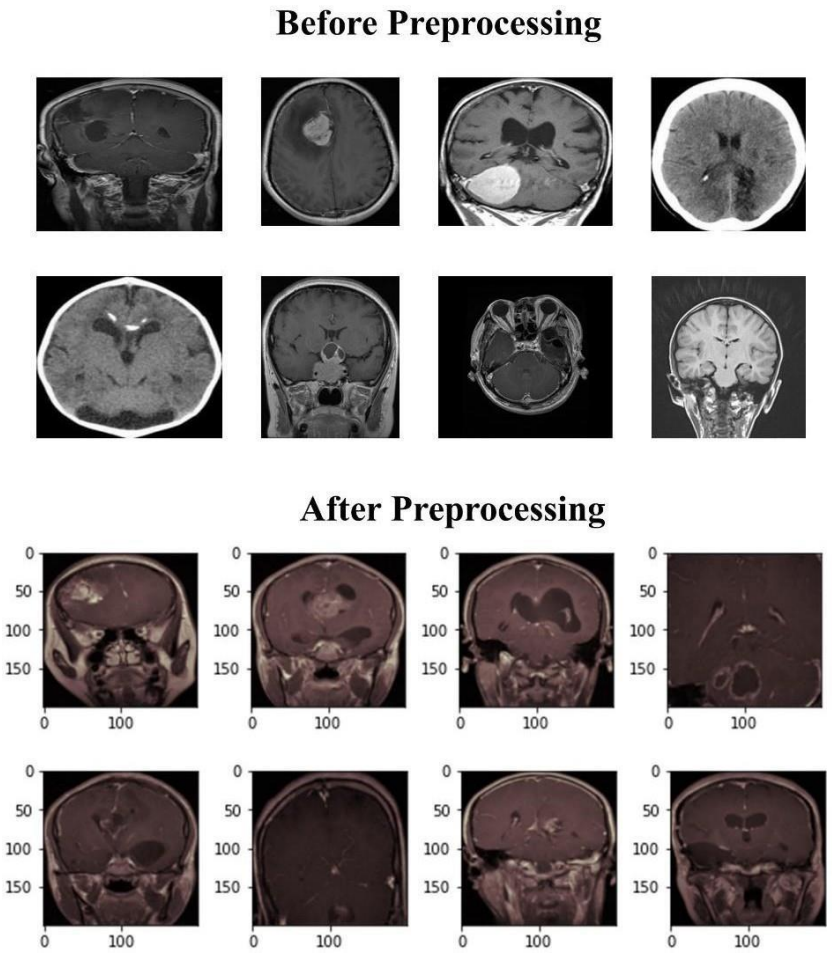


Fig. 5 Before and After Pre-processing

2.4.4 Pre-processing for specific models

Different pre-trained models may have specific pre-processing requirements. For example, some models expect images in a specific colour format (e.g., RGB or BGR), or they may require additional pre-processing steps such as mean subtraction or channel-wise normalization. It is important to consult the documentation of the specific pre-trained model being used to ensure proper pre-processing.

After the above operation, the feature extraction stage is performed using the GLCM which is called the Gray level occurrence matrix. The GLCM captures the spatial relationships between pixels and extracts texture features that are relevant for tumor classification[11].

2.5 Transfer Learning Models

Transfer learning is a widely used technique in machine learning, particularly in the context of neural networks. It involves utilizing pre-trained models and adapting them for new tasks. In the field of image classification and detection, various popular models are employed based on transfer learning, they are ResNet-50, CNN, VGG16, and InceptionV3. This approach offers significant use in terms of profitability and reduces training time and knowledge transfer[25]. Rather than starting from scratch, which requires substantial computational resources and extensive image databases, using pre- trained models as a starting point transfer learning enables the network to leverage the knowledge and generalization capabilities captured by the pre-training process.

Transfer learning involves utilizing pre-trained models to enhance the performance of a new classification task. Here's how it can be implemented:

2.5.1 Loading transfer learning

Architecture that we have chosen for our model, such as ResNet50, VGG-16, InceptionV3, or Sequential, is instantiated. The parameter called "weights" is used to set "ImageNet" taken from the dataset ImageNet. Additionally, the 'include top' parameter is set to 'False' to exclude the top classifier layer of the pre-trained model.

2.5.2 CNN

The CNN architecture employed in our study is designed to capture intricate features from brain images for accurate tumor classification. The architecture comprises several layers, each serving a specific purpose. The initial layers consist of convolutional filters that scan the input images to detect spatial patterns and features. Non-linear activation functions, such as ReLU (Rectified Linear Unit), are applied to introduce non-linearity and improve the network's ability to learn complex relationship. Max pooling layers follow convolutional layers to down sample feature maps, reducing computational complexity while retaining essential information. The final layers are fully connected, allowing the network to make predictions based on the learned features.

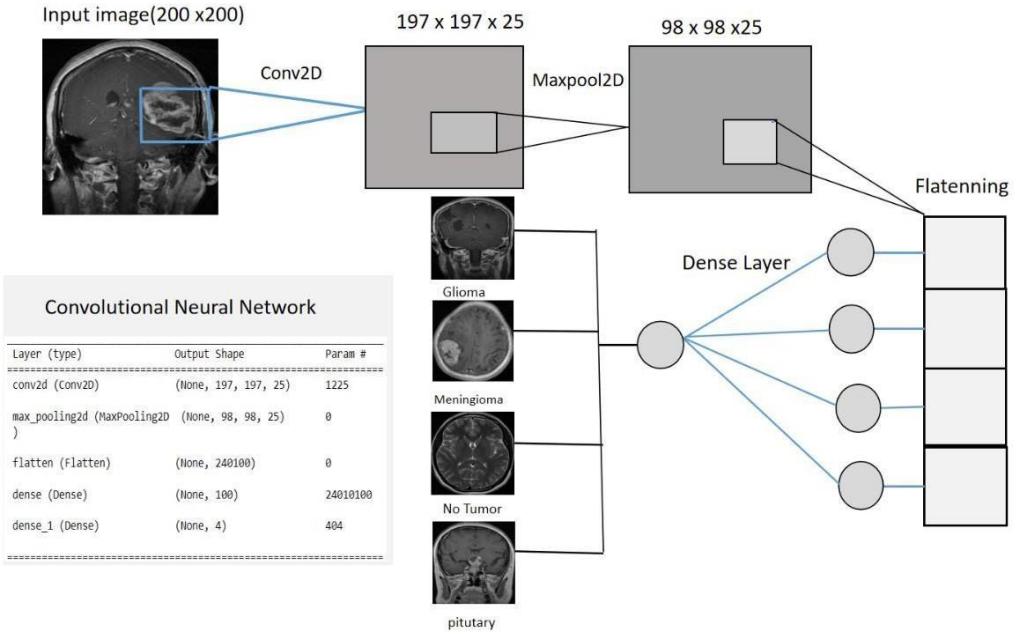


Fig. 6 CNN Architecture

2.5.3 VGG-16

The VGG16 architecture is characterized by its deep structure, consisting of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers are configured with small receptive fields (3x3) and are followed by max-pooling layers for spatial down sampling. This architecture's depth and relatively simple convolutional kernel sizes contribute to its effectiveness in capturing intricate features from images. The architecture can be summarized as the input layer Accepts input images of size 200 x 200 x 3. It consist of multiple convolutional layers followed by max-pooling layers. These blocks contribute to the hierarchical feature learning process. Follow the convolutional blocks, enabling the network to make predictions based on the learned features.

2.5.4 InceptionV3

InceptionV3 is characterized by its inception modules, which allow the model to simultaneously capture features at different scales. The architecture comprises stem layers, multiple inception blocks, and fully connected layers. Initial layers that process input images and extract foundational features. Modules with parallel convolutional operations of different kernel sizes, allowing the model to capture both local and global features efficiently. Follows the inception blocks and enables the model to make predictions based on the learned features.

2.5.5 Resnet-50

ResNet50 is characterized by its residual blocks, where the output from one layer is added to the input of a subsequent layer. This enables the model to learn residual functions, facilitating the training of very deep networks. The architecture comprises stem layers, multiple residual blocks, and fully connected layers. Initial layers that process input images and extract foundational features. Modules with shortcut connections that allow the model to learn residual functions, mitigating the vanishing gradient problem. Follows the residual blocks and enables the model to make predictions based on the learned features.

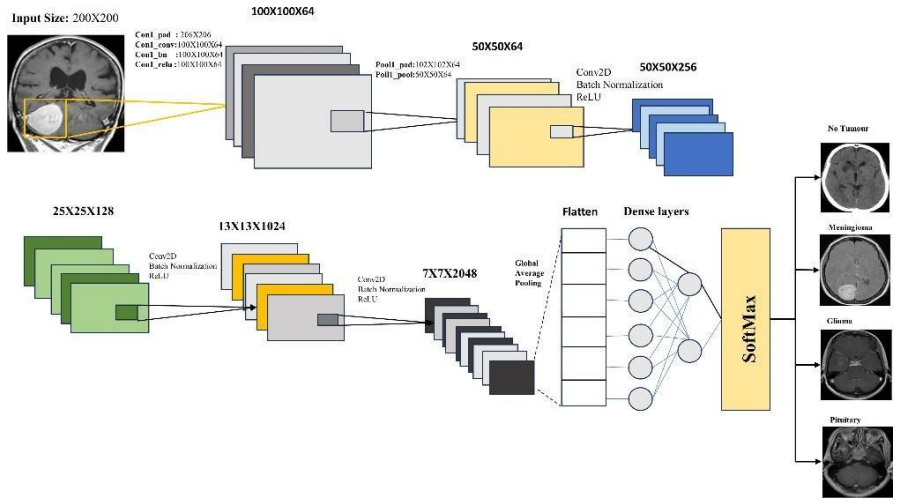


Fig. 7 Resnet50 Architecture

2.5.6 Building Model Architecture

To customize the model for the new classification task, additional layers are added. A 'GlobalAveragePooling2D' layer is included to perform global average pooling over the spatial dimensions of the tensor. This reduces the spatial dimensions while retaining important features. A 'Dropout' layer is added to randomly deactivate a portion of the input units during training, mitigating overfitting. The dropout rate is set to 0.4 (40). Lastly, a 'Dense' layer with the desired number of units and SoftMax activation is appended as the final classification layer. This layer outputs probabilities for each of the classes in the new task. The model's input and output tensors are defined using the 'Model' function.

2.5.7 Compiling the model

An optimizer, such as 'Adam', is instantiated with a specific learning rate. The 'compile' method is then called on the model, specifying the optimizer, the loss function (e.g., 'categorical cross entropy'), and metrics to track during training (e.g., accuracy).

2.5.8 Summarizing the model

To provide an overview of the model's architecture, including the layers, output shapes, and trainable parameters, is called the 'summary' method This transfer learning pipeline utilizes a pre-trained model (e.g., ResNet50, VGG-16, InceptionV3, Sequen- tial), adds additional layers

for customization, and compiles the model for training on a new classification task with a specific number of classes.

2.6 Performance Metrics

Performance metrics in deep learning are used to evaluate and assess the effective- ness of machine learning models and algorithms. These metrics help quantify the model's accuracy, generalization capability, and efficiency. Here are some commonly used performance metrics in deep learning:

2.6.1 Accuracy

Accuracy measures the proportion of correctly classified instances in a dataset. It is a fundamental metric for classification tasks and is calculated by dividing the number of correct predictions by the total number of predictions.

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

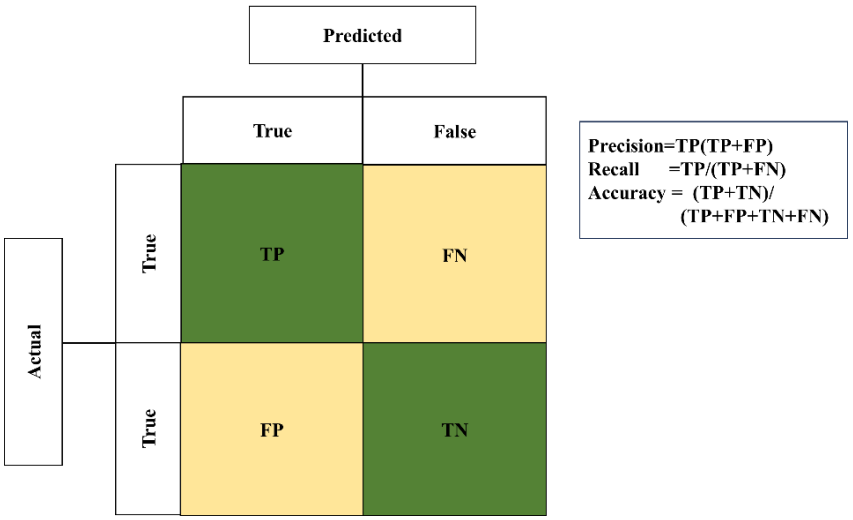


Fig. 8 Metrics

2.6.2 Precision

These metrics are commonly used for binary classification tasks. Precision measures the proportion of true positive predictions out of the total predicted positives.

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

2.6.3 Recall

Recall, also known as sensitivity, measures the proportion of true positive predictions out of the total actual positives.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

2.6.4 F1 Score

F1 score combines precision and recall to provide a single metric that balances both measures.

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

CHAPTER 3

RESULTS AND DISSCION

The projects experimental phase involved the analysis of various performance metrics including precision, recall, F1 score and accuracy to evaluate the various deep learning models such as Resnet50,VGG-16,InceptionV3,CNN.In brain tumor detection and classification various models were compared to obtain higher accuracy and most efficient prediction. Out of these four models Resnet50 got highest training accuracy of 99% and prediction accuracy was 97.71%.Here,compared to Resnet50 the other models i.e VGG16,CNN,InceptionV3 got less accuracy. So, we have chosen the ResNet-50 model for deployment choosing as it has obtained the highest accuracy compared to other models.

The results are presented in detail with figures and labels illustrating the performance analysis of deep learning models.

3.1 Accuracy Curve

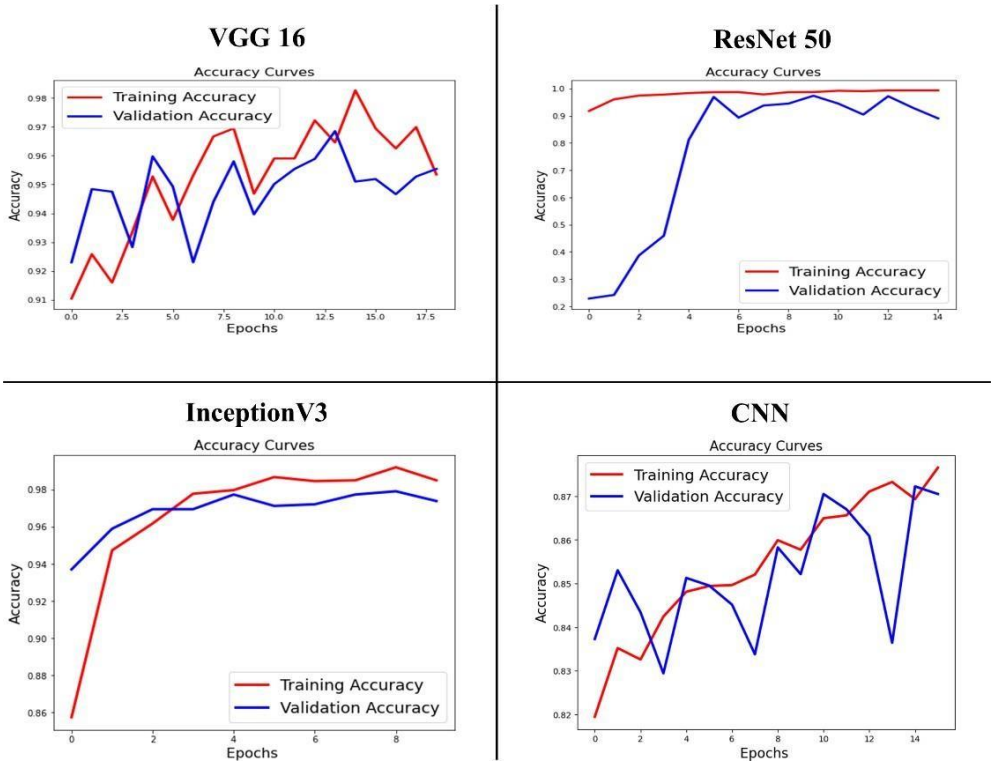


Fig. 9 Accuracy Curves for models Resnet50, CNN, VGG-16, InceptionV3.

The above figure shows performance analysis of accuracy of various models. In graphs it is very clear that Resnet50 Training and Validation accuracy of 99% and 97.71% respectively which is very high compared to the accuracy given by other models.

3.2 Loss Curve

The figure attached below shows the performance analysis of various models loss. In graphs it is very clear that Resnet50 is having lowest Training and Validation accuracy compared to other models which is of 0.017% and 0.13% respectively.

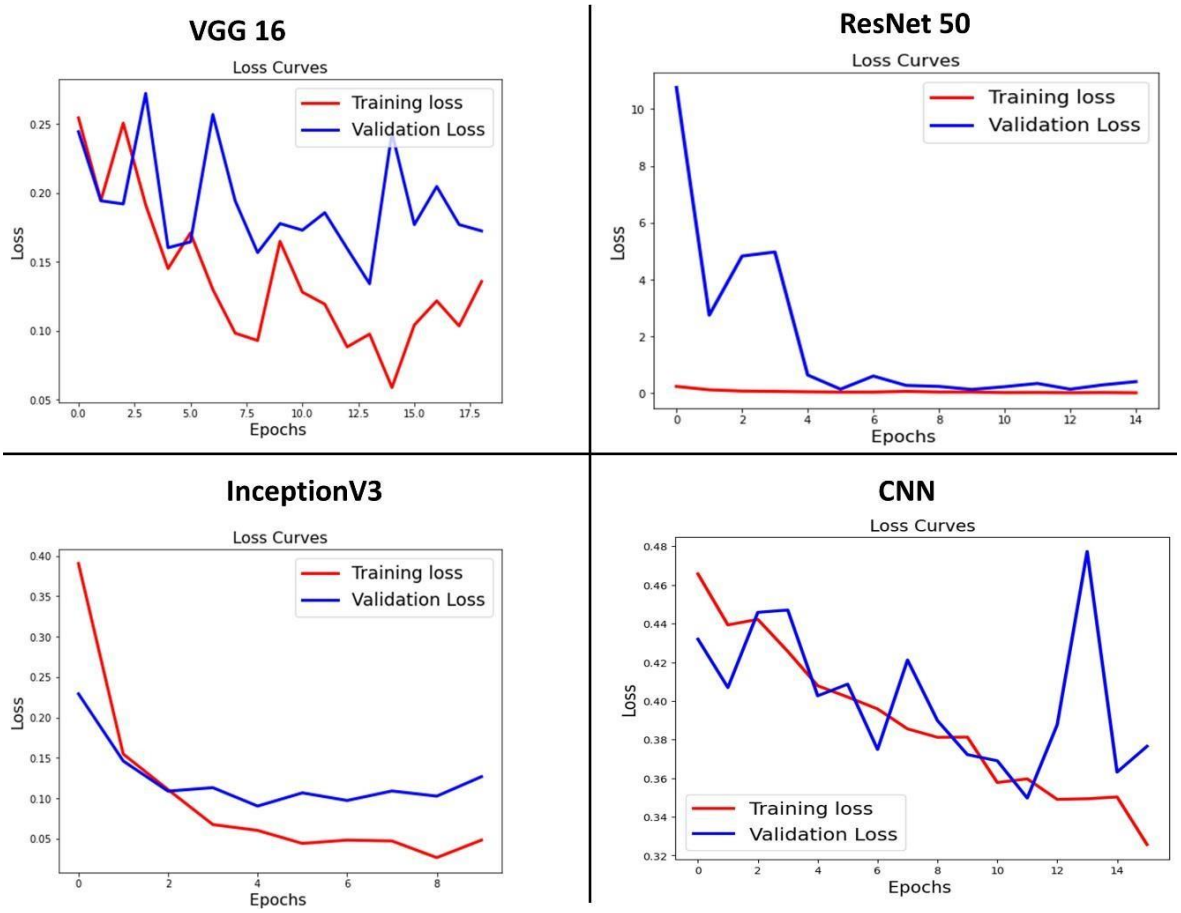
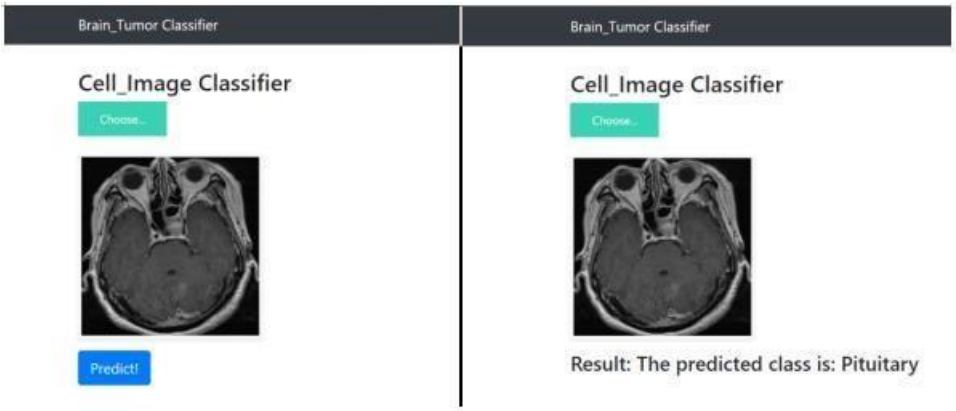


Fig. 10 Loss Curves for models Resnet50, CNN, VGG-16, and InceptionV3

3.3 Deployment

Flask framework is used for the deployment.

Fig. 11 Before and After Prediction



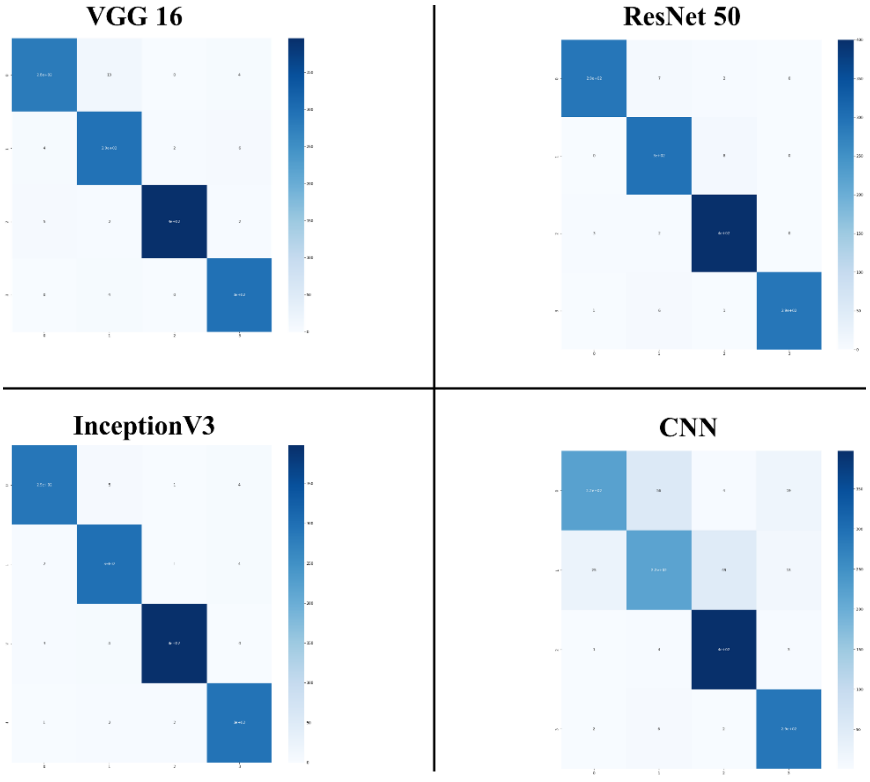


Fig.12 Heat Map Comparison

3.4 Performance Metrics

The performance metrics for the Resnet50, VGG16, InceptionV3, and CNN.

Table 2 Performance Metrics for Resnet-50

Model	Percision	Recall	F1-Score	Support
0	0.99	0.97	0.98	300
1	0.95	0.97	0.96	306
2	0.97	0.99	0.98	405
3	1.00	0.97	0.99	300
Accuracy			0.98	1311
Macro avg	0.98	0.98	0.98	1311
weighted avg	0.98	0.98	0.98	1311

Table 3 Performance Metrics for VGG-16

Model	Percision	Recall	F1-Score	Support
0	0.97	0.94	0.96	300
1	0.94	0.97	0.95	306
2	0.99	0.99	0.99	405
3	0.96	0.99	0.97	300
Accuracy			0.97	1311
Macro avg	0.97	0.97	0.97	1311
weighted avg	0.97	0.97	0.97	1311

Table 4 Performance Metrics for InceptionV3

Model	Percision	Recall	F1-Score	Support
0	0.98	0.97	0.97	300
1	0.96	0.98	0.97	306
2	0.99	0.98	0.98	405
3	0.97	0.98	0.98	300
Accuracy			0.98	1311
Macro avg	0.98	0.98	0.98	1311
weighted avg	0.98	0.98	0.98	1311

Table 5 Performance Metrics for CNN

Model	Percision	Recall	F1-Score	Support
0	0.89	0.74	0.81	300
1	0.77	0.72	0.74	306
2	0.88	0.98	0.93	405
3	0.89	0.97	0.93	300
Accuracy			0.86	1311
Macro avg	0.86	0.85	0.85	1311
weighted avg	0.86	0.86	0.86	1311

CHAPTER 4

CONCLUSION

The intended task has been successfully carried out using emerging Deep Learning techniques for brain tumor detection and classification and satisfying results have been achieved. The algorithms such as CNN,ResNet-50,VGG-16 and Inception v3 have been used for brain tumor classification and ResNet-50 has yielded good results.

The FLASK framework has been used for deployment smooth integration of the web pages with the built models to build a user-friendly web application and the user can have a comfortable and smooth experience for carrying out intended predictions.

This project can be further developed in order to achieve even higher accuracies by implementing more advanced learning models . Performing data segmentation over the dataset and input images will also help in creating more enhanced versions of the images which again contribute to slightly better results and accuracies.

Table 6 Results

Model	Training Accuracy	Testing Accuracy	Validation Accuracy
Resnet-50	99.34%	89.06%	97.71%
VGG-16	95.35%	95.5%	96.80%
InceptionV3	98.49%	97.38%	97.79%
CNN	87.66%	87.05%	85.96%

CHAPTER 5

REFERENCE

[1] Preethi, S., Aishwarya, P.: An efficient wavelet-based image fusion for brain tumor detection and segmentation over pet and mri image. Multimedia Tools and Applications 80(10), 14789–14806 (2021)

[2] Irmak, E.: Multi-classification of brain tumor mri images using deep convolutional neural network with fully optimized framework. Iranian Journal of Science and Technology, Transactions of Electrical Engineering 45(3), 1015–1036 (2021)

[3] Nayak, B., Dash, G.P., Ojha, R.K., Mishra, S.K.: Application of machine learning techniques for detection and segmentation of brain tumors. SN Computer Science 4(5), 520 (2023)

[4] Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer, Boston (1992)

[5] Broy, M.: Software engineering—from auxiliary to key technologies. In: Broy, M., Denert, E. (eds.) Software Pioneers, pp. 10–13. Springer, New York (1992)

[6] Seymour, R.S. (ed.): Conductive Polymers. Plenum, New York (1981)

[7] Smith, S.E.: Neuromuscular blocking drugs in man. In: Zaimis, E. (ed.) Neuromuscular Junction. Handbook of Experimental Pharmacology, vol. 42, pp. 593–660. Springer, Heidelberg (1976)

[8] Ghamry, F.M., Emara, H.M., Hagag, A., El-Shafai, W., El-Banby, G.M., Dessouky, M.I., El-Fishawy, A.S., El-Hag, N.A., El-Samie, F.E.A.: Efficient algorithms for compression and classification of brain tumor images. Springer (2023)

[9] Deshpande, A., Estrela, V.V., Patavardhan, P.: The DCT-CNN-ResNet50 archi- tecture to classify brain tumors with super-resolution, convolutional neural network, and the ResNet50. Elsevier (2021)

[10] Mukada, N., Tosaka, M., Yamaguchi, R., Tanaka, Y., Takahashi, A., Shimauchi-Otaki, H., Osawa, S., Tsushima, Y., Yoshimoto, Y.: Preoperative magnetic resonance imaging localization of the normal pituitary gland in nonfunctioning pituitary adenoma patients using the radiological sign of “internal carotid artery notch”. World Neurosurgery 166, 177–188 (2022)

[11] Arya, K., Rajawat, A., Pandey, M.K., Rajput, S.S.: Very low resolution face recognition using fused visual and texture features. In: 2017 Conference on Information and Communication Technology (CICT), pp. 1–5 (2017). IEEE

CHAPTER

6

APPENDIX

Python Code for ResNet50:

```
from keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint,TensorBoard,
from keras.layers import Input,Dropout, Dense,GlobalAveragePooling2D
from keras.models import Sequential,Model
from keras.applications.resnet import ResNet50
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.utils import shuffle import matplotlib.pyplot as plt
from tensorflow import keras
import tensorflow as tf
from tqdm import tqdm
import seaborn as sns
import numpy as np
import itertools
import datetime
import cv2
import os
import io

labels = ['glioma', 'meningioma', 'notumor', 'pituitary']
x_train = [] # training images.
y_train = [] # training labels.
x_test = [] # testing images.
y_test = [] # testing labels.
image_size = 200 for label in labels: trainPath = os.path.join('Training',label) for file in
tqdm(os.listdir(trainPath)): image = cv2.imread(os.path.join(trainPath, file),0) # load images in gray.
image = cv2.bilateralFilter(image, 2, 50, 50)
# remove images noise. image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
 # produce a pseudocolored im image = cv2.resize(image, (image_size, image_size))
# resize images into 150*150
x_train.append(image)
y_train.append(labels.index(label))
testPath = os.path.join('Testing',label)
for file in tqdm(os.listdir(testPath)):
image = cv2.imread(os.path.join(testPath, file),0)
image = cv2.bilateralFilter(image, 2, 50, 50)
image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
image = cv2.resize(image, (image_size, image_size)) x_test.append(image)
y_test.append(labels.index(label)) x_train = np.array(x_train) / 255.0
# normalize Images into range 0 to 1. x_test = np.array(x_test) / 255.0
 print(x_train.shape)
print(x_test.shape)
```

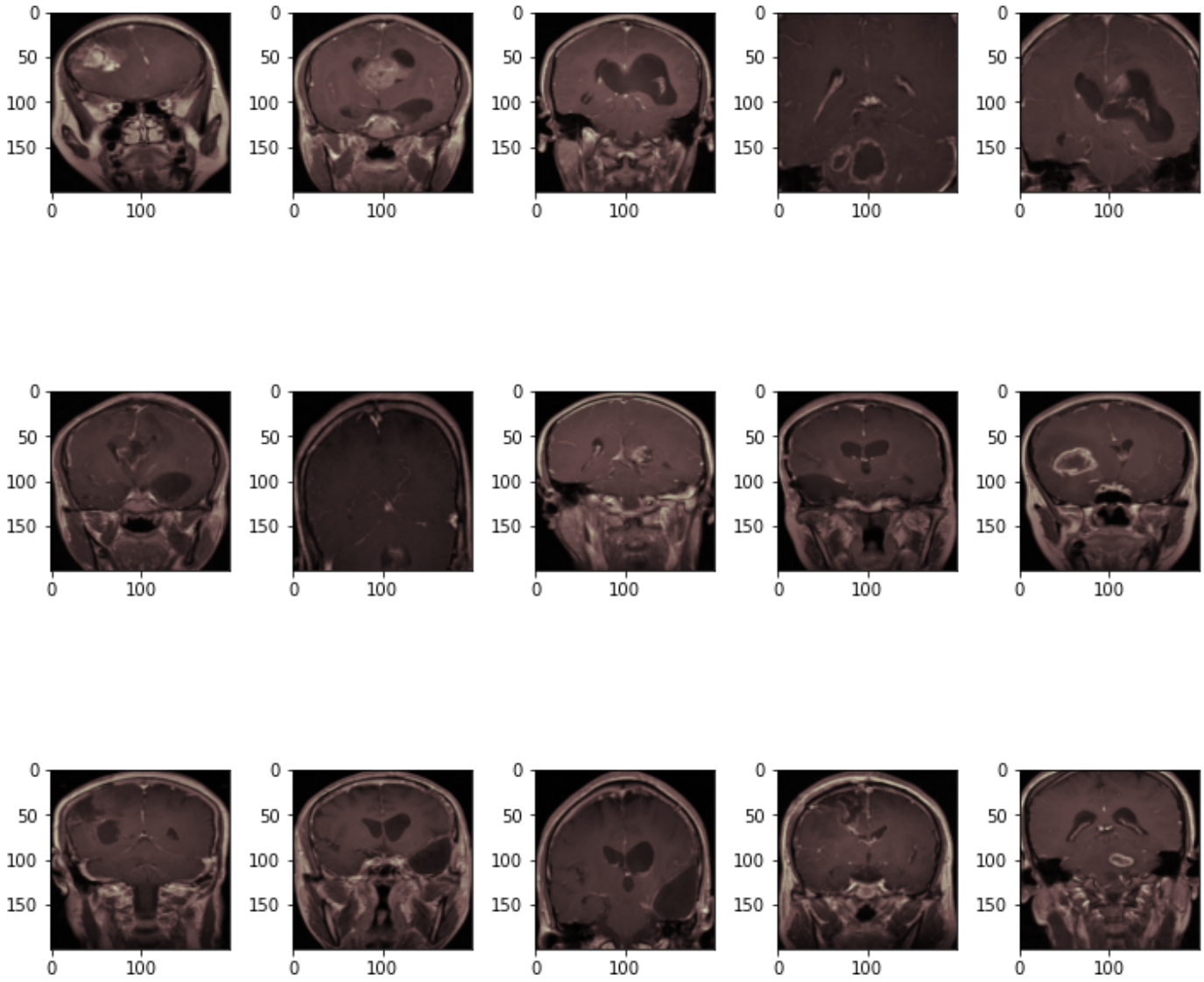
In [2]:
```
from keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint,TensorBoard,
from keras.layers import Input,Dropout, Dense,GlobalAveragePooling2D
from keras.models import Sequential,Model
from keras.applications.resnet import ResNet50
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from tensorflow import keras
import tensorflow as tf
from tqdm import tqdm
import seaborn as sns
import numpy as np
import itertools
import datetime

import cv2
import os
import io
```

In [3]:
```
labels = ['glioma', 'meningioma', 'notumor', 'pituitary']

x_train = [] # training images.
y_train  = [] # training labels.
x_test = [] # testing images.
y_test = [] # testing labels.

image_size = 200


for label in labels:
    trainPath = os.path.join('Training',label)
    for file in tqdm(os.listdir(trainPath)):
        image = cv2.imread(os.path.join(trainPath, file),0) # load images in gray.
        image = cv2.bilateralFilter(image, 2, 50, 50) # remove images noise.
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE) # produce a pseudocolored im
        image = cv2.resize(image, (image_size, image_size)) # resize images into 150*150
        x_train.append(image)
        y_train.append(labels.index(label))

    testPath = os.path.join('Testing',label)
    for file in tqdm(os.listdir(testPath)):
        image = cv2.imread(os.path.join(testPath, file),0)
        image = cv2.bilateralFilter(image, 2, 50, 50)
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
        image = cv2.resize(image, (image_size, image_size))
        x_test.append(image)
        y_test.append(labels.index(label))


x_train = np.array(x_train) / 255.0 # normalize Images into range 0 to 1.
x_test = np.array(x_test) / 255.0

print(x_train.shape)
print(x_test.shape)
```

```
100%|██████████| 132
1/1321 [00:05<00:00, 251.37it/s]
100%|██████████| 30
0/300 [00:00<00:00, 369.06it/s]
100%|██████████| 133
9/1339 [00:05<00:00, 263.23it/s]
```

```
100%|██████████| 306/306 [00:01<00:00, 277.74it/s]
100%|██████████| 1595/1595 [00:06<00:00, 238.63it/s]
100%|██████████| 405/405 [00:01<00:00, 219.36it/s]
100%|██████████| 1457/1457 [00:05<00:00, 254.80it/s]
100%|██████████| 300/300 [00:01<00:00, 211.81it/s]
(5712, 200, 200, 3)
(1311, 200, 200, 3)
```

In [4]:
```
images = [x_train[i] for i in range(15)]
fig, axes = plt.subplots(3, 5, figsize = (10, 10))
axes = axes.flatten()
for img, ax in zip(images, axes):
    ax.imshow(img)
plt.tight_layout()
plt.show()
```



In [5]:
```
import random
x_train, y_train = shuffle(x_train,y_train, random_state=42)

y_train = tf.keras.utils.to_categorical(y_train) #One Hot Encoding on the labels
y_test = tf.keras.utils.to_categorical(y_test)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, rando
```

```
print(x_val.shape)
```

```
(1143, 200, 200, 3)
```

In [6]:
```
from keras.preprocessing.image import ImageDataGenerator
# ImageDataGenerator transforms each image in the batch by a series of random translatio
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True)

# After you have created and configured your ImageDataGenerator, you must fit it on your
datagen.fit(x_train)
```

In [7]:
```
net = ResNet50(
    weights='imagenet', # Load weights pre-trained on ImageNet.
     include_top=False, # Do not include the ImageNet classifier at the top.
     input_shape=(image_size,image_size,3))
```

In [8]:
```
model = net.output
model = GlobalAveragePooling2D()(model)
model = Dropout(0.4)(model)
model = Dense(4, activation="softmax")(model)
model = Model(inputs= net.input, outputs= model)

#compile our model.
adam = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=adam, loss = 'categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
__________
 Layer (type)                    Output Shape         Param #     Connected to

==========================================================================================
==========
 input_1 (InputLayer)            [(None, 200, 200, 3  0           []

                                 )]


 conv1_pad (ZeroPadding2D)       (None, 206, 206, 3)  0           ['input_1[0][0]']


 conv1_conv (Conv2D)             (None, 100, 100, 64  9472        ['conv1_pad[0][0]']

                                 )


 conv1_bn (BatchNormalization)   (None, 100, 100, 64  256         ['conv1_conv[0][0]']

                                 )


 conv1_relu (Activation)         (None, 100, 100, 64  0           ['conv1_bn[0][0]']

                                 )
```

```
 pool1_pad (ZeroPadding2D)      (None, 102, 102, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 50, 50, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 50, 50, 64)   4160        ['pool1_pool[0][0]']

 conv2_block1_1_bn (BatchNormal  (None, 50, 50, 64)  256         ['conv2_block1_1_conv
[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 50, 50, 64)  0           ['conv2_block1_1_bn[0]
[0]']
 n)

 conv2_block1_2_conv (Conv2D)   (None, 50, 50, 64)   36928       ['conv2_block1_1_relu
[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 50, 50, 64)  256         ['conv2_block1_2_conv
[0][0]']
 ization)

 conv2_block1_2_relu (Activatio  (None, 50, 50, 64)  0           ['conv2_block1_2_bn[0]
[0]']
 n)

 conv2_block1_0_conv (Conv2D)   (None, 50, 50, 256)  16640       ['pool1_pool[0][0]']

 conv2_block1_3_conv (Conv2D)   (None, 50, 50, 256)  16640       ['conv2_block1_2_relu
[0][0]']

 conv2_block1_0_bn (BatchNormal  (None, 50, 50, 256)  1024       ['conv2_block1_0_conv
[0][0]']
 ization)

 conv2_block1_3_bn (BatchNormal  (None, 50, 50, 256)  1024       ['conv2_block1_3_conv
[0][0]']
 ization)

 conv2_block1_add (Add)         (None, 50, 50, 256)  0           ['conv2_block1_0_bn[0]
[0]',
```

```
                                                                  'conv2_block1_3_bn[0]
[0]']

 conv2_block1_out (Activation)  (None, 50, 50, 256)  0           ['conv2_block1_add[0]
[0]']

 conv2_block2_1_conv (Conv2D)   (None, 50, 50, 64)   16448       ['conv2_block1_out[0]
[0]']

 conv2_block2_1_bn (BatchNormal  (None, 50, 50, 64)  256         ['conv2_block2_1_conv
[0][0]']
 ization)

 conv2_block2_1_relu (Activatio  (None, 50, 50, 64)  0           ['conv2_block2_1_bn[0]
[0]']
 n)

 conv2_block2_2_conv (Conv2D)   (None, 50, 50, 64)   36928       ['conv2_block2_1_relu
[0][0]']

 conv2_block2_2_bn (BatchNormal  (None, 50, 50, 64)  256         ['conv2_block2_2_conv
[0][0]']
 ization)

 conv2_block2_2_relu (Activatio  (None, 50, 50, 64)  0           ['conv2_block2_2_bn[0]
[0]']
 n)

 conv2_block2_3_conv (Conv2D)   (None, 50, 50, 256)  16640       ['conv2_block2_2_relu
[0][0]']

 conv2_block2_3_bn (BatchNormal  (None, 50, 50, 256)  1024       ['conv2_block2_3_conv
[0][0]']
 ization)

 conv2_block2_add (Add)         (None, 50, 50, 256)  0           ['conv2_block1_out[0]
[0]',
                                                                  'conv2_block2_3_bn[0]
[0]']

 conv2_block2_out (Activation)  (None, 50, 50, 256)  0           ['conv2_block2_add[0]
[0]']

 conv2_block3_1_conv (Conv2D)   (None, 50, 50, 64)   16448       ['conv2_block2_out[0]
[0]']

 conv2_block3_1_bn (BatchNormal  (None, 50, 50, 64)  256         ['conv2_block3_1_conv
[0][0]']
```

```
 ization)


 conv2_block3_1_relu (Activatio  (None, 50, 50, 64)  0           ['conv2_block3_1_bn[0]
[0]']
 n)


 conv2_block3_2_conv (Conv2D)   (None, 50, 50, 64)   36928       ['conv2_block3_1_relu
[0][0]']

 conv2_block3_2_bn (BatchNormal  (None, 50, 50, 64)  256         ['conv2_block3_2_conv
[0][0]']
 ization)


 conv2_block3_2_relu (Activatio  (None, 50, 50, 64)  0           ['conv2_block3_2_bn[0]
[0]']
 n)


 conv2_block3_3_conv (Conv2D)   (None, 50, 50, 256)  16640       ['conv2_block3_2_relu
[0][0]']

 conv2_block3_3_bn (BatchNormal  (None, 50, 50, 256)  1024       ['conv2_block3_3_conv
[0][0]']
 ization)


 conv2_block3_add (Add)         (None, 50, 50, 256)  0           ['conv2_block2_out[0]
[0]',
                                                                  'conv2_block3_3_bn[0]
[0]']

 conv2_block3_out (Activation)  (None, 50, 50, 256)  0           ['conv2_block3_add[0]
[0]']

 conv3_block1_1_conv (Conv2D)   (None, 25, 25, 128)  32896       ['conv2_block3_out[0]
[0]']

 conv3_block1_1_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block1_1_conv
[0][0]']
 ization)


 conv3_block1_1_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block1_1_bn[0]
[0]']
 n)


 conv3_block1_2_conv (Conv2D)   (None, 25, 25, 128)  147584      ['conv3_block1_1_relu
[0][0]']
```

```
 conv3_block1_2_bn (BatchNormal  (None, 25, 25, 128)  512         ['conv3_block1_2_conv
[0][0]']
 ization)

 conv3_block1_2_relu (Activatio  (None, 25, 25, 128)  0           ['conv3_block1_2_bn[0]
[0]']
 n)

 conv3_block1_0_conv (Conv2D)   (None, 25, 25, 512)  131584      ['conv2_block3_out[0]
[0]']

 conv3_block1_3_conv (Conv2D)   (None, 25, 25, 512)  66048       ['conv3_block1_2_relu
[0][0]']

 conv3_block1_0_bn (BatchNormal  (None, 25, 25, 512)  2048       ['conv3_block1_0_conv
[0][0]']
 ization)

 conv3_block1_3_bn (BatchNormal  (None, 25, 25, 512)  2048       ['conv3_block1_3_conv
[0][0]']
 ization)

 conv3_block1_add (Add)         (None, 25, 25, 512)  0           ['conv3_block1_0_bn[0]
[0]',
                                                                  'conv3_block1_3_bn[0]
[0]']

 conv3_block1_out (Activation)  (None, 25, 25, 512)  0           ['conv3_block1_add[0]
[0]']

 conv3_block2_1_conv (Conv2D)   (None, 25, 25, 128)  65664       ['conv3_block1_out[0]
[0]']

 conv3_block2_1_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block2_1_conv
[0][0]']
 ization)

 conv3_block2_1_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block2_1_bn[0]
[0]']
 n)

 conv3_block2_2_conv (Conv2D)   (None, 25, 25, 128)  147584      ['conv3_block2_1_relu
[0][0]']

 conv3_block2_2_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block2_2_conv
[0][0]']
 ization)
```

```
 conv3_block2_2_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block2_2_bn[0]
[0]']
 n)

 conv3_block2_3_conv (Conv2D)   (None, 25, 25, 512)  66048       ['conv3_block2_2_relu
[0][0]']

 conv3_block2_3_bn (BatchNormal  (None, 25, 25, 512)  2048       ['conv3_block2_3_conv
[0][0]']
 ization)

 conv3_block2_add (Add)         (None, 25, 25, 512)  0          ['conv3_block1_out[0]
[0]',
                                                                  'conv3_block2_3_bn[0]
[0]']

 conv3_block2_out (Activation)  (None, 25, 25, 512)  0          ['conv3_block2_add[0]
[0]']

 conv3_block3_1_conv (Conv2D)   (None, 25, 25, 128)  65664       ['conv3_block2_out[0]
[0]']

 conv3_block3_1_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block3_1_conv
[0][0]']
 ization)

 conv3_block3_1_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block3_1_bn[0]
[0]']
 n)

 conv3_block3_2_conv (Conv2D)   (None, 25, 25, 128)  147584      ['conv3_block3_1_relu
[0][0]']

 conv3_block3_2_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block3_2_conv
[0][0]']
 ization)

 conv3_block3_2_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block3_2_bn[0]
[0]']
 n)

 conv3_block3_3_conv (Conv2D)   (None, 25, 25, 512)  66048       ['conv3_block3_2_relu
[0][0]']

 conv3_block3_3_bn (BatchNormal  (None, 25, 25, 512)  2048       ['conv3_block3_3_conv
[0][0]']
```

```
 ization)


 conv3_block3_add (Add)         (None, 25, 25, 512)  0           ['conv3_block2_out[0]
[0]',
                                                                  'conv3_block3_3_bn[0]
[0]']

 conv3_block3_out (Activation)  (None, 25, 25, 512)  0           ['conv3_block3_add[0]
[0]']

 conv3_block4_1_conv (Conv2D)   (None, 25, 25, 128)  65664       ['conv3_block3_out[0]
[0]']

 conv3_block4_1_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block4_1_conv
[0][0]']
 ization)

 conv3_block4_1_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block4_1_bn[0]
[0]']
 n)

 conv3_block4_2_conv (Conv2D)   (None, 25, 25, 128)  147584      ['conv3_block4_1_relu
[0][0]']

 conv3_block4_2_bn (BatchNormal  (None, 25, 25, 128)  512        ['conv3_block4_2_conv
[0][0]']
 ization)

 conv3_block4_2_relu (Activatio  (None, 25, 25, 128)  0          ['conv3_block4_2_bn[0]
[0]']
 n)

 conv3_block4_3_conv (Conv2D)   (None, 25, 25, 512)  66048       ['conv3_block4_2_relu
[0][0]']

 conv3_block4_3_bn (BatchNormal  (None, 25, 25, 512)  2048       ['conv3_block4_3_conv
[0][0]']
 ization)

 conv3_block4_add (Add)         (None, 25, 25, 512)  0           ['conv3_block3_out[0]
[0]',
                                                                  'conv3_block4_3_bn[0]
[0]']

 conv3_block4_out (Activation)  (None, 25, 25, 512)  0           ['conv3_block4_add[0]
[0]']
```

```
 conv4_block1_1_conv (Conv2D)   (None, 13, 13, 256)  131328      ['conv3_block4_out[0]
[0]']

 conv4_block1_1_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block1_1_conv
[0][0]']
 ization)

 conv4_block1_1_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block1_1_bn[0]
[0]']
 n)

 conv4_block1_2_conv (Conv2D)   (None, 13, 13, 256)  590080      ['conv4_block1_1_relu
[0][0]']

 conv4_block1_2_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block1_2_conv
[0][0]']
 ization)

 conv4_block1_2_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block1_2_bn[0]
[0]']
 n)

 conv4_block1_0_conv (Conv2D)   (None, 13, 13, 1024  525312      ['conv3_block4_out[0]
[0]']
                                )

 conv4_block1_3_conv (Conv2D)   (None, 13, 13, 1024  263168      ['conv4_block1_2_relu
[0][0]']
                                )

 conv4_block1_0_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block1_0_conv
[0][0]']
 ization)                       )

 conv4_block1_3_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block1_3_conv
[0][0]']
 ization)                       )

 conv4_block1_add (Add)         (None, 13, 13, 1024  0           ['conv4_block1_0_bn[0]
[0]',
                                )                                 'conv4_block1_3_bn[0]
[0]']

 conv4_block1_out (Activation)  (None, 13, 13, 1024  0           ['conv4_block1_add[0]
[0]']
                                )
```

```
 conv4_block2_1_conv (Conv2D)   (None, 13, 13, 256)  262400      ['conv4_block1_out[0]
[0]']

 conv4_block2_1_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block2_1_conv
[0][0]']
 ization)

 conv4_block2_1_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block2_1_bn[0]
[0]']
 n)

 conv4_block2_2_conv (Conv2D)   (None, 13, 13, 256)  590080      ['conv4_block2_1_relu
[0][0]']

 conv4_block2_2_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block2_2_conv
[0][0]']
 ization)

 conv4_block2_2_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block2_2_bn[0]
[0]']
 n)

 conv4_block2_3_conv (Conv2D)   (None, 13, 13, 1024  263168      ['conv4_block2_2_relu
[0][0]']
                                )

 conv4_block2_3_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block2_3_conv
[0][0]']
 ization)                       )

 conv4_block2_add (Add)         (None, 13, 13, 1024  0           ['conv4_block1_out[0]
[0]',
                                )                                 'conv4_block2_3_bn[0]
[0]']

 conv4_block2_out (Activation)  (None, 13, 13, 1024  0           ['conv4_block2_add[0]
[0]']
                                )

 conv4_block3_1_conv (Conv2D)   (None, 13, 13, 256)  262400      ['conv4_block2_out[0]
[0]']

 conv4_block3_1_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block3_1_conv
[0][0]']
 ization)
```

```
 conv4_block3_1_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block3_1_bn[0]
[0]']
 n)


 conv4_block3_2_conv (Conv2D)   (None, 13, 13, 256)  590080      ['conv4_block3_1_relu
[0][0]']


 conv4_block3_2_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block3_2_conv
[0][0]']
 ization)


 conv4_block3_2_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block3_2_bn[0]
[0]']
 n)


 conv4_block3_3_conv (Conv2D)   (None, 13, 13, 1024  263168      ['conv4_block3_2_relu
[0][0]']
                                )


 conv4_block3_3_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block3_3_conv
[0][0]']
 ization)                       )


 conv4_block3_add (Add)         (None, 13, 13, 1024  0          ['conv4_block2_out[0]
[0]',
                                )                                 'conv4_block3_3_bn[0]
[0]']


 conv4_block3_out (Activation)  (None, 13, 13, 1024  0          ['conv4_block3_add[0]
[0]']
                                )


 conv4_block4_1_conv (Conv2D)   (None, 13, 13, 256)  262400      ['conv4_block3_out[0]
[0]']


 conv4_block4_1_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block4_1_conv
[0][0]']
 ization)


 conv4_block4_1_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block4_1_bn[0]
[0]']
 n)


 conv4_block4_2_conv (Conv2D)   (None, 13, 13, 256)  590080      ['conv4_block4_1_relu
[0][0]']
```

```
 conv4_block4_2_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block4_2_conv
[0][0]']
 ization)

 conv4_block4_2_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block4_2_bn[0]
[0]']
 n)

 conv4_block4_3_conv (Conv2D)   (None, 13, 13, 1024  263168      ['conv4_block4_2_relu
[0][0]']
                                )

 conv4_block4_3_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block4_3_conv
[0][0]']
 ization)                       )

 conv4_block4_add (Add)         (None, 13, 13, 1024  0           ['conv4_block3_out[0]
[0]',
                                )                                 'conv4_block4_3_bn[0]
[0]']

 conv4_block4_out (Activation)  (None, 13, 13, 1024  0           ['conv4_block4_add[0]
[0]']
                                )

 conv4_block5_1_conv (Conv2D)   (None, 13, 13, 256)  262400      ['conv4_block4_out[0]
[0]']

 conv4_block5_1_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block5_1_conv
[0][0]']
 ization)

 conv4_block5_1_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block5_1_bn[0]
[0]']
 n)

 conv4_block5_2_conv (Conv2D)   (None, 13, 13, 256)  590080      ['conv4_block5_1_relu
[0][0]']

 conv4_block5_2_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block5_2_conv
[0][0]']
 ization)

 conv4_block5_2_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block5_2_bn[0]
[0]']
```

```
 n)


 conv4_block5_3_conv (Conv2D)   (None, 13, 13, 1024  263168      ['conv4_block5_2_relu
[0][0]']
                                )


 conv4_block5_3_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block5_3_conv
[0][0]']
 ization)                       )


 conv4_block5_add (Add)         (None, 13, 13, 1024  0           ['conv4_block4_out[0]
[0]',
                                )                                 'conv4_block5_3_bn[0]
[0]']


 conv4_block5_out (Activation)  (None, 13, 13, 1024  0           ['conv4_block5_add[0]
[0]']
                                )


 conv4_block6_1_conv (Conv2D)   (None, 13, 13, 256)  262400      ['conv4_block5_out[0]
[0]']


 conv4_block6_1_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block6_1_conv
[0][0]']
 ization)


 conv4_block6_1_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block6_1_bn[0]
[0]']
 n)


 conv4_block6_2_conv (Conv2D)   (None, 13, 13, 256)  590080      ['conv4_block6_1_relu
[0][0]']


 conv4_block6_2_bn (BatchNormal  (None, 13, 13, 256)  1024       ['conv4_block6_2_conv
[0][0]']
 ization)


 conv4_block6_2_relu (Activatio  (None, 13, 13, 256)  0          ['conv4_block6_2_bn[0]
[0]']
 n)


 conv4_block6_3_conv (Conv2D)   (None, 13, 13, 1024  263168      ['conv4_block6_2_relu
[0][0]']
                                )
```

```
 conv4_block6_3_bn (BatchNormal  (None, 13, 13, 1024  4096       ['conv4_block6_3_conv
[0][0]']
 ization)                       )


 conv4_block6_add (Add)         (None, 13, 13, 1024  0           ['conv4_block5_out[0]
[0]',
                                )                                 'conv4_block6_3_bn[0]
[0]']

 conv4_block6_out (Activation)  (None, 13, 13, 1024  0           ['conv4_block6_add[0]
[0]']
                                )


 conv5_block1_1_conv (Conv2D)   (None, 7, 7, 512)    524800      ['conv4_block6_out[0]
[0]']

 conv5_block1_1_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block1_1_conv
[0][0]']
 ization)

 conv5_block1_1_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block1_1_bn[0]
[0]']
 n)

 conv5_block1_2_conv (Conv2D)   (None, 7, 7, 512)    2359808     ['conv5_block1_1_relu
[0][0]']

 conv5_block1_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block1_2_conv
[0][0]']
 ization)

 conv5_block1_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block1_2_bn[0]
[0]']
 n)

 conv5_block1_0_conv (Conv2D)   (None, 7, 7, 2048)   2099200     ['conv4_block6_out[0]
[0]']

 conv5_block1_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block1_2_relu
[0][0]']

 conv5_block1_0_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block1_0_conv
[0][0]']
 ization)

 conv5_block1_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block1_3_conv
[0][0]']
```

```
 ization)


 conv5_block1_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block1_0_bn[0]
[0]',
                                                                  'conv5_block1_3_bn[0]
[0]']

 conv5_block1_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block1_add[0]
[0]']

 conv5_block2_1_conv (Conv2D)   (None, 7, 7, 512)    1049088     ['conv5_block1_out[0]
[0]']

 conv5_block2_1_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block2_1_conv
[0][0]']
 ization)

 conv5_block2_1_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block2_1_bn[0]
[0]']
 n)

 conv5_block2_2_conv (Conv2D)   (None, 7, 7, 512)    2359808     ['conv5_block2_1_relu
[0][0]']

 conv5_block2_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block2_2_conv
[0][0]']
 ization)

 conv5_block2_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block2_2_bn[0]
[0]']
 n)

 conv5_block2_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block2_2_relu
[0][0]']

 conv5_block2_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block2_3_conv
[0][0]']
 ization)

 conv5_block2_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block1_out[0]
[0]',
                                                                  'conv5_block2_3_bn[0]
[0]']

 conv5_block2_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block2_add[0]
[0]']
```

```
 conv5_block3_1_conv (Conv2D)   (None, 7, 7, 512)    1049088     ['conv5_block2_out[0]
[0]']

 conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_1_conv
[0][0]']
 ization)

 conv5_block3_1_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_1_bn[0]
[0]']
 n)

 conv5_block3_2_conv (Conv2D)   (None, 7, 7, 512)    2359808     ['conv5_block3_1_relu
[0][0]']

 conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_2_conv
[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_2_bn[0]
[0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu
[0][0]']

 conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block3_3_conv
[0][0]']
 ization)

 conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0]
[0]',
                                                                  'conv5_block3_3_bn[0]
[0]']

 conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0]
[0]']

 global_average_pooling2d (Glob  (None, 2048)        0           ['conv5_block3_out[0]
[0]']
 alAveragePooling2D)

 dropout (Dropout)              (None, 2048)         0           ['global_average_poolin
g2d[0][0]'
                                                                 ]

 dense (Dense)                  (None, 4)            8196        ['dropout[0][0]']
```

```
==========
Total params: 23,595,908
Trainable params: 23,542,788
Non-trainable params: 53,120
_________
```

In [11]:

```
%reload_ext tensorboard


class_names = list(labels)
def plot_to_image(figure):
    buf = io.BytesIO()
    plt.savefig(buf, format='png')
    plt.close(figure)
    buf.seek(0)

    digit = tf.image.decode_png(buf.getvalue(), channels=4)
    digit = tf.expand_dims(digit, 0)

    return digit

def plot_confusion_matrix(cm, class_names):
    figure = plt.figure(figsize=(8, 8))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Accent)
    plt.title("Confusion matrix")
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=45)
    plt.yticks(tick_marks, class_names)

    cm = np.around(cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)
    threshold = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        color = "white" if cm[i, j] > threshold else "black"
        plt.text(j, i, cm[i, j], horizontalalignment="center", color=color)

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    return figure

# Following function will make predictions from the model and log the confusion matrix a
def log_confusion_matrix(epoch, logs):
    predictions = np.argmax(model.predict(x_test), axis=1)
    cm = confusion_matrix(np.argmax(y_test,axis=1), predictions)
    figure = plot_confusion_matrix(cm, class_names=class_names)
    cm_image = plot_to_image(figure)

    with file_writer_cm.as_default():
        tf.summary.image("Confusion Matrix", cm_image, step=epoch)



# Run tensorBoard
%tensorboard --logdir logs
```

```
ERROR: Could not find `tensorboard`. Please ensure that your PATH
contains an executable `tensorboard` program, or explicitly specify
the path to a TensorBoard binary by setting the `TENSORBOARD_BINARY`
environment variable.
```

In [12]:
```
#create a writer variable for writing into the log folder.
file_writer_cm = tf.summary.create_file_writer(logdir)

tensorboard = TensorBoard(logdir, histogram_freq=1)


BATCH_SIZE = 64
EPOCHS = 50

Checkpoint = ModelCheckpoint(filepath = 'model-{epoch:02d}-{val_accuracy:.2f}-{val_loss:

ES = EarlyStopping(monitor = 'val_loss',min_delta = 0.001,patience = 5,mode = 'min',rest

RL = ReduceLROnPlateau(monitor = 'val_loss',factor = 0.3,patience = 5,verbose = 1,mode =

callbacks = [ES,RL,tensorboard,Checkpoint,LambdaCallback(on_epoch_end=log_confusion_matr

history = model.fit(datagen.flow(x_train, y_train, batch_size=20),validation_data = (x_v
```

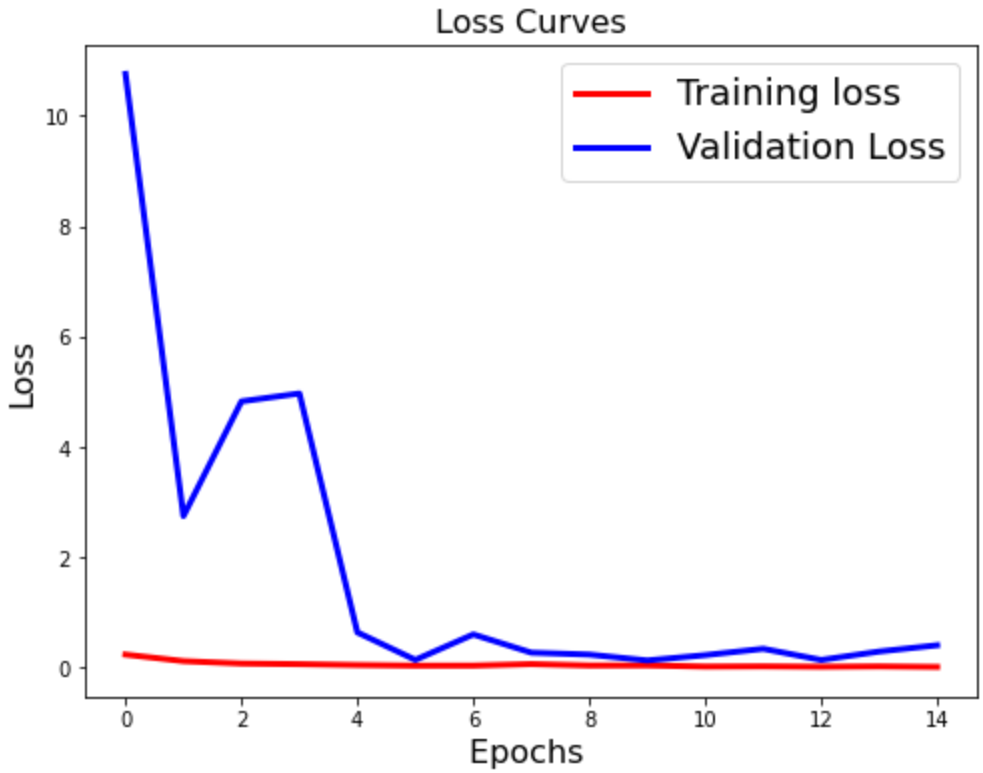
```
Epoch 1/50
229/229 [==============================] - ETA: 0s - loss: 0.2410 - accuracy: 0.9179
Epoch 1: val_loss improved from inf to 10.75426, saving model to model-01-0.23-10.75.h5
41/41 [==============================] - 71s 2s/step
229/229 [==============================] - 1358s 6s/step - loss: 0.2410 - accuracy: 0.91
79 - val_loss: 10.7543 - val_accuracy: 0.2283 - lr: 1.0000e-04
Epoch 2/50
229/229 [==============================] - ETA: 0s - loss: 0.1222 - accuracy: 0.9606
Epoch 2: val_loss improved from 10.75426 to 2.74907, saving model to model-02-0.24-2.75.
h5
41/41 [==============================] - 115s 3s/step
229/229 [==============================] - 1659s 7s/step - loss: 0.1222 - accuracy: 0.96
06 - val_loss: 2.7491 - val_accuracy: 0.2415 - lr: 1.0000e-04
Epoch 3/50
229/229 [==============================] - ETA: 0s - loss: 0.0784 - accuracy: 0.9744
Epoch 3: val_loss did not improve from 2.74907
41/41 [==============================] - 76s 2s/step
229/229 [==============================] - 1726s 8s/step - loss: 0.0784 - accuracy: 0.97
44 - val_loss: 4.8252 - val_accuracy: 0.3867 - lr: 1.0000e-04
Epoch 4/50
229/229 [==============================] - ETA: 0s - loss: 0.0676 - accuracy: 0.9781
Epoch 4: val_loss did not improve from 2.74907
41/41 [==============================] - 73s 2s/step
229/229 [==============================] - 8798s 39s/step - loss: 0.0676 - accuracy: 0.9
781 - val_loss: 4.9684 - val_accuracy: 0.4593 - lr: 1.0000e-04
Epoch 5/50
229/229 [==============================] - ETA: 0s - loss: 0.0515 - accuracy: 0.9836
Epoch 5: val_loss improved from 2.74907 to 0.64280, saving model to model-05-0.81-0.64.h
5
41/41 [==============================] - 74s 2s/step
229/229 [==============================] - 1308s 6s/step - loss: 0.0515 - accuracy: 0.98
36 - val_loss: 0.6428 - val_accuracy: 0.8119 - lr: 1.0000e-04
Epoch 6/50
229/229 [==============================] - ETA: 0s - loss: 0.0407 - accuracy: 0.9869
Epoch 6: val_loss improved from 0.64280 to 0.14432, saving model to model-06-0.97-0.14.h
5
41/41 [==============================] - 96s 2s/step
229/229 [==============================] - 1335s 6s/step - loss: 0.0407 - accuracy: 0.98
69 - val_loss: 0.1443 - val_accuracy: 0.9694 - lr: 1.0000e-04
Epoch 7/50
229/229 [==============================] - ETA: 0s - loss: 0.0411 - accuracy: 0.9871
Epoch 7: val_loss did not improve from 0.14432
41/41 [==============================] - 112s 3s/step
229/229 [==============================] - 1916s 8s/step - loss: 0.0411 - accuracy: 0.98
71 - val_loss: 0.6063 - val_accuracy: 0.8933 - lr: 1.0000e-04
Epoch 8/50
```

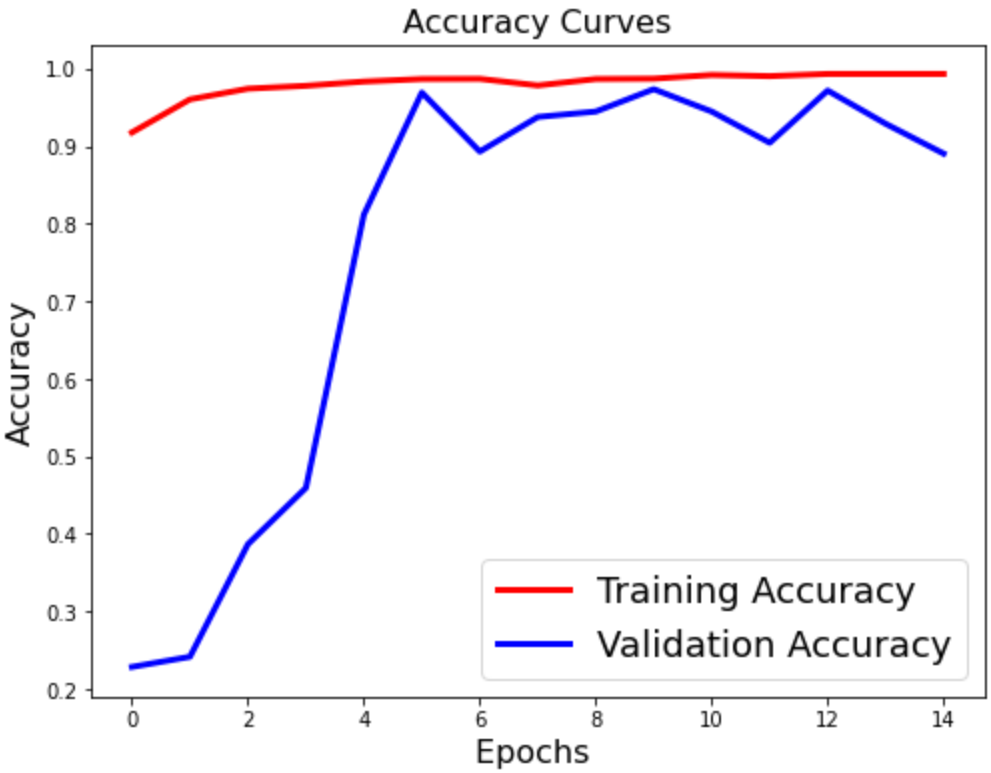
```
229/229 [==============================] - ETA: 0s - loss: 0.0686 - accuracy: 0.9786
Epoch 8: val_loss did not improve from 0.14432
41/41 [==============================] - 111s 3s/step
229/229 [==============================] - 1886s 8s/step - loss: 0.0686 - accuracy: 0.97
86 - val_loss: 0.2756 - val_accuracy: 0.9379 - lr: 1.0000e-04
Epoch 9/50
229/229 [==============================] - ETA: 0s - loss: 0.0431 - accuracy: 0.9869
Epoch 9: val_loss did not improve from 0.14432
41/41 [==============================] - 65s 2s/step
229/229 [==============================] - 1798s 8s/step - loss: 0.0431 - accuracy: 0.98
69 - val_loss: 0.2411 - val_accuracy: 0.9449 - lr: 1.0000e-04
Epoch 10/50
229/229 [==============================] - ETA: 0s - loss: 0.0444 - accuracy: 0.9873
Epoch 10: val_loss improved from 0.14432 to 0.13425, saving model to model-10-0.97-0.13.
h5
41/41 [==============================] - 74s 2s/step
229/229 [==============================] - 1346s 6s/step - loss: 0.0444 - accuracy: 0.98
73 - val_loss: 0.1343 - val_accuracy: 0.9738 - lr: 1.0000e-04
Epoch 11/50
229/229 [==============================] - ETA: 0s - loss: 0.0239 - accuracy: 0.9921
Epoch 11: val_loss did not improve from 0.13425
41/41 [==============================] - 74s 2s/step
229/229 [==============================] - 1335s 6s/step - loss: 0.0239 - accuracy: 0.99
21 - val_loss: 0.2309 - val_accuracy: 0.9449 - lr: 1.0000e-04
Epoch 12/50
229/229 [==============================] - ETA: 0s - loss: 0.0275 - accuracy: 0.9906
Epoch 12: val_loss did not improve from 0.13425
41/41 [==============================] - 111s 3s/step
229/229 [==============================] - 1372s 6s/step - loss: 0.0275 - accuracy: 0.99
06 - val_loss: 0.3460 - val_accuracy: 0.9046 - lr: 1.0000e-04
Epoch 13/50
229/229 [==============================] - ETA: 0s - loss: 0.0202 - accuracy: 0.9934
Epoch 13: val_loss did not improve from 0.13425
41/41 [==============================] - 111s 3s/step
229/229 [==============================] - 1874s 8s/step - loss: 0.0202 - accuracy: 0.99
34 - val_loss: 0.1436 - val_accuracy: 0.9720 - lr: 1.0000e-04
Epoch 14/50
229/229 [==============================] - ETA: 0s - loss: 0.0266 - accuracy: 0.9934
Epoch 14: val_loss did not improve from 0.13425
41/41 [==============================] - 111s 3s/step
229/229 [==============================] - 1887s 8s/step - loss: 0.0266 - accuracy: 0.99
34 - val_loss: 0.2961 - val_accuracy: 0.9291 - lr: 1.0000e-04
Epoch 15/50
229/229 [==============================] - ETA: 0s - loss: 0.0170 - accuracy: 0.9934Rest
oring model weights from the end of the best epoch: 10.

Epoch 15: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.

Epoch 15: val_loss did not improve from 0.13425
41/41 [==============================] - 72s 2s/step
229/229 [==============================] - 1330s 6s/step - loss: 0.0170 - accuracy: 0.99
34 - val_loss: 0.4098 - val_accuracy: 0.8906 - lr: 1.0000e-04
Epoch 15: early stopping
```

In [13]:

```python
#Plot the Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)
plt.show()

#Plot the Accuracy Curves
```

```
plt.figure(figsize=[8,6])
plt.plot(history.history['accuracy'],'r',linewidth=3.0)
plt.plot(history.history['val_accuracy'],'b',linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)
plt.show()
```

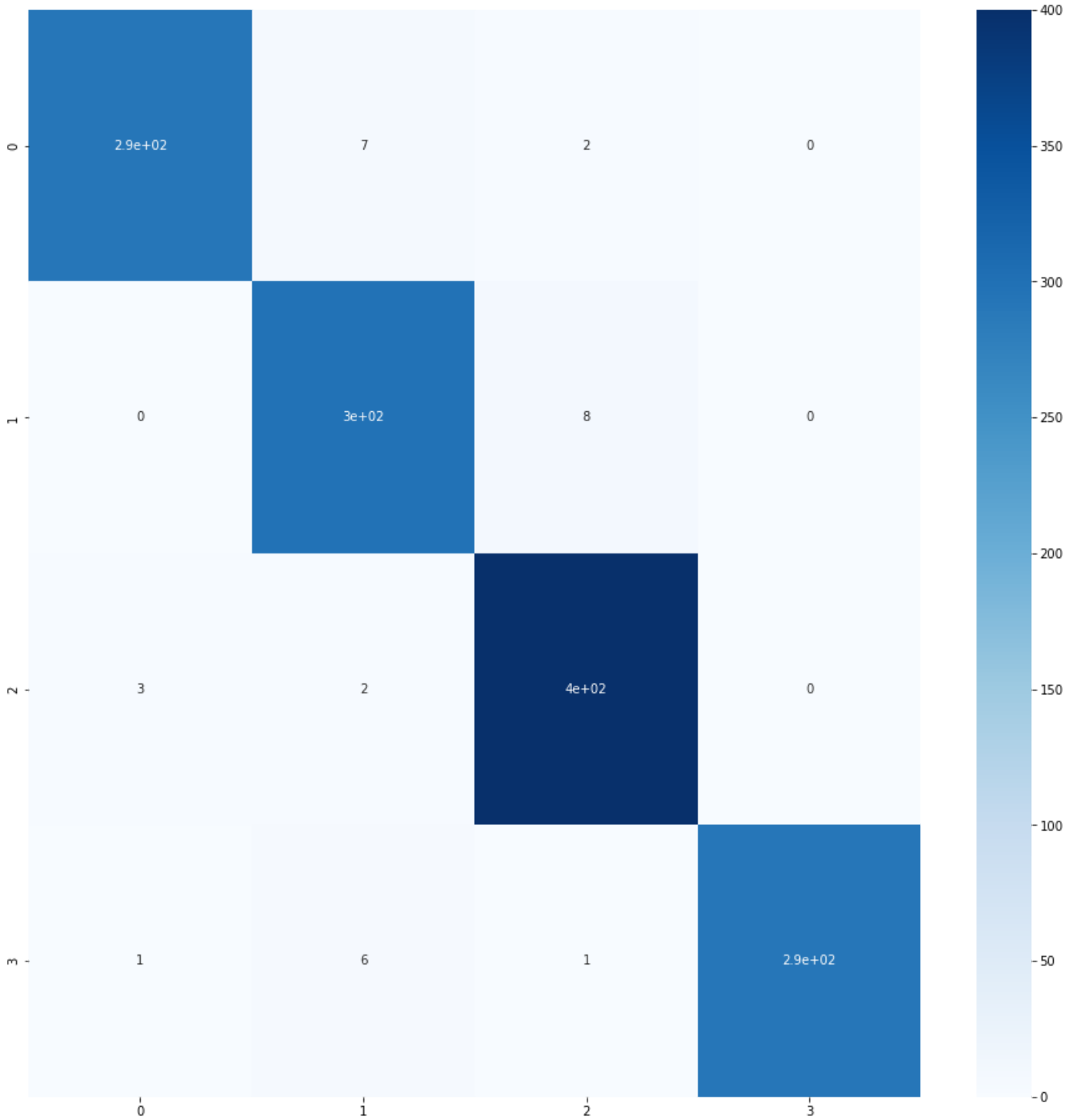




In [14]:

```
predicted_classes = np.argmax(model.predict(x_test), axis = 1)
confusionmatrix = confusion_matrix(np.argmax(y_test,axis=1), predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar = True)
print(classification_report(np.argmax(y_test,axis=1), predicted_classes))
```

```
41/41 [==============================] - 68s 2s/step
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       300
           1       0.95      0.97      0.96       306
           2       0.97      0.99      0.98       405
           3       1.00      0.97      0.99       300

    accuracy                           0.98      1311
   macro avg       0.98      0.98      0.98      1311
weighted avg       0.98      0.98      0.98      1311
```



In [15]:
```
loss,acc = model.evaluate(x_test,y_test)
```

```
41/41 [==============================] - 69s 2s/step - loss: 0.1300 - accuracy: 0.9771
```

In [ ]: