

A Novel Method for Brain Tumor Classification Using Capsule Network

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in

INFORMATION TECHNOLOGY

By

KARTHIKEYAN N (Reg No 38120039)

MADHAN L (Reg No 38120047)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC I 12B Status by UGC I Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC | 12B Status by UGC | Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

SCHOOL OF COMPUTING

BONAFAIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **KARTHIKEYAN N(Reg no:38120039)** and **MADHAN L(Reg no:38120047)** carried out the project entitled "**A NOVEL METHOD FOR BRAIN TUMOR CLASSIFICATION USING CAPSULE NETWORK**" under our supervision from OCTOBER 2021 to JANUARY 2022

Internal Guide

Head of the Department

Dr. SENDURU SRINIVASALU M.E., Ph.D.,

Dr.R.SUBHASHINI M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **KARTHIKEYAN N** and **MADHAN L** hereby declare that the Project Report entitled done by me under the guidance of **Dr. SENDURU SRINIVASALU M.E., Ph.D.,** at Sathyabama Institute of Science and Technology (Deemed to be University), Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119 is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

1.

2.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

We are pleased to acknowledge my sincere thanks to Board of management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey my thanks to **Dr. T. SASIKALA M.E., Ph.D., Dean, School of Computing** and **Dr. R. SUBHASHINI M.E., Ph.D., Head of the Department, Department of Information Technology** for providing me the necessary support and details at the right time during the progressive reviews.

I would like to express our sincere and deep sense of gratitude to our Project Guide

Dr. SENDURU SRINIVASALU M.E., Ph.D., for her valuable guidance, suggestions and constant

encouragement paved way for the successful completion of our project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of **INFORMATION TECHNOLOGY** who were helpful in many ways for the completion of the project.

ABSTRACT

A Novel Method for Brain Tumor Classification Using Capsule Network

Brain tumor is considered as one of the deadliest and most common form of cancer both in children and in adults. Consequently, determining the correct type of brain tumor in early stages is of significant importance to devise a precise treatment plan and predict patient's response to the adopted treatment. In this regard, there has been a recent surge of interest in designing Convolutional Neural Networks (CNNs) for the problem of brain tumor type classification. However, CNNs typically require large amount of training data and cannot properly handle input transformations. Capsule networks (referred to as CapsNets) are brand new machine learning architectures proposed very recently to overcome these shortcomings of CNNs, and posed to revolutionize deep learning solutions. Of particular interest to this work is that Capsule networks are robust to rotation and affine transformation, and require far less training data, which is the case for processing medical image datasets including brain Magnetic Resonance Imaging (MRI) images.

Keywords: Brain Tumor, Convolutional Neural Networks (CNN), Magnetic Resonance Imaging (MRI).

INDEX

CHAPTER NO	CHAPTER NAME	PAGE NO
	ABSTRACT	05
	LIST OF FIGURES	09
	LIST OF ABBREVIATION	10
1	INTRODUCTION	11
	1.1 GENERAL	11
2	LITERATURE SURVEY	14
	2.1. TWO-STEP VERIFICATION OF BRAIN TUMOR SEGMENTATION USING WATERSHED-MATCHING ALGORITHM	14
	2.2 A NOVEL IMPROVED ANTLION OPTIMIZER ALGORITHM AND ITS COMPARATIVE PERFORMANCE	14
	2.3 DETECTION OF BRAIN TUMOR IN MRI IMAGES BY APPLYING SEGMENTATION AND AREA CALCULATION METHOD USING SCILAB.	15
	2.4 INTEGRATION OF CONTOURLET TRANSFORM AND CANNY EDGE DETECTOR FOR BRAIN IMAGE SEGMENTATION	15
	2.5 MR IMAGE CLASSIFICATION USING ADABOOST FOR BRAIN TUMOR TYPE	16
	2.6 SEGMENTATION AND CLASSIFICATION OF MRI BRAIN TUMOR	16

	2.7 CLASSIFICATION OF TUMORS AND IT STAGES IN BRAIN MRI USING SUPPORT VECTOR MACHINE AND ARTIFICIAL NEURAL NETWORK	16
	2.8 OPTIMIZATION OF A VARIATIONAL MODEL USING DEEP LEARNING	17
	2.9 MR IMAGE CLASSIFICATION USING ADABOOST FOR BRAIN TUMOR TYPE	17
	2.10 DETECTION AND ANALYSIS OF BRAIN TUMOR FROM MRI BY INTEGRATED THRESHOLDING AND MORPHOLOGICAL PROCESS	18
3	AIM AND SCOPE	19
	3.1 AIM	19
	3.2 SCOPE	19
	3.3 EXISTING SYSTEM	19
	3.4 PROPOSED SYSTEM	20
	3.4.1 INPUT DATABASE	20
	3.4.2 IMAGE PRE-PROCESSING	20
	3.4.3 SEGMENTATION	21
	3.4.4 FEATURE EXTRACTION	21
	3.4.5 CLASSIFICATION CNN ALGORITHM	22
4	EXPREMENTAL OR MATERIAL METHODS ALGORITHM	23
	4.1 SYSTEM ARCHITECTURE	23
	4.2 MACHINE LEARNING	23
	4.2.1 SUPERVISED LEARNING	23
	4.2.2 UNSUPERVISED LEARNING	24
	4.3 ALGORITHM	25
	4.3.1 CAPSULE NEURAL NETWORK	25
	4.3.2 ALGORITHM USE	25

5	RESULT DISCUSSION AND PERFORMANCE ANALYSIS	26
	5.1 SYSTEM CONFIGURATION	26
	5.1.1 HARDWARE REQUIREMENTS	26
	5.1.2 SOFTWARE REQUIREMENTS	26
	5.2 SAMPLE CODE	31
	5.3 TRAINING DATA	71
	5.4 DATA LOAD	73
	5.5 PERFORMANCE MEASURES	76
	5.6 PERFORMANCE EVALUTION	76
	5.7 OUTPUT	79
	5.8 RESULT	81
6	SUMMARY AND CONCLUSION	82
	REFERENCE	83

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
Figure 1.0	Normal Brain Image	12
Figure 4.1	System Architecture	23
Figure 5.3.1	Training Data History	73
Figure 5.6.1	Represent Performance analysis of CNN	77
Figure 5.6.2	Represents the performance of proposed CNN	78
Figure 5.7.1	Pituitary Tumor	79
Figure 5.7.2	Meningioma Tumor	80
Figure 5.7.3	Glioma Tumor	80

LIST OF ABBREVIATION

ACRONYM	ABBREVIATIONS
MRI	Magnetic Resonance Imaging
CT	Computed Tomography
PET	Positron Emission Tomography
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
SVM	Support Vector Machine
AI	Artificial Intelligence

CHAPTER 1

INTRODUCTION

1.1 GENERAL

The brain tumor is a massive or abnormal cell growth in a brain region. This can spread to any person among different ages. A brain tumor can be classified into different types; these tumors can either be Malignant (cancerous) or Benign (non-cancerous). The tumor that initially affects the brain are termed as the primary tumors whereas the tumors present in the other part of the body that might spread to the brain are known as the secondary tumors or metastatic tumors. This paper includes the method of detecting the tumor region and classifying them either normal, malignant or benign. The (MRI) image magnetic resonance image is an expertise technique that gives a rich standard image of the anthropoid. This helps in medical recognition. Tumor in brain is one of the most dangerous diseases which if not detected at the early stages can even risk the life. Currently, the methods used by neurologists for analysis are not completely error free and states that manual segmentation isn't a good idea. This study presents machine learning based approach for segmentation of brain images and identification of tumor using SVM classification approach which improve the performance, minimize the complexity and works on real time data. Machine learning (ML) is a centre of manmade consciousness. It is the principal approach towards planning wise computers. AI for the most part utilizes acceptance and blend hence enabling PCs to get new information by reproducing human learning conduct. It redesigns current learning to ceaselessly improve computer execution. AI has additionally been broadly applied in numerous fields, for example, medical diagnosis, agriculture, scene analysis and computer vision. With the advancements of neural networks, the idea of deep learning (DL) has been proposed. DL isn't just an improvement in neural systems, yet additionally another field in ML research. The effective utilization of DL brings ML closer to manmade reasoning. Since profound learning comprises of progressively concealed layers in contrast with neural systems,

an increasingly conceptual significant level element portrayal for various classes is framed by utilizing different layers to join low-level features. Like computerized reasoning, deep adapting endeavours to fabricate and reproduce human mind to examine the learning procedure of neural system which mimics the learning component of human cerebrum when it happens to comprehend obscure ideas. DL can be utilized to consequently discover features from a given dataset for every application. By and large, traditional techniques depend on some earlier knowledge to concentrate in a specific application. In this manner, these techniques are hand crafted feature extraction strategies. DL can discover new features that are reasonable to explicit applications yet have never been recently found by specialists. The medical business is one of the most noticeable enterprises where DL can assume an immense job particularly with regards to medicinal imaging. There are various research works that demonstrated productivity and adequacy of the utilization of DL in the field.

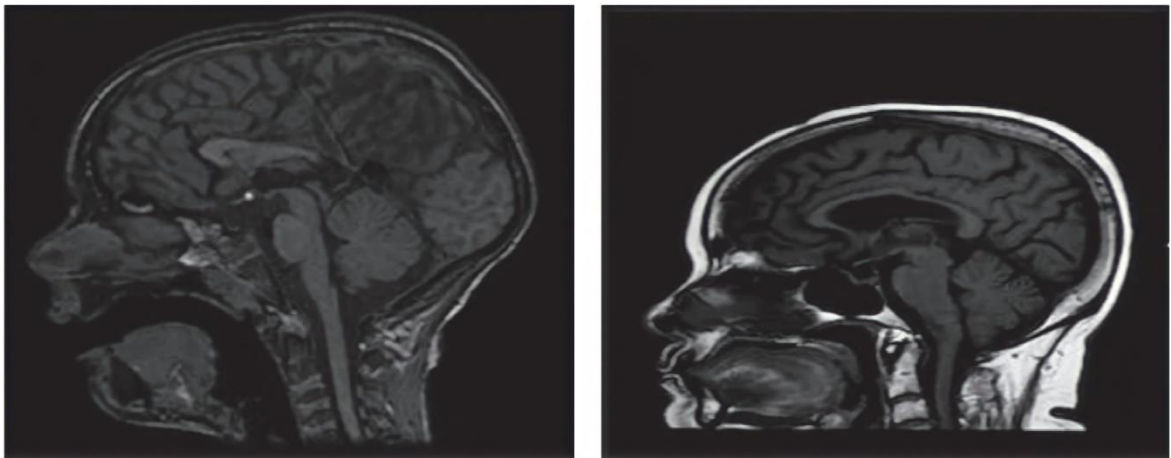


FIGURE 1.0 Normal Brain Image

Brain is a vital part of human body and comprised of many specific regions. Brain tumor is incited because of an ordinary boom of cells. It comes under two categories that are malignant or benign. The benign tumors do not contain cancerous cells while the malignant tumors are comprised of cancerous cells. Brain tumors associations

classify the brain tumor in four grades where grade I and II are referred to as benign and the remaining III and IV are labelled as malignant. The development rate of benign is relatively low when contrasted with malignant. When the benign isn't dealt with opportune then it is changed into malignant tumor. Along these lines, early discovery of tumor is desirable. Grade II patients require ordinary treatment and checking through magnetic resonance imaging (MRI). MRI depicts one of radio imaging types. Some more types include computed tomography (CT) and positron emission tomography (PET) with the help of which brain tumor can be diagnosed in medical imaging. MRI is mostly useful to analyse tumors in general clinical routines. It provides minute information about human cerebrum. One advantage of MRI is its non-intrusive nature and no ionization radiation. DL is most broadly utilized for brain tumor investigation in numerous scenarios for example, ordinary or irregular cerebrum tumor categorization and segmentation. A convolution neural network (CNN) represents a most famous DL structure utilized generally for characterization and segmentation of brain tumors. This is accomplished by convolving the images utilizing learned channels to assemble a chain of features activations. This deep learning work is done in a few layers to such an extent that the features acquired are interpretation and twisting invariant bringing about the high level of exactness. General computer vision steps for image analysis include: image acquiring (input), pre-processing, segmentation (to highlight required regions), feature extraction (to acquire best characteristic points in an image), features reduction (to acquire optimized features) and classification (to predict the image types e.g. HGG or LGG). Nowadays, deep learning strategies have become the cutting-edge strategies for brain tumor analysis as contrasted to old traditional methods. This manuscript mainly encompasses on a survey about recent trends for brain image segmentation and classification by employing DL strategies. The major focus of this work is to give the readers an insight to these areas so that they can contribute further in an effective manner for brain tumor research.

CHAPTER 2

LITERATURE SURVEY

2.1 Two-step verification of brain tumor segmentation using watershed-matching algorithm.

Authors: Hasan, S. K., & Ahmad, M. (2018).

Hasan, S. K and Ahmad, M. et al., 2018 [17] proposed research on two-stage authenticated technique for detection of brain tumor. In this research, the proposed technique is authenticated scheme for detection of brain tumor, called a watershed matched approach. The segmentation of brain tumor area was done through classification approach of the watershed algorithm. In addition, scale invariant feature transform technique was also used to extract feature regions and matching divided area of brain tumor with an actual picture. Along with that, compute dimensions of the tumor using various approaches. They implemented an approach through free surfer system software to search the width of cortical and the compute variance among benign and malignant.

2.2 A novel improved antlion optimizer algorithm and its comparative performance. Neural C.

Authors: Kilic, H., Yuzgec, U., & Karakuzu, C. (2018).

Kilic, H., Yuzgec, U and Karakuzu, C et al., 2018 [18] implemented a novel technique for the development of ant lion optimization (ALO) that encouraged by hunting method of ant lion. The main issue of the used method is maximum time consuming, so proposed a new randomized walk technique to overcome this issue. In this research, the ant lion optimized selection method is proposed that depends on event choice and compared with an investigative approach for benchmarks variables. In addition, testing and training of the ant lion optimized selection method were done. Experimental analysis proved that the planned method gives a better presentation as compared to existing ant lion optimization algorithm

2.3 Detection of Brain Tumor in MRI Images by Applying Segmentation and Area Calculation Method Using SCILAB.

Authors: Kurnar, M., Sinha, A., and Bansode, N. V. (2018). In 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA) (pp. 1-5). IEEE

Kurnar, M., Sinha, A. and Bansode, N. V et al., 2018[19] proposed research on the detection and visualization of the tissues of the brain. Detect the dimension of the tumor of the brain image, different stages of the brain can be determined. The step may be cancerous or non-cancerous. To recognize the size of the brain, a tumor of the brain can be detected using a k mean clustering algorithm based on the separation and morphological technique. The morphological method used for the extraction of the features of the parts of the body through MRI scanning. Experimental analysis on calculating the extracted features of the portion of the brain disease.

2.4 Integration of Contourlet Transform and Canny Edge Detector for Brain Image Segmentation.

Authors: Akter L. A., and Kwon, G. R. (2018) In 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN) (pp. 798-800). IEEE

Akter, L. A., and Kwon, G. R et al., 2018 [20] proposed research on novel distributed picture segmentation technique for magnetic resonance brain pictures. In this technique, the different conversion is called a contour transformation that is the through canny edge detection. In this research work, the distribution technique acquired for the improved method on contour constants through the canny edge detection approach. Experimental analysis was done on improvement of contourlet conversion through the segmentation process using canny edge detection technique.

2.5 MR Image classification using Adaboost for brain tumor type

Authors: Astina Minz, Prof. Chandrakant Mahobiya in IEEE 7th International Advance Computing Conference (IACC) 2017.

Astina Minz et al. [2], In medical diagnostic application, early deformity discovery is a crucial task as it gives basic understanding into analysis. Medical imaging system is currently developing field in engineering. Magnetic Resonance imaging (MRI) is one of those dependable imaging strategies on which medical diagnostic depends on. Manual examination of those images is a repetitive activity as the measure of information and minute points of interest are difficult to perceive by the human.

2.6 Segmentation and Classification of MRI Brain Tumor

Authors: Mukambika P. S. and Uma Rani K. International Research Journal of Engineering and Technology (IRJET) Volume: 04 Issue: 07 July -2017.

Mukambika P. S., et al. [5], Bio-medical image handling is the most testing and rising field in medical finding. Handling of MRI images is one of the troublesome parts of this field. The present work introduces the similar investigation of two methods utilized for tumor identification of MRI images.

2.7 Classification of Tumors and It Stages in Brain MRI Using Support Vector Machine and Artificial Neural Network International Conference on Electrical, Computer and Communication Engineering (ECCE), Cox's Bazar, Bangladesh, February 16-18, 2017 IEEE.

Authors: Rasel Ahmmed, Anirban Sen Swakshar, Md. Foisal Hossain, and Md. Abdur Rafiq International Conference on Electrical, Computer and Communication Engineering (ECCE), Cox's Bazar, Bangladesh, February 16-18, 2017 IEEE.

Rasel Ahmmed et al. [8], Cell is the littlest unit of tissues, whose anomalous development causes tumor in Brain. Support Vector Machine (SVM), Artificial Neural Network (ANN) based tumor and its stages classification in brain MRI images is

displayed in this examination work. This work is begun with the upgrade of the brain MRI images which are acquired from oncology division of College of Maryland Medical Center.

2.8 Optimization of a variational model using deep learning: An application to brain tumor segmentation

Authors: Ramirez, Martin A, Schiavi E 2018

By incorporating a saliency term to a classical Total Variation based restoration functional, the new model is able to discriminate what is relevant (salient) from the background. The resulting non-convex and non-smooth problem has a special Difference of Convex structure and it is numerically solved using a proximal point algorithm that involves using a primal-dual method to solve an internal non-smooth sub-problem. We introduce a Deep Learning framework for using available knowledge from a specific application to optimize the parameters of the energy functional. The proposed framework is successfully tested on a database of Magnetic Resonance Images of the brain of patients suffering from a reaching a Dice Similarity Coefficient of 85.7%.

2.9 MR Image Classification Using Adaboost for Brain Tumor Type

Authors: Astina Minz Chandrakant Mahobiya, 2017

In medical diagnostic application, early defect detection is a crucial task as it provides critical insight into diagnosis. Medical imaging technique is actively developing field in engineering. Magnetic Resonance imaging (MRI) is one those reliable imaging techniques on which medical diagnostic is based upon. Manual inspection of those images is a tedious job as the amount of data and minute details are hard to recognize by the human. For this automating those techniques are very crucial. In this paper, we are proposing a method which can be utilized to make tumor detection easier.

2.10 Detection and analysis of brain tumor from MRI by Integrated Thresholding and Morphological Process with Histogram based method

Authors: Md. Rezwanul Islam; Md. Reezbhan Imteaz ; Marium-E-Jannat in 2018 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering

This paper describes a system that can detect brain tumor more precisely and analysis the different features of the tumor. Our system proposed a computer aided image processing-based method to that gives improved accuracy rate of the brain tumor detection along with the calculation of the tumor size (surface area of the tumor) and its location. It also provides with the information that helps to determine whether the tumor is malignant or not. The system, we describing in this paper, detect brain tumor from MRI by integrated thresholding and morphological process with histogram-based method and gives a thorough analysis. In our experiment we have used BRATS database of magnetic resonance images. The rate of successful detection of our methodology is 86.84%.

CHAPTER 3

AIM AND SCOPE

3.1 AIM

This work is motivated to enhance the diagnostic capability of physicians and reduce the time required for accurate diagnosis. Image segmentation is performed on the input images. This enables easier analysis of the image thereby leading to better tumor detection efficiency.

3.2 SCOPE

Our aim is to develop an automated system for enhancement, segmentation and classification of brain tumors. The system can be used by neurosurgeons and healthcare specialists. The system incorporates image processing, pattern analysis, and computer vision techniques and is expected to improve the sensitivity, specificity, and efficiency of brain tumor screening. The primary goal of medical imaging projects is to extract meaningful and accurate information from these images with the least error possible. The proper combination and parameterization of the phases enables the development of adjunct tools that can help on the early diagnosis or the monitoring of the tumor identification and locations.

3.3 EXISTING SYSTEM

In the existing system for brain tumour classification, they used CNN methods for classification, but CNN doesn't really capture the location and orientation of the brain cancer in their predictions. They lose all internal data on the tumor's position and orientation, and all of the information is directed to the same neurons, which may not be able to handle this input.

3.4 PROPOSED SYSTEM

Adopt and incorporate CapsNets to develop a better architecture that increases the accuracy of the classification challenge at hand for the problem of brain tumour classification. Investigate the CapsNets over-fitting problem using an actual set of MRI pictures. Examine whether CapsNets can provide a better match for the entire brain image or just the segmented tumour, and if so, how To further understand the learnt features, provide a visualisation paradigm for the CapsNet output.

Using the following modules

1. Input Database
2. Image Pre-Processing
3. Segmentation
4. Feature Extraction
5. Classification CNN algorithm

3.4.1 Input Database

The Kaggle Brain MRI data set has been downloaded. There are around 2000 MRI photos in the MRI dataset, which include Normal, Glioma, Pituitary, and Meningioma. These MRI scans will be used to create the database

3.4.2 Image Pre-processing

The Brain MRI image dataset has been downloaded from the Kaggle. The MRI dataset consists of around 1900 MRI images, including normal, benign, and malignant. These MRI images are taken as input to the primary step. The pre-processing is an essential and initial step in improving the quality of the brain MRI Image. The critical steps in pre-processing are the reduction of impulsive noises and image resizing. In the initial phase, we convert the brain MRI image into its corresponding gray-scale

image. The removal of unwanted noise is done using the adaptive bilateral filtering technique to remove the distorted noises that are present in the brain picture. This improves the diagnosis and also increase the classification accuracy rate.

3.4.3 Segmentation

Image segmentation is a technique of segregating the image into many parts. The basic aim of this segregation is to make the images easy to analyze and interpret with preserving the quality. This technique is also used to trace the objects' borders within the images. This technique labels the pixels according to their intensity and characteristics. Those parts represent the entire original image and acquire its characteristics such as intensity and similarity. The image segmentation technique is used to create contours of the body for clinical purposes. Segmentation is used in machine perception, malignant disease analysis, tissue volumes, anatomical and functional analyses, virtual reality visualization, and anomaly analysis, and object definition and detection.

3.4.4 Feature Extraction

Feature extraction is a method that compresses a large amount of raw data into a lesser number of processing units. The large number of factors in these massive datasets requires a significant amount of computational power. Feature extraction refers to approaches for selecting and combining variables into aspects in order to reduce the level of information that must be processed while maintaining the original data set's accuracy and consistency.

3.4.5 Classification CNN Algorithm

Classification is the best approaches for identification of images like any kind of medical imaging. All classification algorithms are based on the prediction of image, where one or more features and that each of these features belongs to one of several classes.

An automatic and reliable classification method Convolutional Neural Network (CNN) will be used since it is robust in structure which helps in identifying every minute details. A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics.

A CNN is able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

CHAPTER 4

EXPERIMENTAL OR MATERIAL METHODS ALGORITHM

4.1 SYSTEM ARCHITECTURE

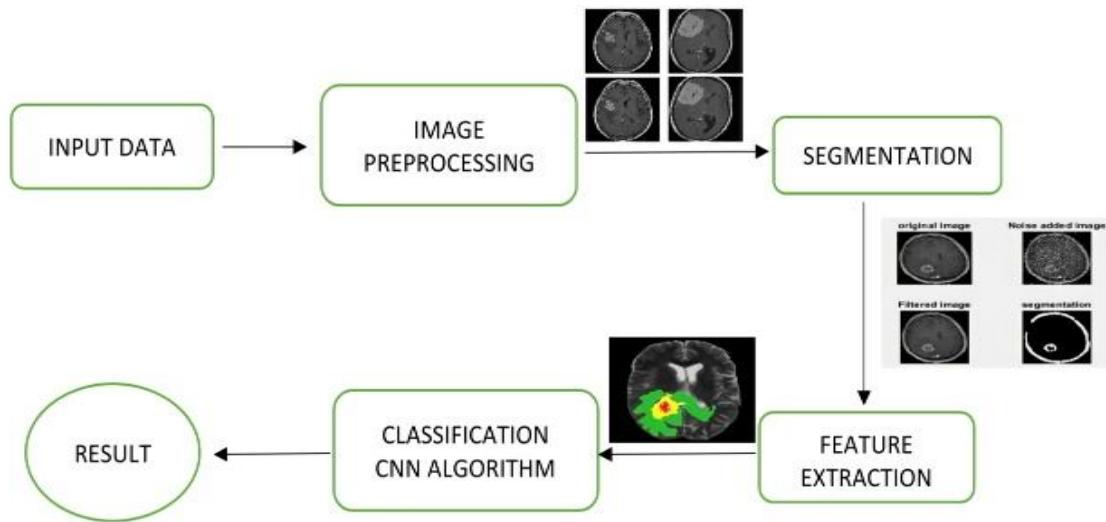


Figure 4.1 System Architecture

4.2 Machine Learning

In machine learning, classification refers to a predictive modelling problem where a class label is predicted for a given example of input data.

4.2.1 Supervised Learning

Supervised learning is the type of machine learning in which machines are trained using well "labelled" training data, and on the basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable.

4.2.2 Unsupervised learning

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar to how a human learns to think by their own Experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabelled and uncategorized data which make Unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

4.3 ALGORITHM

4.3.1 Capsule Neural Network

A CapsNet is a machine learning technology that uses an artificial neural network to better understand hierarchical relationships. The aim of the researcher is to produce a brain structure that is as close to natural as possible.

4.3.2 Algorithm Use:

Capsule Networks are a new type of network that replicates the way a human brain learns by modelling hierarchical relationships in processing a picture. This is in direct contrast to the approach taken by normal neural networks.

CHAPTER 5

PERFORMANCE ANALYSIS

5.1 SYSTEM CONFIGURATION

5.1.1 Hardware Requirements:

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list, especially in case of operating systems. The minimal hardware requirements are as follows,

1. PROCESSOR: PENTIUM IV
2. RAM: 8 GB
3. PROCESSOR: 2.4 GHZ
4. MAIN MEMORY: 8GB RAM
5. PROCESSING SPEED: 600 MHZ
6. HARD DISK DRIVE: 1TB

5.1.2 Software Requirements:

Software requirements deals with defining resource requirements and prerequisites that needs to be installed on a computer to provide functioning of an application. These requirements are need to be installed separately before the software is installed. The minimal software requirements are as follows,

1. FRONT END: PYTHON
2. IDE: ANACONDA
3. OPERATING SYSTEM: WINDOWS 10

5.1.3 Python Language

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc. Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

5.1.4 Python Programming Characteristics

- It provides rich data types and easier to read syntax than any other programming languages
- It is a platform independent scripted language with full access to operating system API's
- Compared to other programming languages, it allows more run-time flexibility
- It includes the basic text manipulation facilities of Perl and Awk
- A module in Python may have one or more classes and free functions

Applications of Python Programming

Web Applications

You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Some of the popular platforms for creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS. Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

Scientific and Numeric Computing

There are numerous libraries available in Python for scientific and numeric computing. There are libraries like: SciPy and NumPy that are used in general purpose computing. And, there are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on. Also, the language is heavily used in machine learning, data mining and deep learning.

Creating software Prototypes

Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited and efficiency is a must. However, Python is a great language for creating prototypes. For example: You can use Pygame (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.

Good Language to Teach Programming

Python is used by many companies to teach programming to kids and newbies. It is a good language with a lot of features and capabilities. Yet, it's one of the easiest languages to learn because of its simple easy-to-use syntax.

About OpenCV Package

Python is a general-purpose programming language started by Guido van Rossum, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working

in background) and second, it is very easy to code in Python. This is how OpenCV-Python works, it is a Python wrapper around original C++ implementation.

And the support of Numpy makes the task easier. Numpy is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

FEATURES OF ANACONDA NAVIGATOR

Anaconda is a free and open source, easy to install distribution of Python and R programming languages. Anaconda provides a working environment which is used for scientific computing, data science, statistical analysis and machine learning.

The latest distribution of Anaconda is Anaconda 5.3 and is released in October, 2018. It has the anaconda package, environment manager and a collection of 1000+ open source packages long with free community support.

Applications Provided in Anaconda Distribution

The Anaconda distribution comes with the following applications along with Anaconda Navigator.

1. JupyterLab
2. Jupyter Notebook

3. Qt Console

4. Spyder

5. Glueviz

6. Orange3

7. RStudio

8. Visual Studio Code

> JupyterLab: This is an extensible working environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

> Jupyter Notebook: This is a web-based, interactive computing notebook environment. We can edit and run human-readable docs while describing the data analysis.

> Qt Console: It is the PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips and more.

> Spyder: Spyder is a scientific Python Development Environment. It is a powerful Python IDE with advanced editing, interactive testing, debugging and introspection features.

> VS Code: It is a streamlined code editor with support for development operations like debugging, task running and version control.

> Glueviz: This is used for multidimensional data visualization across files. It explores relationships within and among related datasets.

> Orange 3: It is a component-based data mining framework. This can be used for data visualization and data analysis. The workflows in Orange 3 are very interactive and provide a large toolbox.

> Rstudio: It is a set of integrated tools designed to help you be more productive with R. It includes R essentials and notebooks.

5.2 Sample Code

```
import tensorflow as tf

from tensorflow.keras.layers import Conv2D, Input, ZeroPadding2D,
BatchNormalization, Activation, MaxPooling2D, Flatten, Dense

from tensorflow.keras.models import Model, load_model

from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint

from sklearn.model_selection import train_test_split

from sklearn.metrics import f1_score

from sklearn.utils import shuffle

import cv2

import imutils

import numpy as np

import matplotlib.pyplot as plt

import time

from os import listdir

%matplotlib inline
```

```

def crop_brain_contour(image, plot=False):

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

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

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

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

    # Find the extreme points
    extLeft = tuple(c[c[:, :, 0].argmin()][0])

```

```
extRight = tuple(c[c[:, :, 0].argmax()][0])
```

```
extTop = tuple(c[c[:, :, 1].argmin()][0])
```

```
extBot = tuple(c[c[:, :, 1].argmax()][0])
```

```
# crop new image out of the original image using the four extreme points (left, right,  
top, bottom)
```

```
new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]
```

```
if plot:
```

```
    plt.figure()
```

```
    plt.subplot(1, 2, 1)
```

```
    plt.imshow(image)
```

```
    plt.tick_params(axis='both', which='both',
```

```
                    top=False, bottom=False, left=False, right=False,
```

```
                    labelbottom=False, labeltop=False, labelleft=False, labelright=False)
```

```
    plt.title('Original Image')
```

```
    plt.subplot(1, 2, 2)
```

```
    plt.imshow(new_image)
```

```
    plt.tick_params(axis='both', which='both',
```

```
                    top=False, bottom=False, left=False, right=False,
```

```
                    labelbottom=False, labeltop=False, labelleft=False, labelright=False)
```

```
    plt.title('Cropped Image')
```

```
    plt.show()
```

```
return new_image
```

```

ex_img = cv2.imread('yes/Y1.jpg')
ex_new_img = crop_brain_contour(ex_img, True)
def load_data(dir_list, image_size):
    """
    Read images, resize and normalize them.
    Arguments:
        dir_list: list of strings representing file directories.
    Returns:
        X: A numpy array with shape = (#_examples, image_width, image_height,
        #_channels)
        y: A numpy array with shape = (#_examples, 1)
    """

    # load all images in a directory
    X = []
    y = []
    image_width, image_height = image_size

    for directory in dir_list:
        for filename in listdir(directory):
            # load the image
            image = cv2.imread(directory + '\\' + filename)
            # crop the brain and ignore the unnecessary rest part of the image
            image = crop_brain_contour(image, plot=False)
            # resize image

```

```

        image = cv2.resize(image, dsize=(image_width, image_height),
interpolation=cv2.INTER_CUBIC)

    # normalize values

    image = image / 255.

    # convert image to numpy array and append it to X
    X.append(image)

    # append a value of 1 to the target array if the image
    # is in the folder named 'yes', otherwise append 0.
    if directory[-3:] == 'yes':
        y.append([1])
    else:
        y.append([0])

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

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

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

return X, y

augmented_path = 'augmented data/'

```

```
# augmented data (yes and no) contains both the original and the new generated
examples
```

```
augmented_yes = augmented_path + 'yes'
```

```
augmented_no = augmented_path + 'no'
```

```
IMG_WIDTH, IMG_HEIGHT = (240, 240)
```

```
X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH,
IMG_HEIGHT))
```

```
def plot_sample_images(X, y, n=50):
```

```
    """
```

```
    Plots n sample images for both values of y (labels).
```

```
    Arguments:
```

```
    X: A numpy array with shape = (#_examples, image_width, image_height,
#_channels)
```

```
    y: A numpy array with shape = (#_examples, 1)
```

```
    """
```

```
    for label in [0,1]:
```

```
        # grab the first n images with the corresponding y values equal to label
```

```
        images = X[np.argwhere(y == label)]
```

```
        n_images = images[:n]
```

```
        columns_n = 10
```

```

rows_n = int(n/ columns_n)

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

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

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

    i += 1

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

```

```

"""

```

Splits data into training, development and test sets.

Arguments:

X: A numpy array with shape = (#_examples, image_width, image_height, #_channels)

y: A numpy array with shape = (#_examples, 1)

Returns:

X_train: A numpy array with shape = (#_train_examples, image_width, image_height, #_channels)

y_train: A numpy array with shape = (#_train_examples, 1)

X_val: A numpy array with shape = (#_val_examples, image_width, image_height, #_channels)

y_val: A numpy array with shape = (#_val_examples, 1)

X_test: A numpy array with shape = (#_test_examples, image_width, image_height, #_channels)

y_test: A numpy array with shape = (#_test_examples, 1)

"""

X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=test_size)

X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, test_size=0.5)

return X_train, y_train, X_val, y_val, X_test, y_test

X_train, y_train, X_val, y_val, X_test, y_test = split_data(X, y, test_size=0.3)

print ("number of training examples = " + str(X_train.shape[0]))

print ("number of development examples = " + str(X_val.shape[0]))

print ("number of test examples = " + str(X_test.shape[0]))

```

print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_val (dev) shape: " + str(X_val.shape))
print ("Y_val (dev) shape: " + str(y_val.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(y_test.shape))
# Nicely formatted time string
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return f"{h}:{m}:{round(s,1)}"

def compute_f1_score(y_true, prob):
    # convert the vector of probabilities to a target vector
    y_pred = np.where(prob > 0.5, 1, 0)

    score = f1_score(y_true, y_pred)

    return score

def build_model(input_shape):
    """
    Arguments:
        input_shape: A tuple representing the shape of the input of the model.
        shape=(image_width, image_height, #_channels)

    Returns:

```

```

        model: A Model object.
        """

        # Define the input placeholder as a tensor with shape input_shape.
        X_input = Input(input_shape) # shape=(?, 240, 240, 3)

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

plt.tick_params(axis='both', which='both',
                top=False, bottom=False, left=False, right=False,
                labelbottom=False, labeltop=False, labelleft=False, labelright=False)
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(new_image)
plt.tick_params(axis='both', which='both',
                top=False, bottom=False, left=False, right=False,
                labelbottom=False, labeltop=False, labelleft=False, labelright=False)
plt.title('Cropped Image')
plt.show()

return new_image

ex_img = cv2.imread('yes/Y1.jpg')
ex_new_img = crop_brain_contour(ex_img, True)
def load_data(dir_list, image_size):
    """

```

Read images, resize and normalize them.

Arguments:

dir_list: list of strings representing file directories.

Returns:

X: A numpy array with shape = (#_examples, image_width, image_height, #_channels)

y: A numpy array with shape = (#_examples, 1)

"""

load all images in a directory

X = []

y = []

image_width, image_height = image_size

for directory in dir_list:

for filename in listdir(directory):

load the image

image = cv2.imread(directory + '\\' + filename)

crop the brain and ignore the unnecessary rest part of the image

image = crop_brain_contour(image, plot=False)

resize image

image = cv2.resize(image, dsize=(image_width, image_height),
interpolation=cv2.INTER_CUBIC)

normalize values

image = image / 255.

```

    # convert image to numpy array and append it to X
    X.append(image)

    # append a value of 1 to the target array if the image
    # is in the folder named 'yes', otherwise append 0.
    if directory[-3:] == 'yes':
        y.append([1])
    else:
        y.append([0])

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

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

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

return X, y

augmented_path = 'augmented data/'

# augmented data (yes and no) contains both the original and the new generated
examples

augmented_yes = augmented_path + 'yes'

```

```

augmented_no = augmented_path + 'no'

IMG_WIDTH, IMG_HEIGHT = (240, 240)

X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH,
IMG_HEIGHT))

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

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

        columns_n = 10
        rows_n = int(n/ columns_n)

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

```

```

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

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

    i += 1

label_to_str = lambda label: "Yes" if label == 1 else "No"
plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
plt.show()

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

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

```

```

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

# FLATTEN X
X = Flatten()(X) # shape=(?, 6272)

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

# Create model. This creates your Keras model instance, you'll use this instance to
train/test the model.

model = Model(inputs = X_input, outputs = X, name='BrainDetectionModel')

return model

start_time = time.time()

plt.imshow(image)

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

plt.title('Original Image')

plt.subplot(1, 2, 2)

plt.imshow(new_image)

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

plt.title('Cropped Image')

```

```

plt.show()

return new_image

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

def load_data(dir_list, image_size):
    """
    Read images, resize and normalize them.

    Arguments:
        dir_list: list of strings representing file directories.

    Returns:
        X: A numpy array with shape = (#_examples, image_width, image_height,
        #_channels)
        y: A numpy array with shape = (#_examples, 1)
    """

    # load all images in a directory
    X = []
    y = []
    image_width, image_height = image_size

    for directory in dir_list:
        for filename in listdir(directory):
            # load the image
            image = cv2.imread(directory + '\\' + filename)

            # crop the brain and ignore the unnecessary rest part of the image

```

```

image = crop_brain_contour(image, plot=False)

# resize image
image = cv2.resize(image, dsize=(image_width, image_height),
interpolation=cv2.INTER_CUBIC)

# normalize values
image = image / 255.

# convert image to numpy array and append it to X
X.append(image)

# append a value of 1 to the target array if the image
# is in the folder named 'yes', otherwise append 0.
if directory[-3:] == 'yes':
    y.append([1])
else:
    y.append([0])

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

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

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

```

```

return X, y

augmented_path = 'augmented data/'

# augmented data (yes and no) contains both the original and the new generated
examples

augmented_yes = augmented_path + 'yes'
augmented_no = augmented_path + 'no'

IMG_WIDTH, IMG_HEIGHT = (240, 240)

X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH,
IMG_HEIGHT))

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

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

```

```

columns_n = 10
rows_n = int(n/ columns_n)

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

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

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

    i += 1

label_to_str = lambda label: "Yes" if label == 1 else "No"
plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
plt.show()

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

```

```

start_time = time.time()

model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

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

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

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

# crop new image out of the original image using the four extreme points (left, right,
top, bottom)

```

```

new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]

if plot:
    plt.figure()

    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.tick_params(axis='both', which='both',
                    top=False, bottom=False, left=False, right=False,
                    labelbottom=False, labeltop=False, labelleft=False, labelright=False)
    plt.title('Original Image')
    plt.subplot(1, 2, 2)
    plt.imshow(new_image)
    plt.tick_params(axis='both', which='both',
                    top=False, bottom=False, left=False, right=False,
                    labelbottom=False, labeltop=False, labelleft=False, labelright=False)
    plt.title('Cropped Image')
    plt.show()

return new_image

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

def load_data(dir_list, image_size):
    """
    Read images, resize and normalize them.

```

Arguments:

dir_list: list of strings representing file directories.

Returns:

X: A numpy array with shape = (#_examples, image_width, image_height, #_channels)

y: A numpy array with shape = (#_examples, 1)

"""

load all images in a directory

X = []

y = []

image_width, image_height = image_size

for directory in dir_list:

for filename in listdir(directory):

load the image

image = cv2.imread(directory + '\\' + filename)

crop the brain and ignore the unnecessary rest part of the image

image = crop_brain_contour(image, plot=False)

resize image

image = cv2.resize(image, dsize=(image_width, image_height),
interpolation=cv2.INTER_CUBIC)

normalize values

image = image / 255.

convert image to numpy array and append it to X

```

X.append(image)

# append a value of 1 to the target array if the image
# is in the folder named 'yes', otherwise append 0.
if directory[-3:] == 'yes':
    y.append([1])
else:
    y.append([0])

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

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

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

return X, y

augmented_path = 'augmented data/'

# augmented data (yes and no) contains both the original and the new generated
examples

augmented_yes = augmented_path + 'yes'
augmented_no = augmented_path + 'no'

```

```
IMG_WIDTH, IMG_HEIGHT = (240, 240)
```

```
X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH, IMG_HEIGHT))
```

```
def plot_sample_images(X, y, n=50):
```

```
    """
```

```
    Plots n sample images for both values of y (labels).
```

```
    Arguments:
```

```
        X: A numpy array with shape = (#_examples, image_width, image_height,
#_channels)
```

```
        y: A numpy array with shape = (#_examples, 1)
```

```
    """
```

```
    for label in [0,1]:
```

```
        # grab the first n images with the corresponding y values equal to label
```

```
        images = X[np.argwhere(y == label)]
```

```
        n_images = images[:n]
```

```
        columns_n = 10
```

```
        rows_n = int(n/ columns_n)
```

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

```
        i = 1 # current plot
```

```

for image in n_images:
    plt.subplot(rows_n, columns_n, i)
    plt.imshow(image[0])

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

    i += 1

label_to_str = lambda label: "Yes" if label == 1 else "No"
plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
plt.show()

plot_sample_images(X, y)

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

model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

end_time = time.time()

```

```

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

model.fit(x=X_train, y=y_train, batch_size=32, epochs=5, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
history = model.history.history
for key in history.keys():
    print(key)
def plot_metrics(history):

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

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

```

```

plt.legend()
plt.show()
# Accuracy
plt.figure()
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Accuracy')
end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
start_time = time.time()
model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])
end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
start_time = time.time()
model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

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

```

```
model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])
```

```
end_time = time.time()
```

```
execution_time = (end_time - start_time)
```

```
print(f"Elapsed time: {hms_string(execution_time)}")
```

```
start_time = time.time()
```

```
model.fit(x=X_train, y=y_train, batch_size=32, epochs=5, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])
```

```
end_time = time.time()
```

```
execution_time = (end_time - start_time)
```

```
print(f"Elapsed time: {hms_string(execution_time)}")
```

```
history = model.history.history
```

```
for key in history.keys():
```

```
    print(key)
```

```
def plot_metrics(history):
```

```
    train_loss = history['loss']
```

```
    val_loss = history['val_loss']
```

```
    train_acc = history['acc']
```

```
    val_acc = history['val_acc']
```

```
# Loss
```

```

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

# Accuracy
plt.figure()
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Accuracy')
plt.legend()
plt.show()
plot_metrics(history)
def data_percentage(y):

    m=len(y)

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

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

```

```

print(f"Number of examples: {m}")

print(f"Percentage of positive examples: {pos_prec}%, number of pos examples:
{n_positive}")

print(f"Percentage of negative examples: {neg_prec}%, number of neg examples:
{n_negative}")

start_time = time.time()

```

```

model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

```

```

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

```

```

model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

```

```

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

```

```

model.fit(x=X_train, y=y_train, batch_size=32, epochs=5, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

```

```

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

history = model.history.history

for key in history.keys():
    print(key)

def plot_metrics(history):

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

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

    # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')

```

```

plt.plot(val_acc, label='Validation Accuracy')
plt.title('Accuracy')
plt.legend()
plt.show()
plot_metrics(history)
print("Training Data:")
data_percentage(y_train)
print("Validation Data:")
data_percentage(y_val)
print("Testing Data:")
data_percentage(y_test)
print(f'Number of examples is: {len(X)}')
print(f'X shape is: {X.shape}')
print(f'y shape is: {y.shape}')

return X, y

augmented_path = 'augmented data/'

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

IMG_WIDTH, IMG_HEIGHT = (240, 240)

```

```
X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH,
IMG_HEIGHT))
```

```
def plot_sample_images(X, y, n=50):
```

```
    """
```

```
    Plots n sample images for both values of y (labels).
```

```
    Arguments:
```

```
    X: A numpy array with shape = (#_examples, image_width, image_height,
#_channels)
```

```
    y: A numpy array with shape = (#_examples, 1)
```

```
    """
```

```
    for label in [0,1]:
```

```
        # grab the first n images with the corresponding y values equal to label
```

```
        images = X[np.argwhere(y == label)]
```

```
        n_images = images[:n]
```

```
        columns_n = 10
```

```
        rows_n = int(n/ columns_n)
```

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

```
        i = 1 # current plot
```

```
        for image in n_images:
```

```
            plt.subplot(rows_n, columns_n, i)
```

```
            plt.imshow(image[0])
```

```

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

i += 1

label_to_str = lambda label: "Yes" if label == 1 else "No"
plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
plt.show()

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

start_time = time.time()

model.fit(x=X_train, y=y_train, batch_size=32, epochs=3, validation_data=(X_val,
y_val), callbacks=[tensorboard, checkpoint])

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

```

App.py

```
import dash
from dash.dependencies import Input, Output, State
import dash_core_components as dcc
import dash_html_components as html
from dash import no_update
import base64
import io
from io import BytesIO
import re
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, Conv3D, BatchNormalization,
Activation
from keras import backend as K
import os
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
from keras.models import load_model

classes = ['glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor']
```

```

enc = OneHotEncoder()
enc.fit([[0], [1], [2], [3]])
def names(number):
    if(number == 0):
        return 'a Glioma tumor'
    elif(number == 1):
        return 'a Meningioma tumor'
    elif(number == 2):
        return 'no tumor'
    elif(number == 3):
        return 'a Pituitary tumor'

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

server = app.server

app.layout = html.Div([

    html.H1(children='BRAIN TUMOR CLASSIFIER', style={'textAlign': 'center'
        })),

```

```
dcc.Markdown("""
    ##### Step 1: Import a single image using upload button
    ##### Step 2: Wait for prediction and interesting facts to appear
"""),
```

```
dcc.Upload(
    id='upload-image',
    children=html.Div([
        'Drag and Drop or ',
        html.A('Select Files')
    ]),
    style={
        'width': '95%',
        'height': '60px',
        'lineHeight': '60px',
        'borderWidth': '1px',
        'borderStyle': 'dashed',
        'borderRadius': '5px',
        'textAlign': 'center',
        'margin': '10px'
    },
    # Allow multiple files to be uploaded
```

```

        multiple=True
    ),
    html.Div(id='output-image-upload', style={'position':'absolute', 'left':'200px',
'top':'250px'}),

    html.Div(id='prediction', style={'position':'absolute', 'left':'800px', 'top':'310px',
'font-size':'x-large'}),
    html.Div(id='prediction2', style={'position':'absolute', 'left':'800px', 'top':'365px',
'font-size': 'x-large'}),

    html.Div(id='facts', style={'position':'absolute', 'left':'800px', 'top':'465px', 'font-
size': 'large',\
                                'height': '200px', 'width': '500px'}),

    ])

def parse_contents(contents):

    return html.Img(src=contents, style={'height':'450px', 'width':'450px'})

@app.callback([Output('output-image-upload', 'children'), Output('prediction',
'children'), Output('prediction2', 'children'),
                Output('facts', 'children')],
                [Input('upload-image', 'contents')])

```

```

def update_output(list_of_contents):

    if list_of_contents is not None:
        children = parse_contents(list_of_contents[0])

        img_data = list_of_contents[0]
        img_data = re.sub('data:image/jpeg;base64,', '', img_data)
        img_data = base64.b64decode(img_data)

        stream = io.BytesIO(img_data)
        img_pil = Image.open(stream)

        #Load model, change image to array and predict
        model = load_model(r'C:\Users\cse\Documents\brain-tumor-classifier-app-master\model_final.h5')
        dim = (150, 150)

        img = np.array(img_pil.resize(dim))

        x = img.reshape(1,150,150,3)

        answ = model.predict(x)
        classification = np.where(answ == np.amax(answ))[1][0]

```

```
pred=str(round(answ[0][classification]*100 ,3)) + '% confidence there is ' +  
names(classification)
```

```
#Second prediction and facts about tumor if there is
```

```
if classification==0:
```

```
facts = 'Glioma is a type of tumor that occurs in the brain and spinal cord.\
```

```
A glioma can affect your brain function and be life-threatening  
depending on\
```

```
its location and rate of growth.'
```

```
no_tumor = str(round(answ[0][2]*100 ,3))
```

```
pred2 = no_tumor + '% confidence there is no tumor'
```

```
elif classification==1:
```

```
facts = 'A meningioma is a tumor that arises from the meninges, the  
membranes that surround your brain.\
```

```
Most meningiomas grow very slowly, often over many years without  
causing symptoms.'
```

```
no_tumor = str(round(answ[0][2]*100 ,3))
```

```
pred2 = no_tumor + '% confidence there is no tumor'
```

```
elif classification==3:
```

```
facts = 'Pituitary tumors are abnormal growths that develop in your pituitary  
gland.\
```

```
Most pituitary tumors are noncancerous (benign) growths that remain in  
your pituitary\
```

```
gland or surrounding tissues.'
```

```

no_tumor = str(round(answ[0][2]*100 ,3))
pred2 = no_tumor + '% confidence there is no tumor'

else:

    facts=None
    pred2 = None

return children, pred, pred2, facts

else:

    return (no_update, no_update, no_update, no_update)

if __name__ == '__main__':
    app.run_server(debug=True)

```

5.3 Training Data

```

import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

import pickle

pickle_in = open("XX.pickle","rb")
X = pickle.load(pickle_in)

```

```
pickle_in = open("YY.pickle","rb")
Y = pickle.load(pickle_in)

X = X/255.0
model = Sequential()

model.add(Conv2D(256,(3,3), input_shape=X.shape[1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(256,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Dense(1))
model.add(Activation('sigmoid'))

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

```
model.fit(X,y, batch_size=164, epochs=10, validation_split=0.3)
```

```
In [9]: 1 model.fit(X, y, batch_size=164, epochs=10, validation_split=0.3)

Train on 1370 samples, validate on 588 samples
Epoch 1/10
1370/1370 [=====] - 178s 130ms/sample - loss: 1.4922 - acc: 0.5898 - val_loss: 0.5774 - val_acc: 0.721
1
Epoch 2/10
1370/1370 [=====] - 151s 110ms/sample - loss: 0.5444 - acc: 0.7292 - val_loss: 0.5420 - val_acc: 0.755
1
Epoch 3/10
1370/1370 [=====] - 151s 110ms/sample - loss: 0.5002 - acc: 0.7723 - val_loss: 0.7345 - val_acc: 0.596
9
Epoch 4/10
1370/1370 [=====] - 153s 112ms/sample - loss: 0.5314 - acc: 0.7438 - val_loss: 0.5976 - val_acc: 0.688
8
Epoch 5/10
1370/1370 [=====] - 151s 110ms/sample - loss: 0.4734 - acc: 0.7759 - val_loss: 0.5356 - val_acc: 0.767
0
Epoch 6/10
1370/1370 [=====] - 143s 105ms/sample - loss: 0.4226 - acc: 0.8139 - val_loss: 0.5307 - val_acc: 0.768
7
Epoch 7/10
1370/1370 [=====] - 151s 110ms/sample - loss: 0.3806 - acc: 0.8314 - val_loss: 0.5076 - val_acc: 0.787
4
Epoch 8/10
1370/1370 [=====] - 143s 104ms/sample - loss: 0.3452 - acc: 0.8518 - val_loss: 0.4853 - val_acc: 0.784
0
Epoch 9/10
1370/1370 [=====] - 152s 111ms/sample - loss: 0.3174 - acc: 0.8672 - val_loss: 0.5170 - val_acc: 0.792
5
Epoch 10/10
1370/1370 [=====] - 147s 107ms/sample - loss: 0.2780 - acc: 0.8839 - val_loss: 0.4818 - val_acc: 0.797
6

Out[9]: <tensorflow.python.keras.callbacks.History at 0x1ca5debd108>
```

Figure 5.3.1 Training Data History

5.4 Data Load

```
import numpy as np
```

```
import
```

```
matplotlib.pyplot as plt
```

```
import os
```

```
import cv2
```

```
from tqdm import tqdm
```

```
DATADIR = "D:\dataset1"
```

```
CATEGORIES = ["no", "yes"]
```

```
for category in CATEGORIES:
```

```

path = os.path.join(DATADIR,category)

for img in os.listdir(path):

img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)

    # convert to array

    plt.imshow(img_array, cmap='gray')

    # graph itplt.show() # display!

    Break # we just want one for now so break

    break #...and one more!

print(img_array)
print(img_array.shape)

    IMG_SIZE = 100
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    plt.imshow(new_array, cmap='gray')

    plt.show()

training_data = []

def create_training_data():

    for category in CATEGORIES:

        path = os.path.join(DATADIR,category) # create path

        class_num = CATEGORIES.index(category) # get the classification (0 or
        a 1)

        for img in tqdm(os.listdir(path)): # iterate over

            each image:

```

```

        img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE) # convert to array

        new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE)) #
resize tonormalize data size
        training_data.append([new_array,class_num]) # add this
to our training_data

create_training_data()

print(training_data)

import random

random.shuffle(training_data)

for sample in training_data[10]:
    print(sample[1])

X = []
Y = []

For feature_labels in training_data

    X.append(features)

    y.append(label)

print(X[0].reshape(-1,IMG_SIZE, IMG_SIZE, 1))

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)

pickle_out = open("YY pickle","wb")

pickle.dump(y, pickle_out)

pickle_out.close()

```

5.5 Performance Measures:

The proposed algorithm has been assessed through various performance evaluation metrics that include True Positive, True Negative the former one that designates how many times does the proposed algorithm is able to correctly recognize the damaged region as damaged region and the later one designates how many times does the proposed algorithm correctly identified non-damaged region as non-damaged region. And the False Positive (FP) and False Negative (FN) the former one designates how many times does the proposed algorithm fails to recognize the damaged region correctly, and the later represents how many times does the proposed algorithm fails to identify the non-tumors region as non-tumors regions. Basing on values of TP, TN, FP, and FN, the values of Accuracy, Specificity and sensitivity are calculated of the proposed algorithm.

$$\text{Accuracy} = (TP + TN) / (TP + FP + FN + TN)$$

$$\text{Specificity} = TN / (TN + FP)$$

$$\text{Sensitivity} = TP / (TP + FN)$$

5.6 Performance Evaluation

On experimentation, it was observed that the proposed methodology seems to be outperformed when compared to all different set of images. Among all the images, the proposed Convolutional Neural Network (CNN) based approach seems too much better in terms of quality

of the output in 128 *128 images when compared to its other sized images which are represented in table and charts.

TABLE 1 Represents the true positive, true negative, false positive and false negative values of the proposed approach for different set of images.

Different set of Images	True Positive (%)	True Negative (%)	False Positive (%)	False Negative (%)
128 * 128 Images	83.7	84.5	16.3	15.5
256 * 256 Images	82.4	84.1	17.6	15.9
512 * 512 Images	82.1	83.7	17.9	16.3

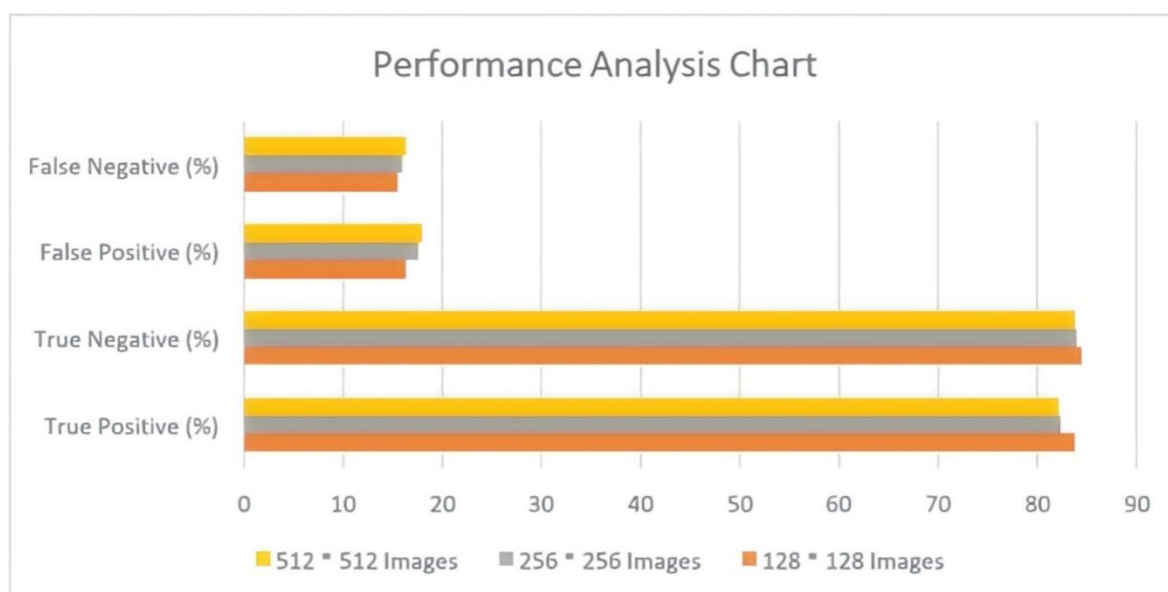


Figure 5.6.1 Represents the performance analysis of CNN

It is observed from table 2 upon performing proposed segmentation technique for different set of images that have the ability to recognize the isolated region from the MR images that are used to analyze the shape and size of the denoised image. We have used Convolutional Neural Network (CNN) for segmentation, and the output of our proposed work is pleased with better accuracy, sensitivity, and computational time.

TABLE 2 Represents the Accuracy, Sensitivity, and Specificity of the proposed approach for different set of images.

Different Set of images	Accuracy (%)	Sensitivity (%)	Specificity (%)
128 * 128 Images	84.1	84.4	83.8
256 * 256 Images	83.3	83.4	82.7
512 * 512 Images	82.9	83.3	82.4

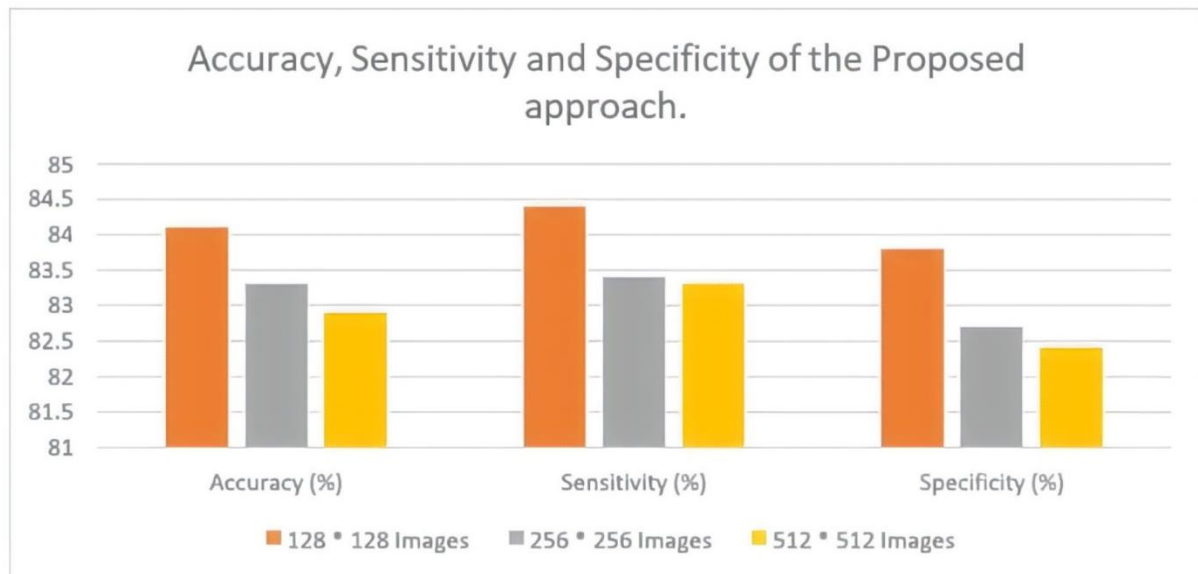


Figure 5.6.2 Represents the performance of proposed CNN

5.7 Output

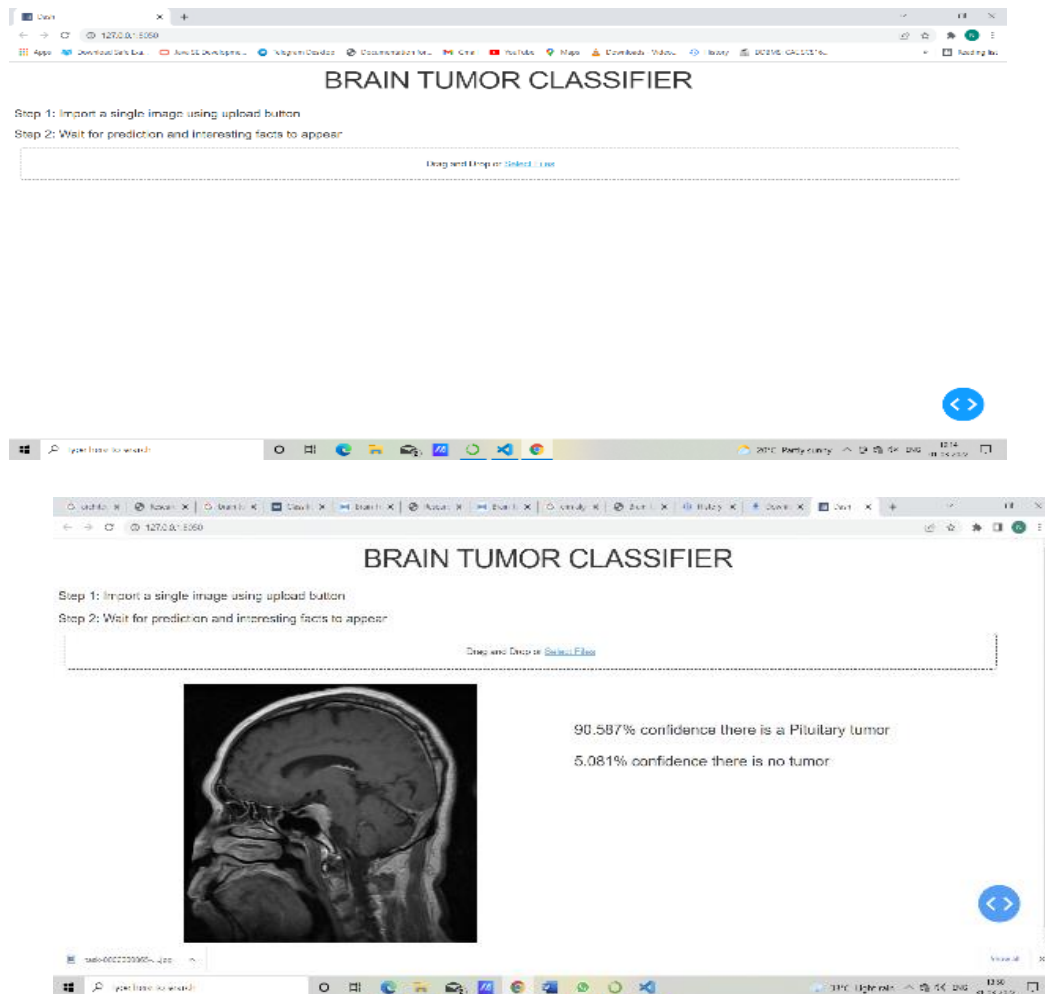


Figure 5.7.1 Pituitary Tumor

Pituitary tumors are abnormal growths that develop in your pituitary gland. Most pituitary tumors are noncancerous (benign) growths that remain in your pituitary gland or surrounding tissues.

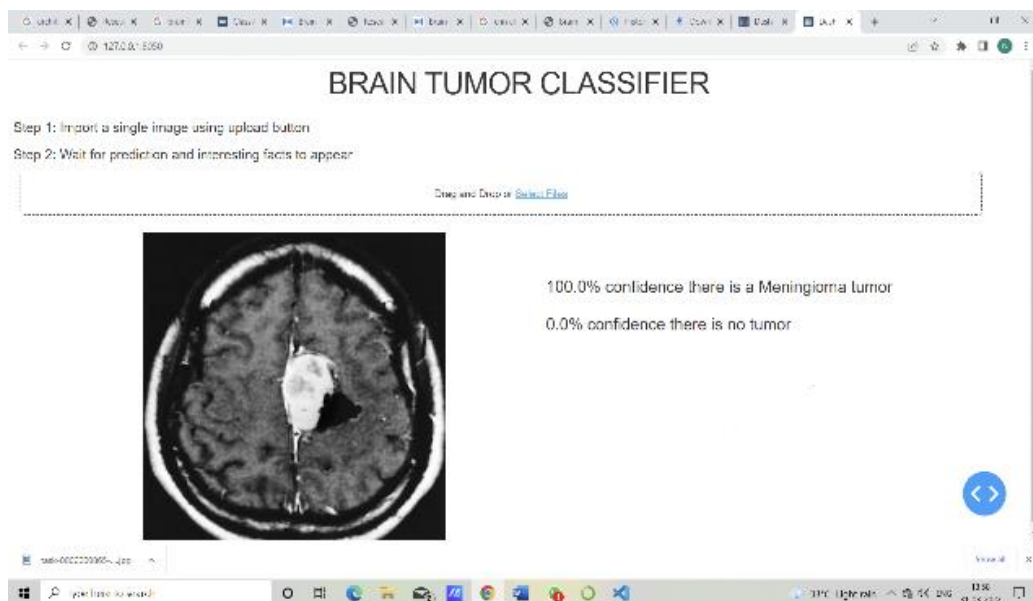


Figure 5.7.2 Meningioma Tumor

A meningioma is a tumor that arises from the meninges, the membranes that surround your brain. Most meningiomas grow very slowly, often over many years without causing symptoms.

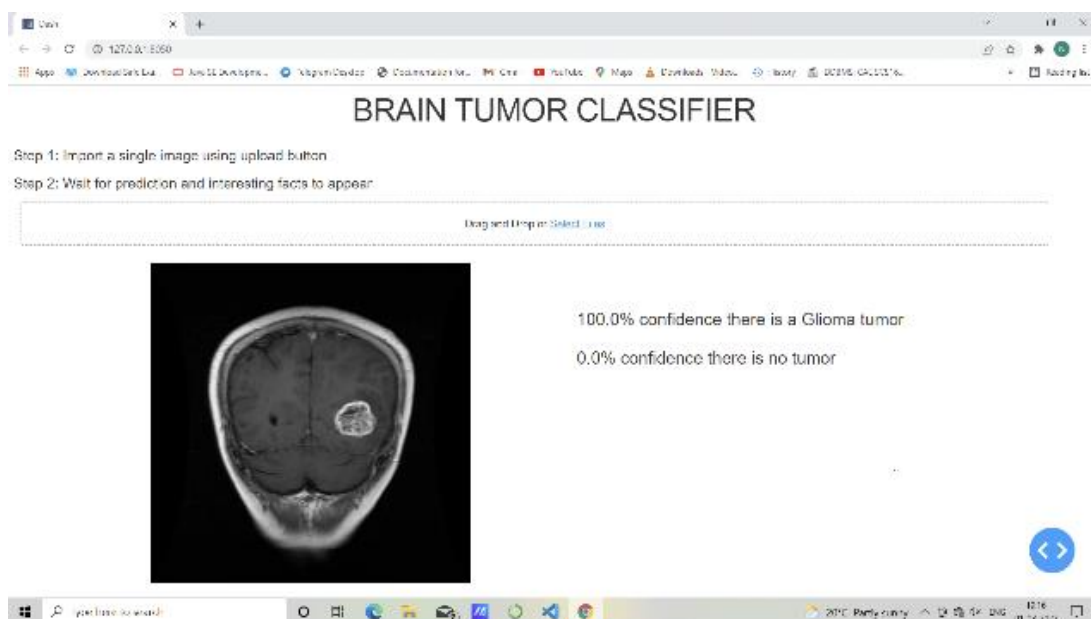


Figure 5.7.3 Glioma Tumor

Glioma is a type of tumor that occurs in brain and spinal cord. A glioma can affect your brain function and be life threatening depending on its location and rate of growth.

5.8 Result

Algorithms for analysing and classifying medical images have gained a great level of attention recently. The experiments we present in this work show that after pre-processing MRI images, Capsule Network classification algorithm was the best. A much higher accuracy can be achieved by gaining a better dataset with high-resolution MRI images. This may be extended this feature extraction process in terms of deep learning neural network. Faster CNN Convolution neural networks are the most delicate network for the field of image processing and also the algorithm need to be extended for segmenting the tumorous area.

CHAPTER 6

Summary and Conclusions

We proposed a computerized method for the segmentation and identification of a brain tumor using the Convolution Neural Network. The input MR images are read from the local device using the file path and converted into grayscale images. These images are pre-processed using an adaptive bilateral filtering technique for the elimination of noises that are present inside the original image. The binary thresholding is applied to the denoised image, and Convolution Neural Network segmentation is applied, which helps in figuring out the tumor region in the MR images. The proposed model had obtained an accuracy of 84% and yields promising results without any errors and much less computational time.

It is observed on experimentation that the proposed approach needs a vast training set for better accurate results; in the field of medical image processing, the gathering of medical data is a tedious job, and, in few cases, the datasets might not be available. In all such cases, the proposed algorithm must be robust enough for accurate recognition of tumor regions from MR Images. The proposed approach can be further improvised through in cooperating weakly trained algorithms that can identify the abnormalities with a minimum training data and also self-learning algorithms would aid in enhancing the accuracy of the algorithm and reduce the computational time.

REFERENCE

- [1] Brain cancer Detection and Identification of MRI Brain Image Using Different Wavelet Transforms and help Vector Machines," 2019 42nd International Conference on Telecommunications and Signal Processing, M. Gurbină, M. Lascu, and D. Lascu (TSP), 2019, pp. 505-508, doi: 10.1109/TSP.2019.8769040.

- [2] Watershed Algorithm based DAPP features for Brain Tumor Classification and Segmentation, T. A. Jemimma and Y. J. Vetharaj, 2018Conference on global Smart Systems and Inventive Technology (ICSSIT), pp. 155-158, doi: 10.1109/ICSSIT.2018.8748436.

- [3] H. T. Zaw, N. Maneerat and K. Y. Win, "Brain tumor detection based on Naïve Bayes Classification," 2019 5th Global Conference on Engineering, Applied Sciences and Technology (ICEAST), 2019, pp. 1-4, doi: 10.1109/ICEAST.2019.8802562

- [4] M. C. Xenya and Z. Wang, "Brain Cancer Detection Using Multi-level Ensemble Transfer Learning in MRI Dataset," 2021 International Conference on Ai Technology, Big Data, Computing and Data Communication Systems, (icABCD), 2021, pp. 1-7, doi: 10.1109/icABCD51485.2021.9519361.

- [5] A. Hussain and A. Khunteta, "Semantic Segmentation of Brain Tumor from Mri Scans and SVM Classification using GLCM Features," 2ndInternational Conference on Inventive Studies in

Computing Applications, 2020. (ICIRCA), 2020, pp. 38-43, doi: 10.1109/ICIRCA48905.2020.9183385.

- [6] K. T.K. and S. Xavier, "An Smart Method for Evaluation and Classification of Brain Tumor," Second International Conference on Incredibly creative Communication and Computational Technologies, 2018. (ICICCT), 2018, pp. 1265-1268, doi: 10.1109/ICICCT.2018.8473297
- [7] Brain Tumor Detection from MRI Images Using Naive Classifier," 2020 6th International Conference on Modern Computing and Communication Systems, D. Divyamary, S. Gopika, S. Pradeeba, and M. Bhuvaneswari. (ICACCS), 2020, pp. 620-622, doi: 10.1109/ICACCS48705.2020.9074213.
- [8] "Multi - class classification Brain Tumor Convolutional Neural Network and Support Vector Machine," 2021 Mohammad Ali Jinnah University International Conference on Computing, H. Kibriya, M. Masood, M. Nawaz, R. Rafique, and S. Rehman. (MAJICC), 2021, pp. 1-4, doi: 10.1109/MAJICC53071.2021.9526262.
- [9] L. A. Akter and G. R. Kwon (2018). The Contourlet Transform and the Canny Edge Detector are used to segment brain images. The International Conference on Ubiquitous and Future Networks (ICUFN) will take place in 2018. (pp. 798-800). IEEE
- [10] Md. Rezwanul Islam, Md. Reezbhan Imteaz, and Marium-E-Jannat "Detection and analysis of brain tumours using MRI using a Histogram-Based Method that Combines Thresholding and

Morphological Processes" 2018 Global Conference on Computer,
Communication, Chemical, Material, and Electronic Engineering