A Project Review-ETE Report

on

Brain Tumor Detection Using Convolutional Neural Network

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Mr. Deependra Rastogi

Submitted By

Avdeep Malik
19SCSE1010214
Shubham Upadhyay
19SCSE1010467

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
December, 2021

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project, entitled **"Brain Tumor Detection Using Convolutional Neural Network"** in partial fulfillment of the requirements for the award of the B,Tech. submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of October, 2021 to December, 2021 under the supervision of **Mr Deependra Rastogi**, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by us for the award of any other degree of this or any other places.

Avdeep Malik, 19SCSE1010214
Shubham Upadhyay, 19SCSE1010467

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr Deependra Rastogi

CERTIFICATE

The Final Project Viva-Voce examination of Avdeep Malik: 19SCSE1010214, Shubham Upadhyay: 19SCSE1010467 has been held on 26 December Year 2021 and Their work is recommended for the award of B.Tech.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

Abstract

The human brain is the major controller of the humanoid system. The abnormal growth and division of cells in the brain lead to a brain tumor, and the further growth of brain tumors leads to brain cancer. In the area of human health, Computer Vision plays a significant role, which reduces the human judgment that gives accurate results. CT scans, X-Ray, and MRI scans are the common imaging methods among magnetic resonance imaging (MRI) that are the most reliable and secure. MRI detects every minute objects. Our paper aims to focus on the use of different techniques for the discovery of brain cancer using brain MRI. In this study, we performed pre-processing using the bilateral filter (BF) for removal of the noises that are present in an MR image. This was followed by the binary thresholding and Convolution Neural Network (CNN) segmentation techniques for reliable detection of the tumor region. Training, testing, and validation datasets are used. Based on our machine, we will predict whether the subject has a brain tumor or not. The resultant outcomes will be examined through various performance examined metrics that include accuracy, sensitivity, and specificity. It is desired that the proposed work would exhibit a more exceptional performance over its counterparts.

KEYWORDS: Brain tumor, Magnetic resonance imaging, Adaptive Bilateral Filter, Convolution Neural Network.

Contents

List of Table

List of Figures

CHAPTER-1 Introduction

Medical imaging is the technique and process of creating visual representations of the interior of a body for clinical analysis and medical intervention, as well as visual representation of the function of some organs or tissues. Medical imaging seeks to reveal internal structures hidden by the skin and bones, as well as to diagnose and treat disease. Medical imaging also establishes a database of normal anatomy and physiology to make it possible to identify abnormalities.

The medical imaging processing refers to handling images by using the computer. This processing includes many types of techniques and operations such as image gaining, storage, presentation, and communication. This process pursues the disorder identification and management. This process creates a data bank of the regular structure and function of the organs to make it easy to recognize the anomalies. This process includes both organic and radiological imaging which used electromagnetic energies (X-rays and gamma), sonography, magnetic, scopes, and thermal and isotope imaging. There are many other technologies used to record information about the location and function of the body. Those techniques have many limitations compared to those modulates which produce images.

An image processing technique is the usage of a computer to manipulate the digital image. This technique has many benefits such as elasticity, adaptability, data storing, and communication. With the growth of different image resizing techniques, the images can be kept efficiently. This technique has many sets of rules to perform in the images synchronously. The 2D and 3D images can be processed in multiple dimensions.

PROBLEM STATEMENT:

Our study deals with automated brain tumor detection and classification. Normally the anatomy of the brain is analyzed by MRI scans or CT scans. The aim of the paper is tumor identification in brain MR images. The main reason for detection of brain tumors is to provide aid to clinical diagnosis. The aim is to provide an algorithm that guarantees the presence of a tumor by combining several procedures to provide a foolproof method of tumor detection in MR brain images. The methods utilized are filtering, erosion, dilation, threshold, and outlining of the tumor such as edge detection.

The focus of this project is MR brain images tumor extraction and its representation in simpler form such that it is understandable by everyone. The objective of this work is to bring some useful information in simpler form in front of the users, especially for the medical staff treating the patient. The aim of this work is to define an algorithm that will result in extracted image of the tumor from the MR brain image. The resultant image will be able to provide information like size, dimension and position of the tumor, and its boundary provides us with information related to the tumor that can prove useful for various cases, which will provide a better base for the staff to decide the curing procedure. Finally, we detect whether the given MR brain image has tumor or not using Convolution Neural Network.

Tool and Technology Used

Python:
Python is an interpreted, high-level, general purpose programming language created by Guido Van Rossum and first released in 1991, Python's design philosophy emphasizes code Readability with its notable use of significant Whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

PIP:
It is the package management system used to install and manage software packages written in Python.

NumPy:
NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:
- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Pandas:
Pandas is the most popular python library that is used for data analysis. It provides highly optimized performance with back-end source code is purely written in *C* or *Python*. We can analyze data in pandas with

1. Series
2. Data frames

Tensor Flow:
Tensor flow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

Keras:
Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

OpenCV:
OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by willow garage then Itseez (which was later acquired by Intel). The library is cross platform and free for use under the open source BSD license. OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers. It promotes Open Vision Capsules. which is a portable format, compatible with all other formats.

CHAPTER-2
Reference Publication

In Medical diagnosis, robustness and accuracy of the prediction algorithms are very important, because the result is crucial for treatment of patients. There are many popular classification and clustering algorithms used for prediction. The goal of clustering a medical image is to simplify the representation of an image into a meaningful image and make it easier to analyze. Several Clustering and Classification algorithms are aimed at enhancing the prediction accuracy of diagnosis process in detecting abnormalities.

In the literature survey we provide a brief summary of the different methods that have been proposed for clustering over the period of 2002 to 2018.We have been though 10 papers each of which has a unique approach towards segmentation in some parameter or the other. The summaries of each of the papers are provided below.

 A. Sivaramakrishnan And Dr. M. Karnan "A Novel Based Approach for Extraction Of Brain Tumor In MRI Images Using Soft Computing Techniques," International Journal Of Advanced Research In Computer And Communication Engineering, Vol. 2, Issue 4, April 2013.

A. Sivaramakrishnan et al. (2013) [1] projected an efficient and innovative discovery of the brain tumor vicinity from an image that turned into finished using the Fuzzy Capproach grouping algorithm and histogram equalization. The disintegration of images is achieved by the usage of principal factor evaluation is done to reduce the extent of the wavelet coefficient. The outcomes of the anticipated FCM clustering algorithm accurately withdrawn tumor area from the MR images.

 Asra Aslam, Ekram Khan, M.M. Sufyan Beg, Improved Edge Detection Algorithm for Brain Tumor Segmentation, Procedia Computer Science, Volume 58,2015, Pp 430-437, ISSN 1877-0509.

M. M. Sufyan et al. [2] has presented a detection using enhanced edge technique for brain-tumor segmentation that mainly relied on Sobel feature detection. Their presented work associates the binary thresholding operation with the Sobel approach and excavates diverse extents using a secure contour process. After the

completion of that process, cancer cells are extracted from the obtained picture using intensity values.

 B.Sathya and R.Manavalan, Image Segmentation by Clustering Methods: Performance Analysis, International Journal of Computer Applications (0975 – 8887) Volume 29– No.11, September 2011.

Sathya et al. (2011) [3], provided a different clustering algorithm such as K-means, Improvised K-means, C-means, and improvised C-means algorithms. Their paper presented an experimental analysis for massive dat=asets consisting of unique photographs. They analyzed the discovered consequences using numerous parametric tests.

Devkota, B. & Alsadoon, Abeer & Prasad, P.W.C. & Singh, A.K. & Elchouemi, A. (2018). Image Segmentation for Early Stage Brain Tumor Detection using Mathematical Morphological Reconstruction. Procedia Computer Science. 125. 115-123. 10.1016/j.procs.2017.12.017.
B.

Devkota et al. [4] have proposed that a computer-aided detection (CAD) approach is used to spot abnormal tissues via Morphological operations. Amongst all different segmentation approaches existing, the morphological opening and closing operations are preferred since it takes less processing time with the utmost efficiency in withdrawing tumor areas with the least faults.

 K. Sudharani, T. C. Sarma and K. Satya Rasad, "Intelligent Brain Tumor lesion classification and identification from MRI images using a K-NN technique," 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil, 2015, pp. 777-780. DOI: 10.1109/ICCICCT.2015.7475384

K. Sudharani et al. [5] presented a K- nearest neighbor algorithm to the MR images to identify and confine the hysterically full-fledged part within the abnormal tissues. The proposed work is a sluggish methodology but produces exquisite effects. The accuracy relies upon the sample training phase.

 Kaur, Jaskirat & Agrawal, Sunil & Renu, Vig. (2012). A Comparative Analysis of Thresholding and Edge Detection Segmentation Techniques. International Journal of Computer Applications.vol. 39.pp. 29-34. 10.5120/4898-7432.

Jaskirat Kaur et al. (2012) [6] defined a few clustering procedures for the segmentation process and executed an assessment on distinctive styles for those techniques. Kaur represented a scheme to measure selected clustering techniques based on their steadiness in exceptional tenders. They also defined the diverse performance metric tests, such as sensitivity, specificity, and accuracy.

 Li, Shutao, JT-Y. Kwok, IW-H. Tsang and Yaonan Wang. "Fusing images with different focuses using support vector machines." IEEE Transactions on neural networks 15, no. 6 (2004): 1555-1561.

J.T. Kwok et al. [7] delivered wavelet-based photograph fusion to easily cognizance at the object with all focal lengths as several vision-related processing tasks can be carried out more effortlessly when wholly substances within the images are bright. In their work Kwok et al. investigated with different datasets, and results show that presented work is extra correct as it does not get suffering from evenness at different activity stages computations.

 M. Kumar and K. K. Mehta, "A Texture based Tumor detection and automatic Segmentation using Seeded Region Growing Method," International Journal of Computer Technology and Applications, ISSN: 2229-6093, Vol. 2, Issue 4, PP. 855-859 August 2011.

Kumar and Mehta [8] proposed the texture-based technique in this paper. They highlighted the effects of segmentation if the tumor tissue edges aren't shrill. The performance of the proposed technology may get unwilling results due to those edges. The texture evaluation and seeded region approach turned into executed inside the MATLAB environment.

 Mahmoud, Dalia & Mohamed, Eltaher. (2012). Brain Tumor Detection Using Artificial Neural Networks. Journal of Science and Technology. 13. 31-39.

Dalia Mahmoud et al. [9] presented a model using Artificial Neural Networks for tumor detection in brain images. They implemented a computerized recognition system for MR imaging the use of Artificial Neural Networks. That was observed that after the Elman community was used during the recognition system, the period time and the accuracy level were high, in comparison with other ANNs systems. This neural community has a sigmoid characteristic which elevated the extent of accuracy of the tumor segmentation.

 Marroquin J.L., Vemuri B.C., Botello S., Calderon F. (2002) An Accurate and Efficient Bayesian Method for Automatic Segmentation of Brain MRI. In: Heyden A., Sparr G., Nielsen M., Johansen P. (eds) Computer Vision − ECCV 2002. ECCV 2002. Lecture Notes in Computer Science, vol 2353. Springer, Berlin, Heidelberg.

L. Marroquin et al. [10] presented the automated 3d segmentation for brain MRI scans. Using a separate parametric model in preference to a single multiplicative magnificence will lessen the impact on the intensities of a grandeur. Brain atlas is hired to find nonrigid conversion to map the usual brain. This transformation is further used to segment the brain from nonbrain tissues, computing prior probabilities and finding automatic initialization and finally applying the MPM-MAP algorithm to find out optimal segmentation. Major findings from the study show that the MPM-MAP algorithm is comparatively robust than EM in terms of errors while estimating the posterior marginal. For optimal segmentation, the MPM-MAP algorithm involves only the solution of linear systems and is therefore computationally efficient.

CHAPTER-3
Functionality/Working of Project

CHALLENGES IN TUMOR CLASSIFICATION

The identification of tumor is a very challenging task. The location, shape and the structure of tumor varies significantly from patient to patient which makes the segmentation a very challenging task. In the figure shown below, we have shown some images of the same brain slice from different patients, which clearly reflect the variation of the tumor. We can clearly see that the location of the tumor is different in all the 8 images/patients shown below. To make it worse, the shape and the intra-tumoral structure is also different for all the eight patients/images. In fact, there can be more than one region of the tumor as can be seen from the images below. This indeed reflects the complexity of automatic segmentation.

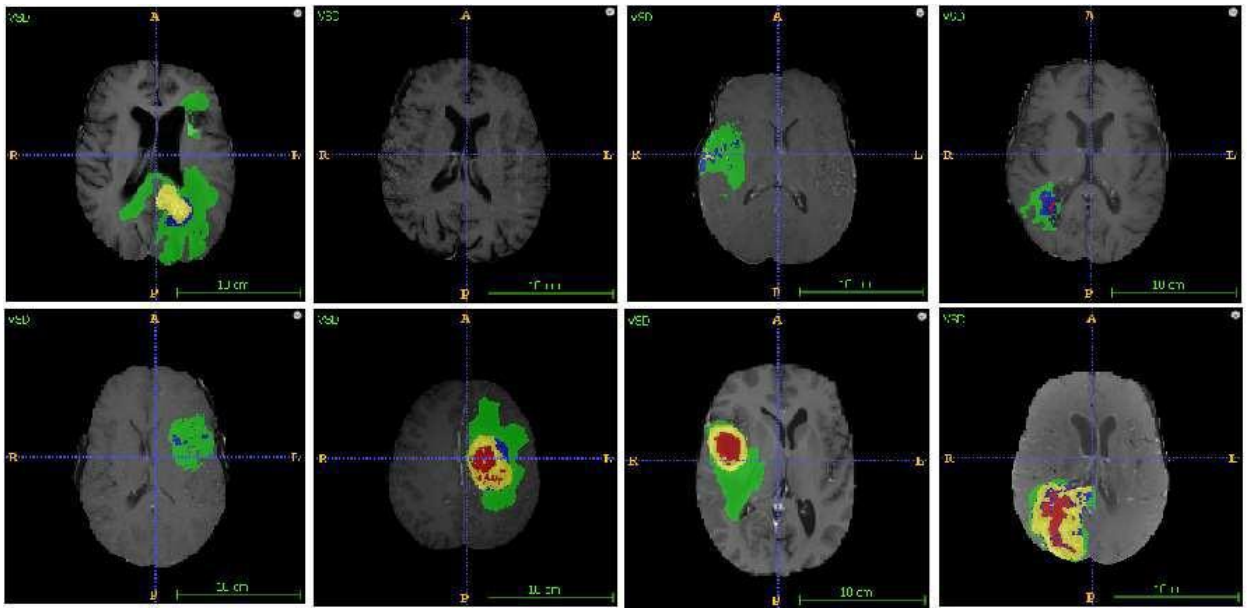
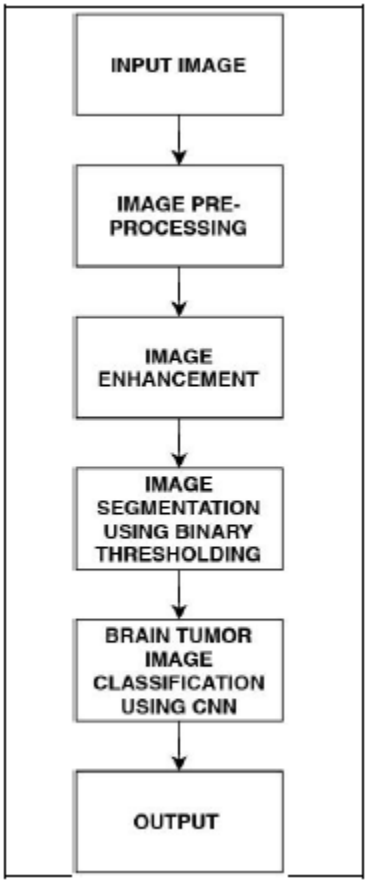


Fig.3.1 Location of tumors in eight different images.

MODULE DIVISON

This provides the architecture of the system that would be developed by our hands. It consists of six steps where the execution starts from taking an input image from the data set followed by the image pre-processing, image enhancement, Image segmentation using binary thresholding and the brain tumor classification using Convolutional Neural Network. Finally, the output is observed after all the above-mentioned steps are completed.

Each module is unique in its own way. Every step has its importance. This architecture also includes a testing and training data set. The data set used is has been downloaded from Kaggle which consists of nearly 2000 images that are used to test and train the system. The input image is pre-processed by using the noise filter like Median Filter and Bilateral Filter and the image is enhanced using the Sobel Filter. Then the obtained image using segmented using binary thresholding and morphological operations are performed on it. Finally, the image classification is done using Convolutional Neural Network to predict whether the tumor is present or not.



Module Division

MODULE 1: IMAGE PREPROCESSING AND IMAGE ENHANCEMENT

IMAGE PREPROCESSING:

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.

IMAGE ACQUISITION FROM DATASET:

In image processing, image acquisition is done by retrieving an image from dataset for processing. It is the first step in the workflow sequence because, without an image no processing is possible. The image that is acquired is completely unprocessed.
Here we process the image using the file path from the local device.

CONVERT THE IMAGE FROM ONE COLOR SPACE TO ANOTHER:

There are more than 150 color-space conversion methods available in OpenCV. For color conversion, we use the function cv2.cvtColor(input_image, flag) where flag determines the type of conversion. In our work, we convert the input image into the gray-scale image.

FILTERS:

In image processing, filters are mainly used to suppress the high frequencies in the image.

Median filter: It is a non-linear filtering technique used to remove noise from the images. It is performed by sorting all the pixel values from the window into numerical order and then replacing the pixel being considered with the median pixel value. This filter removes the speckle noise and salt and pepper noise through ‘ON’ and ‘OFF’ of pixels by white and dark spots.

Bilateral filter: It is also a non-linear, noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels. This weight is based on the Gaussian distribution. Bilateral

filtering smooth images while conserving edges utilizing a nonlinear grouping of neighbouring image pixels. This filtering technique is simple, local, and concise. It syndicates a grey level grounded on their likeness and the symmetrical nearness and chooses near vales to farther values in both range and domain.

IMAGE ENHANCEMENT:

Image enhancement is a technique used to improve the image quality and perceptibility by using computer-aided software. This technique includes both objective and subjective enhancements. This technique includes points and local operations. The local operations depend on the district input pixel values. Image enhancement has two types: spatial and transform domain techniques. The spatial techniques work directly on the pixel level, while the transform technique works on Fourier and later on the spatial technique.

Edge detection is a segmentation technique that uses border recognition of strictly linked objects or regions. This technique identifies the discontinuity of the objects. This technique is used mainly in image study and to recognize the parts of the image where a huge variation in intensity arises.

SOBEL FILTER:

The Sobel filter is used for edge detection. It works by calculating the gradient of image intensity at each pixel within the image. It is widely used in image analysis to help locate edges in images. Sobel operator is used for segmentation purpose. This technique can be dependent on the central difference which tends toward the central pixels on average. This technique can be expressed as 3 × 3 matric to the first derivative of the Gaussian kernel. It combines smoothing and differentiation. For Sobel edge detection the gradient of the image is calculated for each pixel position in the image.

1. We calculate two derivatives:

a. **Horizontal changes**: This is computed by convolving I with a kernel Gx with odd size. For example, for a kernel size of 3, Gx would be computed as:

$$\mathbf{Gx} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

b. **Vertical changes**: This is computed by convolving I with a kernel Gy with odd size. For example, for a kernel size of 3, Gy would be computed as:

$$\mathbf{Gy} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

2. At each point of the image we calculate an approximation of the *gradient* in that point by combining both results above:

$$G=(G2\ x+G2\ y)1/2$$

3. Although sometimes the following simpler equation is used:

$$G=|Gx|+|Gy|$$

After the completion of the pre-processing, the image will be free from the noises, but we still need to enhance the image since the obtained image is smoothened, edges may not be preserved, and the image will be dull. To overcome all these, we used edge detection called Sobel filtering technique. The whole thing is done by calculating the gradient of image intensities at each pixel within the image. It is widely used in image analysis to help locate edges in images. It will also enhance the darker areas of the image, slightly increase contrast and as sharp as possible.

MODULE 2: IMAGE SEGMENTATION USING BINARY THRESHOLD

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.

Segmentation methods has ability to detect or identify the abnormal portion from the image which is useful for analyzing the size, volume, location, texture and shape of the extracted image. MR image segmentation with the aid of preserving the threshold information, which is convenient to identify the broken regions extra

precisely. It was a trendy surmise that the objects that are placed in close propinquity might be sharing similar houses and characteristics.

THRESHOLDING:

Thresholding is the simplest method of image segmentation. It is a non-linear operation that converts a grey-scale image into a binary image where the two levels are assigned to pixels that are below or above the specified threshold value. In Open CV, we use **cv2.threshold()** function:

cv2.threshold(src, thresh, maxval, type[dst])

This function applies fixed-level thresholding to a single-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image for removing a noise, that is, filtering out pixels with too small or too large values. “maxval” is the set threshold value which compares with input values, when the input is greater than the set threshold value it gives output is set maxval value and it is shown with white color in gray images. when the input pixel intensity values are less than the set threshold, its output is black color. There are several types of thresholding supported by the function.

The function returns the computed threshold value and thresholder image.

1. **src** - input array (single-channel, 8-bit or 32-bit floating point). This is the source image, which should be a grayscale image.

2. **thresh** - threshold value, and it is used to classify the pixel values.

3. maxval - maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types. It represents the value to be given if pixel value is more than (sometimes less than) the threshold value.

4. type - thresholding type

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INVY

MORPHOLOGICAL OPERATIONS:

Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors.

The Morphological techniques are also used with segmentation techniques. The morphological action is normally performed on binary images. It processes the operations based on shape and it has a wide set of the image processing operation. Erosion and Dilation are two methods of morphological operations which used in this proposed work. We perform both Erosion and dilation operations used together.

Two main steps of the erosion and dilation morphological operation are opening and closing. The first step is the opening of the MRI binary image. The main work of opening operation is open up a gap which is present in between object and connect that to a small collection of pixels. After setting of the bridge, the erosion again restored with their actual size using dilation. If the binary image has been opened then the subsequent opened same structured elements have not affected on that image. After completing the opening operations next step is the closing operation. Based on the closing operation while keeping the original region sizes, the erosion and dilation can handle different hole in the image region. Dilation and Erosion are the basic morphological operations. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries.

Watershed Method: considers the gradient magnitude of an image as a topographic surface where high gradient denotes peaks, while low gradient denotes valleys. Start by filling every isolated valley with different coloured water. As the water rises, water from different valleys will start to merge. To avoid that, barriers are built in the locations where water merges. Continue the work of filling water and building barriers until all the peaks are under water. Then the created barriers give the segmentation result.

MODULE 3: BRAIN TUMOR IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

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 (ConvNet/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 preprocessing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNet have the ability to learn these filters/characteristics.

A ConvNet 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.

For this step we need to import Keras and other packages that we're going to use in building the CNN. Import the following packages:

 Sequential is used to initialize the neural network.

 Convolution2D is used to make the convolutional network that deals with the images.

 MaxPooling2D layer is used to add the pooling layers.

 Flatten is the function that converts the pooled feature map to a single column that is passed to the fully connected layer.

 Dense adds the fully connected layer to the neural network.

SEQUENTIAL:

 To initialize the neural network, we create an object of the Sequential class.

 classifier = Sequential ()

CONVOLUTION:

 To add the convolution layer, we call the *add* function with the classifier object and pass in *Convolution2D* with parameters. The first argument *feature_detectors* which is the number of feature detectors that we want to create. The second and third parameters are dimensions of the feature detector matrix.

 We used 256 feature detectors for CNNs. The next parameter is *input shape* which is the shape of the input image. The images will be converted into this shape during pre-processing. If the image is black and white it will be converted into a 2D array and if the image is coloured it will be converted into a 3D array.

 In this case, we'll assume that we are working with coloured images. *Input_shape* is passed in a tuple with the number of channels, which is 3 for a coloured image, and the dimensions of the 2D array in each channel. If you are not using a GPU it's advisable to use lower dimensions to reduce the computation time. The final parameter is the activation function. Classifying images is a nonlinear problem. So, we use the rectifier function to ensure that we don't have negative pixel values during computation. That's how we achieve non-linearity.

 classifier.add (Convolution2D (256, 3, 3, input_shape = (256, 256, 3), activation='relu'))

POOLING:

 The Pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

 There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Generally, we use max pooling.

 In this step we reduce the size of the feature map. Generally, we create a pool size of 2x2 for max pooling. This enables us to reduce the size of the feature map while not losing important image information.

 classifier.add (MaxPooling2D (pool_size= (2,2)))

FLATTENING:

 In this step, all the pooled feature maps are taken and put into a single vector for inputting it to the next layer.

 The *Flatten* function flattens all the feature maps into a single long column.

 classifier.add (Flatten ())

FULLY CONNECTION:

 The next step is to use the vector we obtained above as the input for the neural network by using the *Dense* function in Keras. The first parameter is *output* which is the number of nodes in the hidden layer. You can determine the most appropriate number through experimentation. The higher the number of dimensions the more computing resources you will need to fit the model. A common practice is to pick the number of nodes in powers of two.

 classifier.add (Dense (output = 64))

 The next layer we have to add is the output layer. In this case, we'll use the *sigmoid* activation function since we expect a binary outcome. If we expected more than two outcomes, we would use the *SoftMax* function.

 The *output* here is 1 since we just expect the predicted probabilities of the classes.

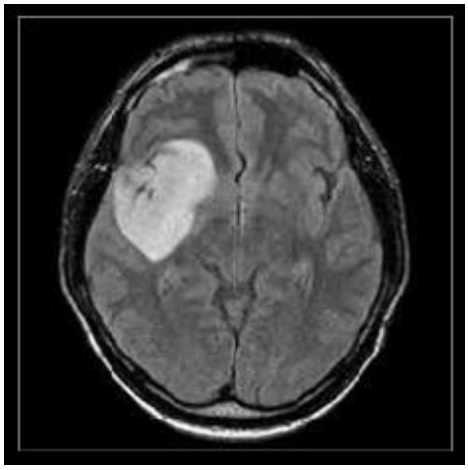
 classifier.add (Dense (output=1, activation='sigmoid'))

Chapter 4
Results and Discussion

SAMPLE CODE AND RESULTS

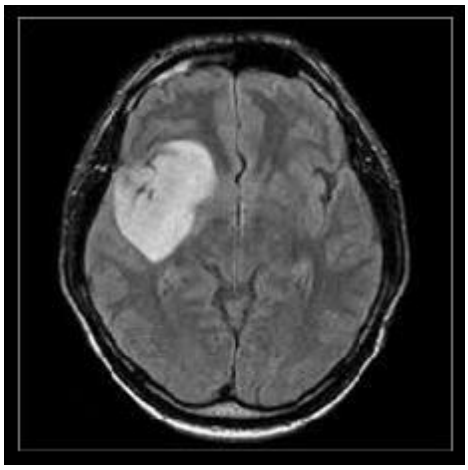
#SEGMENT

```
import cv2
# matplotlib is used for displaying images
import matplotlib.pyplot as plt
%matplotlib inline
# numpy is used for matrix manipulations
import numpy as np

path = r’D: \images\yes\Y32.jpg'
orig_img = cv2.imread(path,1) # 1 indicates color image
# OpenCV uses BGR while Matplotlib uses RGB format
# Display the color image with matplotlib
plt.imshow(cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



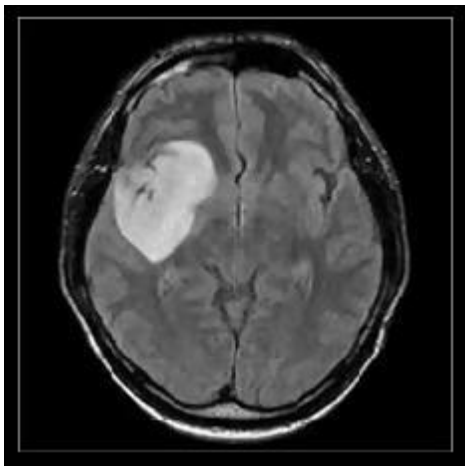
Input Image

```
gray_img = cv2.cvtColor( orig_img, cv2.COLOR_BGR2GRAY )
plt.imshow(gray_img,cmap='gray')
plt.axis('off')
plt.show()
```



Gray-scale Image

```
# To remove salt and pepper noise #Using 5*5 kernel
filtered = cv2.bilateralFilter(gray_img, 5,10,2.5)
plt.imshow(filtered,cmap='gray')
plt.axis('off')
plt.show()
```



Filtered Image

```
# 3*3 Sobel Filters
Gx= np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
print ("Gx \n", Gx)
Gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
print ("Gy \n", Gy)
img_sobelx = cv2.Sobel(filtered,cv2.CV_8U,1,0,ksize=3)

img_sobely = cv2.Sobel(filtered,cv2.CV_8U,0,1,ksize=3)
```

```
#del f = Gx + Gy
# Adding mask to the image
img_sobel = img_sobelx + img_sobely+gray_img
plt.imshow(img_sobel,cmap='gray')
plt.axis('off')
plt.show()
```

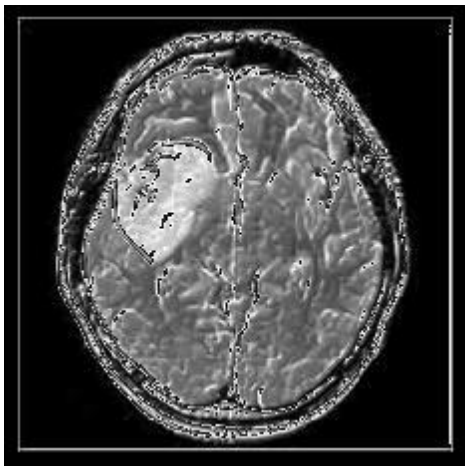
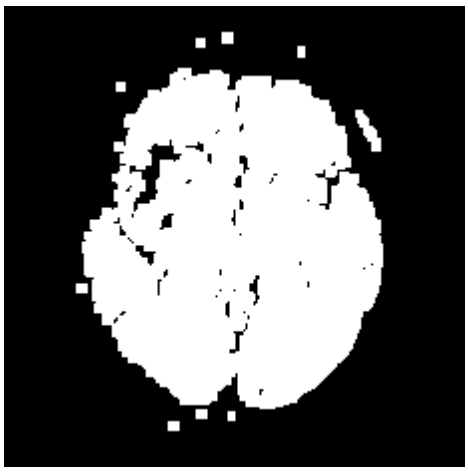


Image after edge detection

```
# Set threshold and maxValue
threshold = 50
maxValue = 255
# Threshold the pixel values
th, thresh = cv2.threshold(img_sobel, threshold, maxValue, cv2.THRESH_BINARY)
plt.imshow(thresh,cmap='gray')
plt.axis('off')
plt.show()
```

Thresholded and Binary Image

```
# To remove any small white noises in the image using morphological opening.
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations = 2)
plt.imshow(opening,cmap='gray')
plt.axis('off')
plt.show()
```



Morphological Image

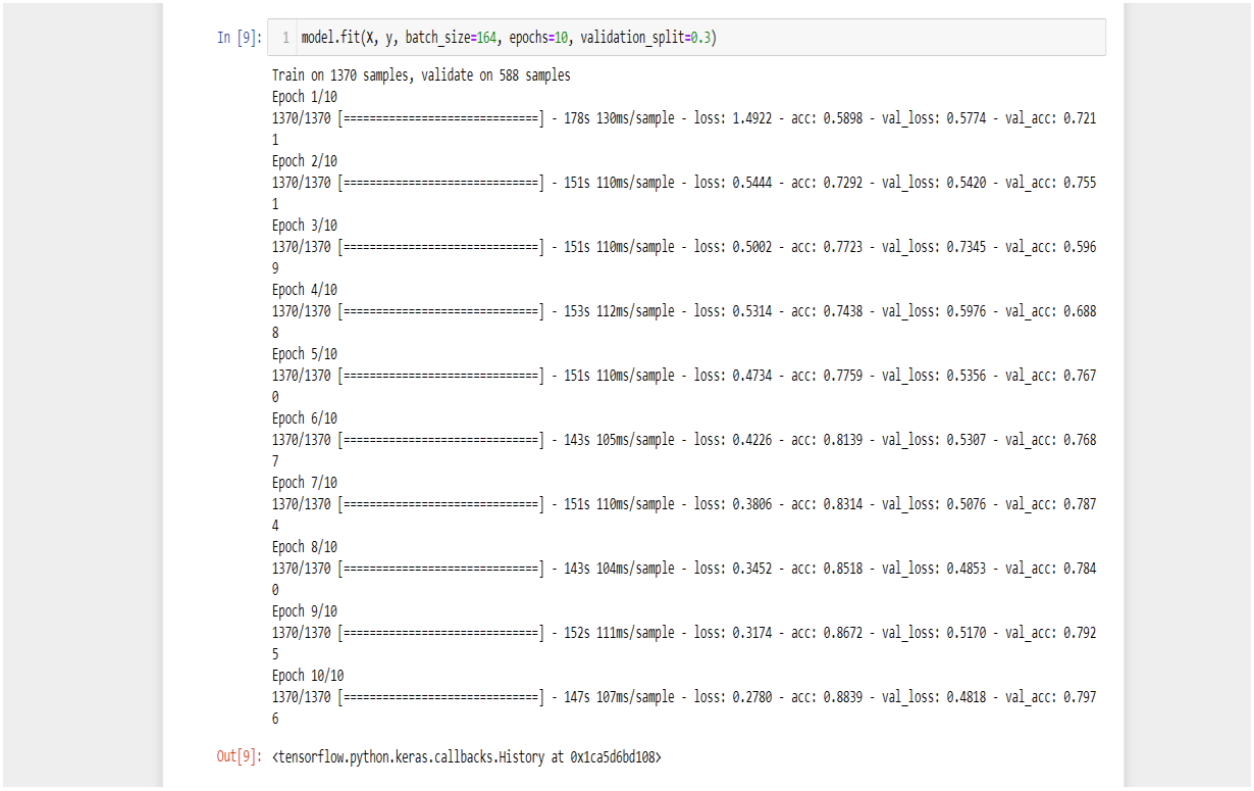
```
# White region shows sure foreground area
sure_bg = cv2.dilate(opening,kernel,iterations=3)
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
plt.imshow(sure_fg,cmap='gray')
plt.axis('off')
plt.show()
```

Dilated Image

#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]: 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 0x1ca5d6bd108>
```

Training data history

#DATALOAD

```
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 it
                plt.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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
```

```
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
```

```
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
```

```
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass
```

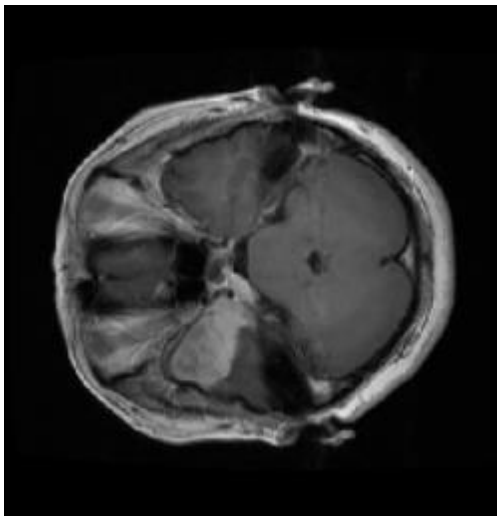
```
IMG_SIZE = 100
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
```

```
training_data
except Exception as e: # in the interest in keeping the output clean...
pass


create_training_data()
print(len(training_data))

import random
random.shuffle(training_data)

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

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
try:
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 to
normalize data size
training_data.append([new_array, class_num]) # add this to our
training_data
except Exception as e: # in the interest in keeping the output clean...
pass


X = []
y = []
for features,label 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)

import pickle
pickle_out = open("XX.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("YY.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

EXPERIMENTAL RESULTS:

Sample Input:

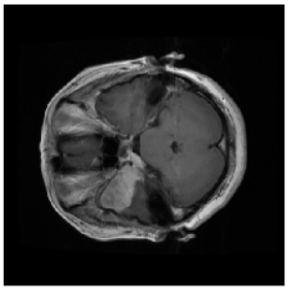


Predicted Output: Yes

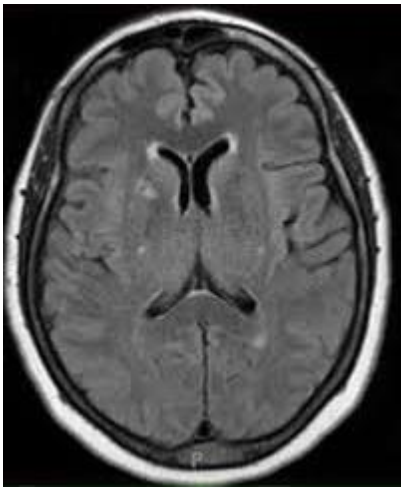
Observed Output:

```
    # The basic purpose of the operation is to show only that part of the image having
    # more intensity which has the tumor that is the part of the image forming our desired extraction.
    kernel=cv2.getStructuringElement(cv2.MORPH_RECT,(7,7))
    erosion = cv2.morphologyEx(median_filtered, cv2.MORPH_ERODE, kernel)
    plt.imshow(erosion,cmap='gray')
    plt.axis('off')
    plt.show()

    #Opening by dilation
    dilation = cv2.morphologyEx(erosion, cv2.MORPH_DILATE, kernel)
    plt.imshow(dilation,cmap='gray')
    plt.axis('off')
    plt.show()

```

```
TUMOR DETECTED
```



Sample Input:



Predicted Output: No

Observed Output:

```
    plt.show()

    copy_img[marker==-1]=(0,0,255)
    cv2.imwrite('img.jpg',copy_img)
    plt.imshow(copy_img,cmap='gray')
    plt.axis('off')
    plt.show()

    #Opening by erosion
    # The basic purpose of the operation is to show only that part of the image having
    # more intensity which has the tumor that is the part of the image forming our desired extraction.
    kernel=cv2.getStructuringElement(cv2.MORPH_RECT,(7,7))
    erosion = cv2.morphologyEx(median_filtered, cv2.MORPH_ERODE, kernel)
    plt.imshow(erosion,cmap='gray')
    plt.axis('off')
    plt.show()

    #Opening by dilation
    dilation = cv2.morphologyEx(erosion, cv2.MORPH_DILATE, kernel)
    plt.imshow(dilation,cmap='gray')
    plt.axis('off')
    plt.show()

else:
    print("NO TUMOR DETECTED")
```

```
NO TUMOR DETECTED
```

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 (FN) and False Negative (FN) the former one designates how m my 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.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

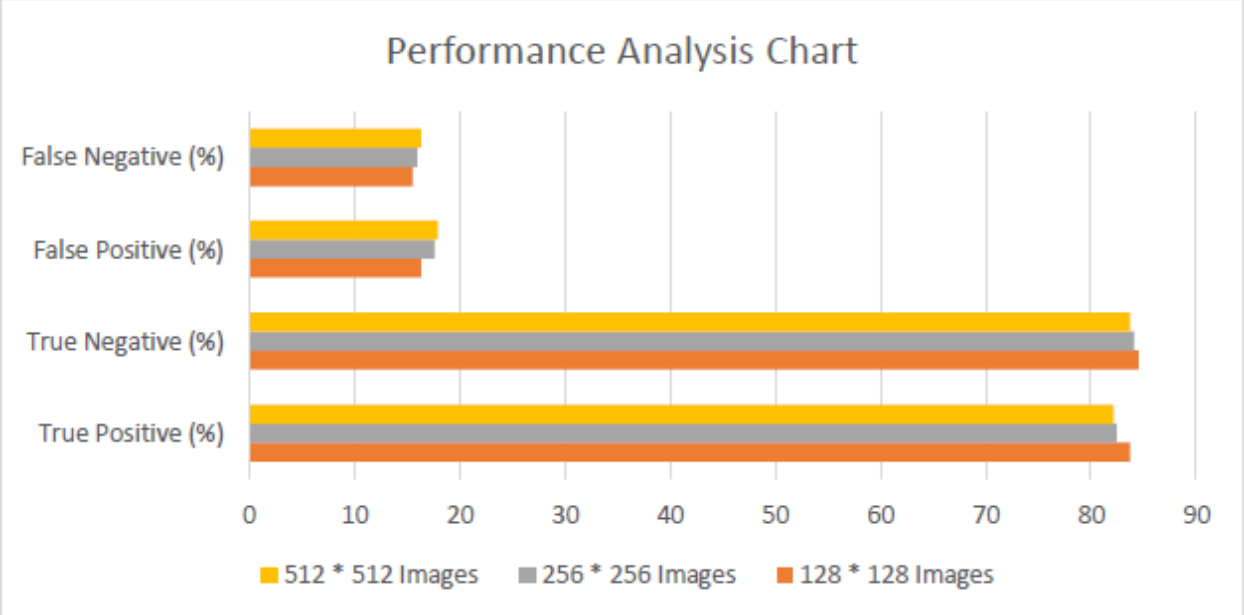
$$Sensitivity = \frac{TN}{TP+FN}$$

PERFORMANCE EVALUTION:

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

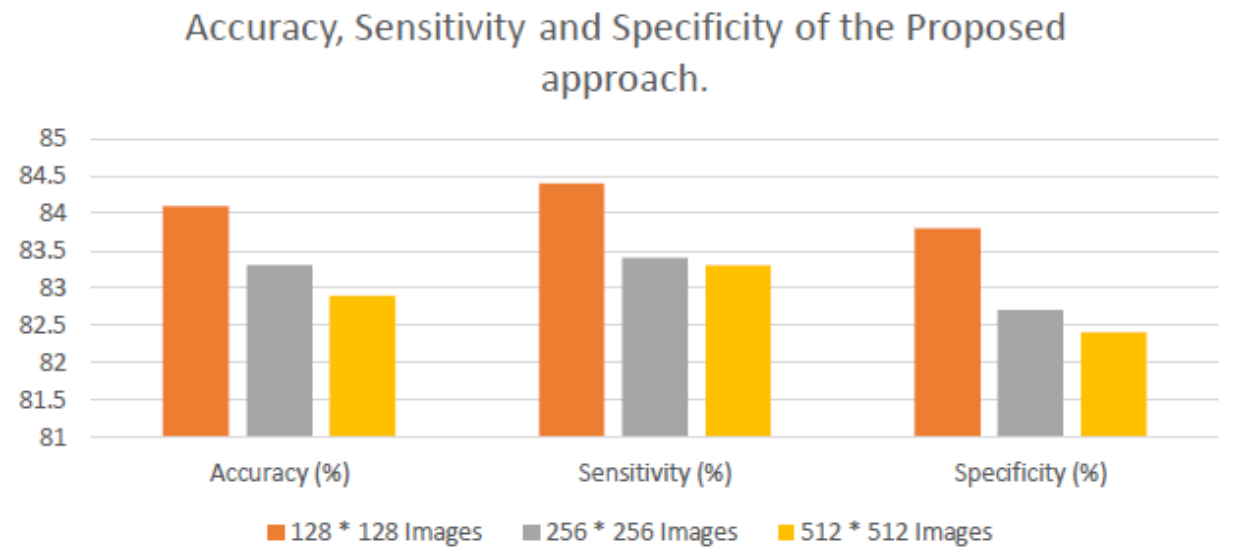


Represents the performance analysis of CNN

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



Represents the performance of proposed CNN

Chapter 5
Conclusion and Future Scope

CONCLUSION:

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.

FUTURE SCOPE:

It is observed on extermination 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.

REFERENCES

[1] A. Sivaramakrishnan And Dr.M.Karnan "A Novel Based Approach For Extraction Of Brain Tumor In MRI Images Using Soft Computing Techniques," International Journal Of Advanced Research In Computer And Communication Engineering, Vol. 2, Issue 4, April 2013.

[2] Asra Aslam, Ekram Khan, M.M. Sufyan Beg, Improved Edge Detection Algorithm for Brain Tumor Segmentation, Procedia Computer Science, Volume 58,2015, Pp 430-437, ISSN 1877-0509.

[3] B.Sathya and R.Manavalan, Image Segmentation by Clustering Methods: Performance Analysis, International Journal of Computer Applications (0975 – 8887) Volume 29– No.11, September 2011.

[4] Devkota, B. & Alsadoon, Abeer & Prasad, P.W.C. & Singh, A.K. & Elchouemi, A.. (2018). Image Segmentation for Early Stage Brain Tumor Detection using Mathematical Morphological Reconstruction. Procedia Computer Science. 125. 115- 123. 10.1016/j.procs.2017.12.017.

[5] K. Sudharani, T. C. Sarma and K. Satya Rasad, "Intelligent Brain Tumor lesion classification and identification from MRI images using k-NN technique," 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil, 2015, pp. 777-780. DOI: 10.1109/ICCICCT.2015.7475384

[6] Kaur, Jaskirat & Agrawal, Sunil & Renu, Vig. (2012). A Comparative Analysis of Thresholding and Edge Detection Segmentation Techniques. International Journal of Computer Applications.vol. 39.pp. 29-34. 10.5120/4898-7432.

[7] Li, Shutao, JT-Y. Kwok, IW-H. Tsang and Yaonan Wang. "Fusing images with different focuses using support vector machines." IEEE Transactions on neural networks 15, no. 6 (2004): 1555-1561.

[8] M. Kumar and K. K. Mehta, "A Texture based Tumor detection and automatic Segmentation using Seeded Region Growing Method," International Journal of Computer Technology and Applications, ISSN: 2229-6093, Vol. 2, Issue 4, PP. 855- 859 August 2011.

[9] Mahmoud, Dalia & Mohamed, Eltaher. (2012). Brain Tumor Detection Using Artificial Neural Networks. Journal of Science and Technology. 13. 31-39.

[10] Marroquin J.L., Vemuri B.C., Botello S., Calderon F. (2002) An Accurate and Efficient Bayesian Method for Automatic Segmentation of Brain MRI. In: Heyden A., Sparr G., Nielsen M., Johansen P. (eds) Computer Vision — ECCV 2002. ECCV 2002. Lecture Notes in Computer Science, vol 2353. Springer, Berlin, Heidelberg.