# Brain Tumor Detection from MR Image of Brain

**Project Report submitted to the**
**West Bengal State University**
**For the partial fulfillment of the requirement**
**For B.Sc. Honors, Computer Science**

**By**

**Suny Dutta**
**(Reg. No. : 1241711400112 OF 2017)**
**(ROLL: 3201105 No. : 03907)**

**Tamal Baroi**
**(Reg. No. :1241511100107 OF 2015)**
**(ROLL: 3201105 No. : 03903)**

**Shreyashi Pal**
**(Reg. No. : 1241721300111 OF 2017)**
**(ROLL: 3202105 No. : 03902)**

# P. R. THAKUR GOVT. COLLEGE

## CERTIFICATE

The project report entitled **“Brain Tumor Detection from MR Image”** is prepared and submitted by **Tamal Baroi (1241511100107 OF 2015), Shreyashi Pal (1241721300111 OF 2017) & Suny Dutta (1241711400112 OF 2017)**. It has been found satisfactory in terms of scope, quality, and presentation as partial fulfillment of the requirements for the award of the degree **Bachelor of Science in Computer Science** at **West Bengal State University, India**.

Academic Session: 2017 - 2020

_____

**(Prof. Probir Modal),**
Project Guide,
HEAD of the Department,
Department of Computer Science

_____

**External Examiner**

# ACKNOWLEDGEMENT

First of all, we would like to thank our project guide, **Prof. Probir Mondal**, for providing us the opportunity to work in this project and spending his time and knowledge during the building and completion of this project. We also want to thank the **Department of Computer Science** of **P. R. Thakur Govt. College** in this regard. We would also like to thank the principal of our college, Officer-in-charge **Swapan Sarkar**, for providing us with a great environment to learn and implement new ideas. Last but not least, we want to thank all who help us in this project directly and indirectly.

_____

***(Tamal Baroi)***
(Reg. No. : 1241511100107 OF 2015)
(ROLL: 3201105 No. : 03903)

_____

***(Shreyashi Pal)***
(Reg. No. 1241721300111 OF 2017)
(ROLL: 3202105 No. : 03902)

_____

***(Suny Dutta)***
(Reg. No. 1241711400112 OF 2017)
(ROLL: 3201105 No. : 03907)

# Index

## Topics ---------- Page Number


(1) Introduction ..... 5
(2) Related Work ..... 5
(3) Project Overview ..... 6
(4) Software Requirement Specification ..... 7
(5) Artifact Removal ..... 8
(6) Denoising ..... 9
(7) Enhancement ..... 11
(8) Segmentation ..... 13
(9) Conclusion ..... 17
References ..... 18
Appendix: Implementation ..... 19

# 1. Introduction

Brain tumors, known as intracranial tumors, are abnormal masses of tissue inside the brain that is a result of uncontrollable and continuous cell growth. Brain tumors are mainly categorized into broad categories - primary brain tumors and metastatic brain tumors. Primary brain tumors originate from the brain or brain-surrounding tissue cells and on the other hand, metastatic tumors are those where cancerous cells grow in other body parts and migrate to the brain through the blood. There is no exact cause for a brain tumor to grow but some DNA mutation which changes DNA's normal structure and the cell becomes cancerous.

Brain tumor patients survive if they are in the early stages of the tumor. Doctors can diagnose patients from their behavior and symptoms related to brain tumors. But they can't look into their brains directly for tumors. Detection of a brain tumor at the early stages is a key issue for providing better and faster treatment and increasing the survival chances of the patient. Here they use techniques such as ultrasonography, magnetic resonance imaging(MRI), computer tomography(CT), positron emission tomography(PET), etc. using which doctors can take images inside of the body for diagnostic purposes. But to detect tumors from those medical images, it takes a significant amount of time. MR imaging gives better results in terms of soft-tissue contrast, depicting anatomy in greater detail, more sensitive, and specific for abnormalities within the brain.

Imaging technology has progressed immensely in recent years and different image processing techniques are available for detailed study of images. Today machine learning and deep learning are used to solve many real-world problems, especially in the medical field. Many automatic image segmentation algorithms have been developed. Still, this area plays important roles in the medical research field, because of no such methods have been found that can work well for all kinds of MRI datasets due to restriction in image acquisition and variations in tumor types.

# 2. Related Work

Khaled Hammouda[5] has surveyed different techniques of data clustering in, "A comparative study of data clustering technique". It implemented different data clustering techniques - k-means clustering, Fuzzy c-means clustering, and Mountain clustering and subtractive clustering. Analyzing these different techniques, it is found that k-means and Fuzzy c-means are preferred when the number of clusters is known.

H.D. Cheng, X.H.Jiang, Y.Sun and Jingli Wang[6] in their, "A Comparative study of data clustering technique" did a literature survey on various color image segmentation techniques. They discussed different approaches to segmentation of monochrome images. They had done comparisons on many techniques and pointed out the advantages and disadvantages of them.

Nicholas Sia Pik Kong, Haidi Ibrahim and Seng Chun Hoo[4] in their, "A literature Review on Histogram Equalization and its Variation for Digital Image Enhancement", had a survey on the histogram equalization techniques and implemented the methods like Histogram Equalization, Local Histogram Equalization, Mean Brightness prevention Histogram Equalization and Modified Histogram Equalization.

Min Ren, Peiyu Liu, Zhihao Wang, and Jing Yi[8] have done surveys on different Clustering Validity Index calculating methods and describe their drawbacks. They have also proposed a

Density-Based algorithm to calculate the clustering validity index in their paper "A Self-Adaptive Fuzzy c-Means Algorithm for Determining the Optimal Number of Clusters".

Sudipta Roy, Kingshuk Chatterjee, Indra Kanta Maitra, and Prof. Samir Kumar Bandopadhyay[2] describes artifacts from MRI scans like patient's details, skull, etc using basic operations performed on images very often in their paper "Artefact Removal from MRI of Brain Image".

Parasuraman Kumar, B. Vijay Kumar[1] in their paper "Brain Tumor MRI Segmentation and Classification Using Ensemble Classifier", describes detecting tumors and classify them using Support Vector Machine, Feed Forwarded Artificial Neural Network and Extreme Learning Machine. Before classification, they preprocess MRI images using the Median filter, segment the image using the Fuzzy C-Means algorithm. After that, they extract different characteristics using the Gray-Level Co-occurrence Matrix, after that Ensemble classification is performed.

# 3. Project Overview

**Introduction of the problem:** The MRI test case brain images with a tumor, collected from the internet, give us cross-sectional 2D brain images. However, the tumor region in the images isn't identified. Thus we need to identify the tumor region in the images using some algorithm executing on the computer.

**Proposed solution:** In our proposed solution, we use a set of algorithms that are frequently used in the image processing field. A noise-removal algorithm is used to remove noises like salt-pepper noise. The image may contain unnecessary data(artifact) like patient details attached along with an MRI image. Thus we apply an artifact-removal algorithm. We enhance images for getting better results. Finally, we apply a clustering algorithm to segment the image in different regions. From the segmented image, we select only the tumor region and identify the boundary of the tumor. The boundary of the tumor is clapped with the original tumor to reflect tumor region in the MRI brain image.

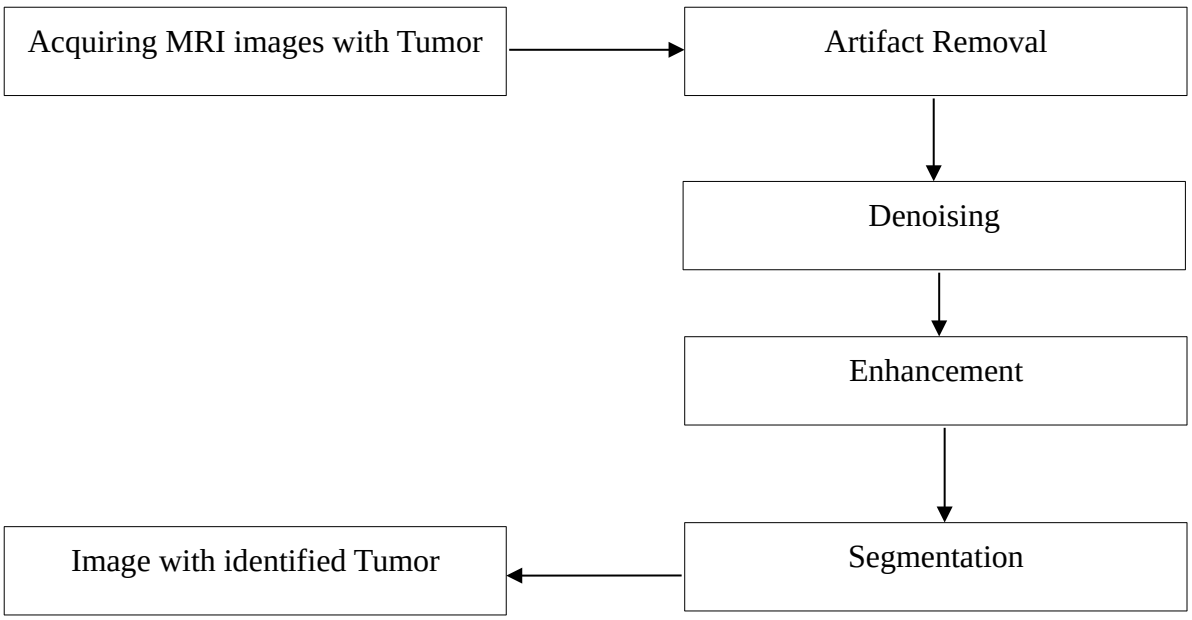


Fig 3: Steps followed to detect Brain Tumor from MRI image

# 4. Software Requirement Specification

Two types of user can use this software -

(a) Doctors
(b) Patients

(a) Doctors use resultant images to find tumor location, size, stage, etc.
(b) Patients use resultant images to see the size of the tumor and its location.

## Functional Requirement:

1) Find a tumor in the provided MRI image.
2) Find the boundary of the tumor and add the boundary around the tumor in the original image.

## Non-Functional Requirement:

1) **Availability -** This software runs on mostly used operating systems like Windows, Ubuntu, Mac OS, etc.
2) **Simplicity -** This software is very easy to use. Anyone can easily understand how to use it in just a few seconds.
3) **Performance -** The performance of this software is time-consuming. Time to find a tumor depends on the size of the MR image.
4) **Security -** No security feature is added to this software.
5) **System Requirements –** The system requirements for this software can be classified into two categories -

(5.1) Hardware Requirement
(5.2) Software Requirement

(5.1) **Hardware Requirement –**

(a) CPU – Any Intel or AMD 64-bit and higher bit processor
(b) RAM – 512 MB (Recommended)
(c) Disk Space – 512 MB (Recommended)

(5.2) **Software Requirement -**

(a) OS – Windows 7, 8, 10 or Ubuntu 18.04 LTS and higher version
(b) Programming Language – C++ programming language is used to build this software.

# 5. Artifact Removal

Artifacts in an MR image can degrade the diagnosis quality that's why artifact removal is a preprocessing step for automated abnormality detection from MRI of the brain. Artifacts are redundant data in MR images which isn't useful for image analysis and must be removed from the image.

We use the algorithm proposed by Sudipta Roy, Kingshuk Chatterjee, Indra Kanta Maitra, and Prof. Samir Kumar Bandopadhyay[2] in their paper "Artefact Removal from MRI of Brain Image". In this method, first, a binarization method has been proposed and a global threshold value has been selected by the standard deviation of the image. Global threshold value has been chosen due to the

large intensity variation of the whole image i.e. the large variation in background and foreground of the whole image. Using global threshold value, binarization is performed i.e. pixel values greater than the threshold are set to maximum and pixel values lesser than or equal to the threshold are set to the minimum value. After binarization, connected component analysis is performed. In this connected component analysis, we assign a unique value to each connected component in the image. Then that connected area is considered which contains the maximum number of pixels. Here the brain area always occupies most of the pixels, the brain region is always identified as the maximum area in the MR image. The maximum area's pixel values are set to 1 and the remaining components are discarded. Then we copy only those pixel values from the original image that are set to 1 in the manipulated image. Using this technique, we can find the brain area from most of the images. Using this technique in low-intensity images may lead to data loss.

The algorithm is as follows -

(1) Calculate the global threshold value using the standard deviation method.
(2) Binarize the image using this global threshold value.
(3) Perform connected component analysis.
(4) Identify the component that holds most of the pixels of the image.
(5) Set intensities of pixels of the max component to 1 and discard the remaining pixel values.

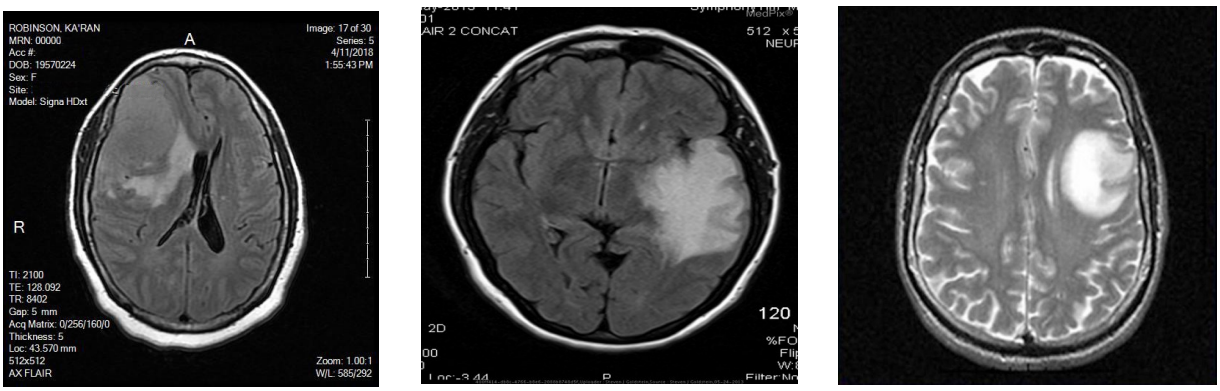


Fig 5.1: Input Image with Artifacts

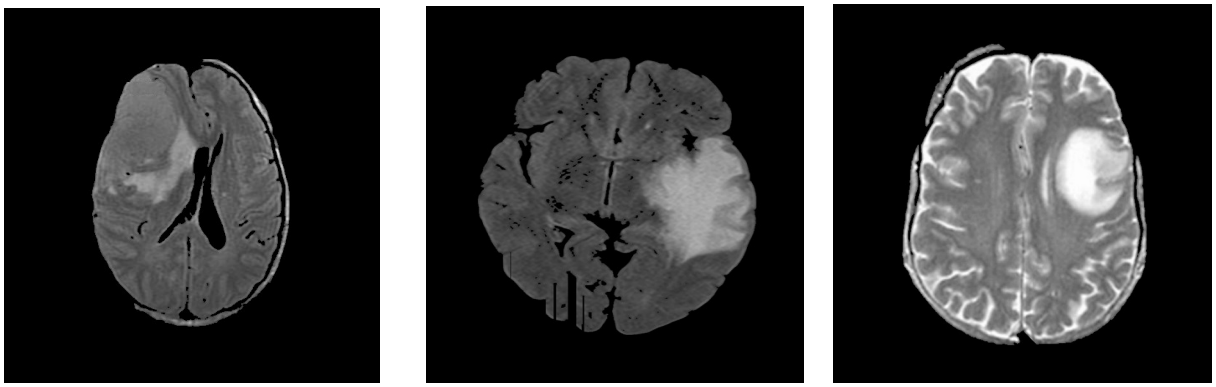

Fig 5.2: Output Image without Artifacts

# 6. Denoising

The term noise in image processing denotes the random variation in brightness or color information in images that probably is not related to the image subject or causes an undesirable image. Noise is often caused by random fluctuation in the signal.

**Gaussian smoothing –** This method is named after the brilliant German mathematician Carl Friedrich Gauß. It is used to smooth images so that noises can't effect image much. The Gaussian kernel is defined in 1-D, 2-D, and N-D form respectively.

1-D form of the Gaussian kernel -

$$G_{1D}(x,\sigma)=\frac{1}{\sqrt{2\pi}\,\sigma}\cdot e^{\frac{-x^2}{2\sigma^2}} \qquad \text{..... (6.1)}$$

2-D form of the Gaussian kernel -

$$G_{2D}(x,y;\sigma)=\frac{1}{2\pi\sigma^2}e^{\frac{-\left(x^2+y^2\right)}{2\sigma^2}} \qquad \text{..... (6.2)}$$

N-D form of the Gaussian kernel -

$$G_{ND}(\dot{x};\sigma)=\frac{1}{\left(\sqrt{2\pi}\,\sigma\right)^N}e^{\frac{-(\dot{x})^2}{2\sigma^2}} \qquad \text{..... (6.3)}$$

Here, σ determine the width of the Gaussian kernel, called the standard deviation. The term $\frac{1}{\sqrt{2\pi}\,\sigma}$ in front of the 1-D Gaussian kernel is the normalization constant. It comes from the fact that the integral over the exponential function isn't unity. With the help of normalization constant, the gaussian kernel becomes a normalized kernel. This also implies that with increasing σ, amplitude reduces substantially. The normalization ensures that the average gray level of the image remains the same when we blur the image with the gaussian kernel.

The Gaussian kernel is a special case of the pascal triangle of binomial coefficients. We can generate some gaussian kernels from the pascal triangle.

To construct a one-dimensional Gaussian kernel with an arbitrary, we simply sample the continuous zero-mean Gaussian function $G(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-x^2}{2\sigma^2}}$ and then normalize by dividing each element by the sum $\sum_i G[i]$ of all the elements. Since G(x) has zero mean, but in implementation, the discrete Gaussian kernel G[i] is centered around $i=\frac{width\ of\ the\ kernel}{2}$ i.e. at half of the kernel's width, we must subtract half the width of the kernel from the index.

Pseudo-code to generate Gaussian kernel for 1-D -

Input: width of kernel w, sigma;

$w_{half}$ <- w/2;
sum <- 0;
for(i <- 0; i < w;i <- i+1) do
G[i] <- (1 / sqrt(2 * π) * sigma) * (exp(-(i - $w_{half}$) * (i - $w_{half}$)) / (2 * sigma * sigma));
sum <- sum + G[i];
done
for(i <- 0;i < w;i <- i+1) do
G[i] <- G[i] / sum;
done
return(G);

In case of 2-D gaussian kernel, two for loops will be used and the statement in 1st inner for loop will be -
G[i][j] <- (1 / 2 * π * sigma) * exp(-(pow(i – $w_{half}$, 2) + pow(j – $w_{half}$, 2)) / (2 * pow(sigma, 2)));
sum <- sum + G[i][j];

And the statement in the inner loop of 2nd for loop will be -

G[i][j] <- G[i][j] / sum;

In our project, we use following 2D kernel having sigma=0.9 and width=7 -

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 9 | 17 | 9 | 1 | 0 |
| 0 | 9 | 57 | 106 | 57 | 9 | 0 |
| 1 | 17 | 106 | 196 | 106 | 17 | 1 |
| 0 | 9 | 57 | 106 | 57 | 9 | 0 |
| 0 | 1 | 9 | 17 | 9 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**Some important properties of Gaussian smoothing method -**

1. Neighborhood values nearer to the central location are more important than more remote values; moreover, the spread parameter σ determines how broad or focused the neighborhood will be. 97% of total weight will be contained within 3σ of the center.
2. Flipping the function for convolution produces the same kernel.
3. A 2D Gaussian kernel is separable. We can decompose a 2D kernel into two 1D gaussian kernels. Every 2D Gaussian kernel is separable.

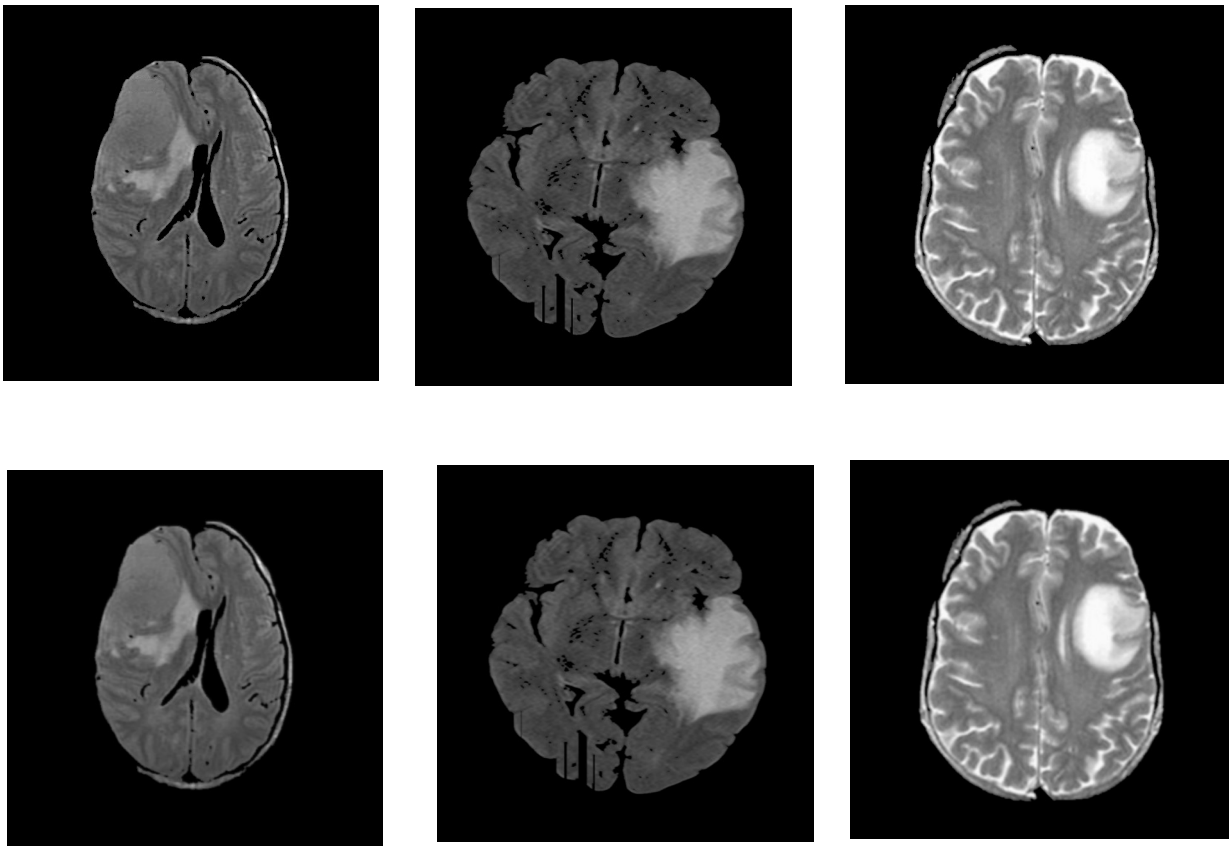

Fig 6: [Upper row] Before smoothing [Lower row] After smoothing

# 7. Enhancement

Enhancement means altering the properties of the object under consideration to provide a better impact on the observer. Image enhancement means altering pixel values such that the image provides better results for operation desired. The processed image is more fitting than the original image for a specific application.

**Histogram Equalization -** A well-known image enhancement technique is histogram equalization. The theoretical basis for histogram equalization depends on the probability theory. An image I represented as $M_r$ x $M_c$ ($M_r$ by $M_c$) matrix of integer pixel intensities ranging from 0-255. A probability of appearance of an intensity $r_k$ is -

$$p_r(r_k) = \frac{n_k}{n}, \quad k = 0, 1, 2, ...., 255 \qquad .... (7.1)$$

where $n_k$ is the number of pixels having intensity $r_k$ and n represents the total number of pixels $(=M_r \times M_c)$ in the image I.

The cumulative distribution process for $p_r$ is as follows -

$$s_k = T(r_k) = \sum_{j=0}^{k} \left( p_r(r_j) \right) = \sum_{j=0}^{k} \left( \frac{n_k}{n} \right) \text{ , k = 0, 1, 2, ...., 255} \qquad \text{..... (7.2)}$$

where $T(r_k)$ represents the transforming of every pixel value $r_k$ to a level $s_k$.

As a result, the resulting image has been obtained through transforming the original pixel intensities $r_k$ to the new pixel intensities $s_k$ via the function $T(r_k)$. The process consists of the following steps:

1. Calculate the $n_k$ of original image, k = 0,1,2,....,255.
2. Compute the probability of appearance($p_r$) of pixel intensity.
3. Calculate the cumulative distribution process($s_k$).
4. Multiply the result of the cumulative distribution process by 255 and round off to the nearest integer.

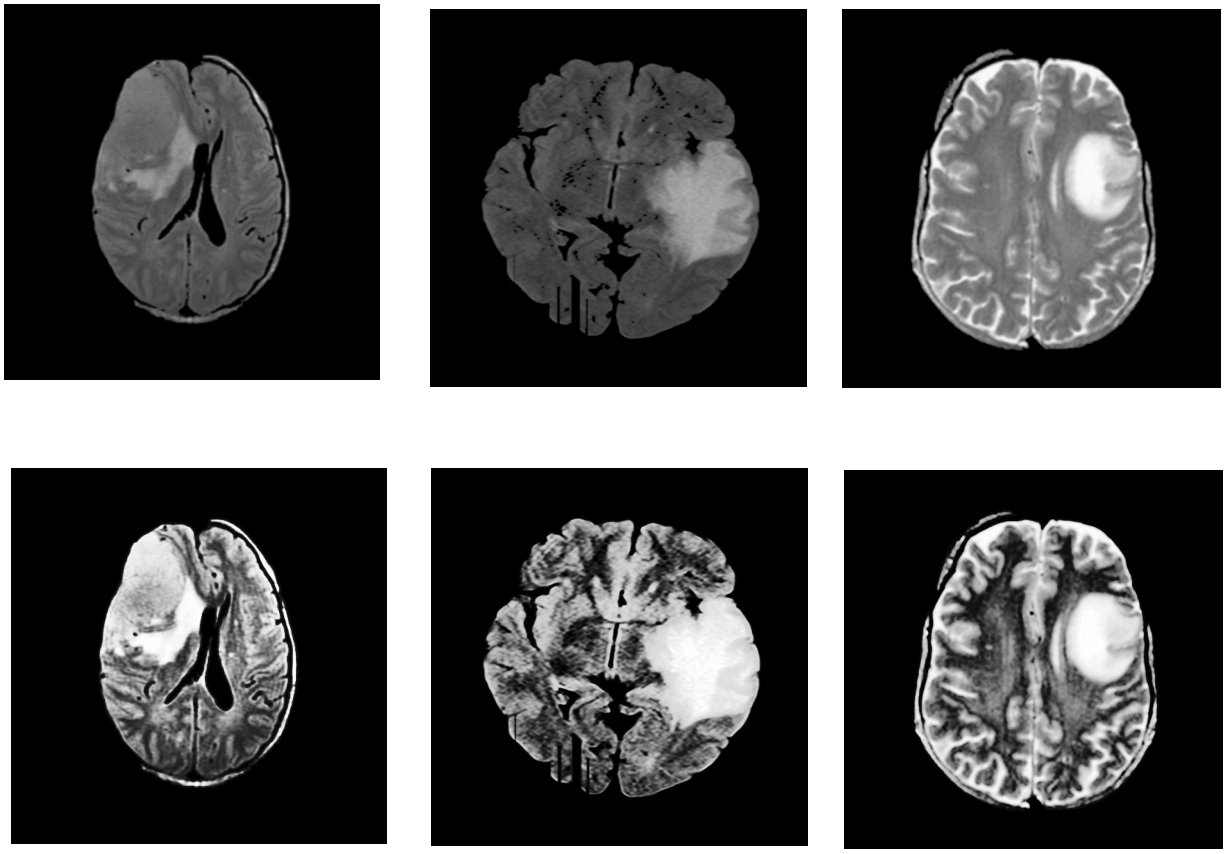

Fig 7 : [ Upper Row ] Input Image [ Lower Row ] Enhanced Output Image

# 8. Segmentation

Image segmentation is a technique to partition image-region into discrete regions based on some attributes such as color, intensity, texture, etc. Image segmentation in image processing is an important, necessary, and challenging part for most real-world problems. It is necessary because it

partitions the image into disjoint sets which ease the job of further processing. It is challenging as well as a complex job because real-world images aren't of the same category. Images are of large categories and different kinds of operations may be required with the same category images, so different kinds of attributes we need to use with perspective to the problem. Still, no general algorithm or method exists which can be used for any kind of image processing problem and gives us a resultant segmented image that is very much suitable for that problem. These algorithms can be broadly categorized into the following categories -

(1) Thresholding based segmentation
(2) Clustering-based segmentation
(3) Edge detection
(4) Graph-based segmentation
(5) Region growing

etc.

In general, clustering refers to a broad spectrum of methods that try to subdivide a data set X into c subsets (clusters) which are pairwise disjoint, all non-empty and reproduce X via union. The clusters are termed a hard c-partition of X. An important fact about this hard c-partition method is that each point in X is grouped with other members of its cluster and doesn't show other apparent similarities to other members of X which belong to different clusters.

**Fuzzy C-Means Algorithm:** One such manner to characterize an individual point's similarity to all clusters is referred to as 'Fuzzy C-Partitioning' which is developed by Dann and later improved by Bezdek. This algorithm tries to put each data point to one of the clusters where the number of clusters must be predefined. What makes FCM different is that it doesn't decide the absolute membership of a data point for a given cluster, instead, it calculates the likelihood (Degree Of Membership) that a point will belong to that cluster. The value of the degree of membership must be between zero and one for each data point for each cluster and depends on the distance between the cluster center and datapoint. Membership close to one implies a high degree of similarity and close to zero implies a degree of dissimilarity between the point and that cluster. Since absolute membership isn't calculated, FCM can be extremely fast but the number of iterations required to achieve a specific accuracy of clustering may cause a delay.

**Iteration -** In each iteration of this algorithm, the following objective function J is minimized:

$$j = \sum_{i=1}^{N} \sum_{j=1}^{C} d_{ij} \|x_i - c_j\|^2 \qquad \text{..... (8.1)}$$

Here, N is the number of data points, C is the number of clusters required, $c_j$ is the center vector for cluster j, and $d_{ij}$ is the degree of membership for the i-th data point $x_i$ in cluster j. The norm, $| | x_i - c_j | |$ measures the similarity (or closeness) of the data point $x_i$ to the center vector $c_j$ of cluster j. Note that, in each iteration, the algorithm maintains a center vector for each of the clusters. These data-points are calculated as the weighted average of the data-points, where the weights are given by the degrees of membership.

**Degree of membership -** For a given data point $x_i$, the degree of its membership to cluster j is calculated as follows:

$$d_{ij} = \frac{1}{\sum_{k=1}^{C} \left( \frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \qquad \text{..... (8.2)}$$

where m is the fuzziness coefficient and the center vector $c_j$ is calculated as follows:

$$c_j = \frac{\sum_{i=1}^{N} d_{ij}^m \cdot x_i}{\sum_{i=1}^{N} d_{ij}^m} \qquad \text{..... (8.3)}$$

In equation (8.3) above, $d_{ij}$ is the value of the degree of membership calculated in the previous iteration. Note that, at the start of the algorithm, the degree of membership for data point i to cluster j is initialized with a random value $t_{ij}, 0 \le t_{ij} \le 1$ such that $\sum_{j}^{C} d_{ij} = 1$ .

**Fuzziness coefficient -** In eq$^n$s (8.2) and (8.3), the fuzziness coefficient m, where $1 < m < \infty$, measures the tolerance of the required clustering. This value determines how much the clusters can overlap with one another. The higher the value of m, the larger the overlap between clusters. In other words, the higher the fuzziness coefficient the algorithm uses, a larger number of data points will fall inside a 'fuzzy' band where the degree of membership is neither 0 nor 1, but somewhere in between.

**Termination condition -** The required accuracy of the degree of membership determines the number of iterations completed by the FCM algorithm. This measure of accuracy is calculated using the degree of membership from one iteration to the next, taking the largest of these values across all data points considering all of the clusters. If we represent the measure of accuracy between iteration k and k + 1 with €, we calculate its value as follows:

$$€ = \Delta_i^N \cdot \Delta_j^C \cdot |d_{ij}^{k+1} - d_{ij}^k| \qquad \text{..... (8.4)}$$

- where, $d_{ij}^k$ and $d_{ij}^{k+1}$ are respectively the degree of membership at iteration k and k+1, and the operator Δ, when supplied a vector of values, returns the largest value in that vector.

The steps in the fuzzy C-Means algorithm are as follows -

---

Assume that, X = $x_1$, $x_2$, $x_3$, ..., xn be the set of data points and V = $v_1$, $v_2$, $v_3$, ..., $v_n$ be the set of centers.

1. Assign the initial value of the number of clusters c, fuzziness index m, maximum iteration $I_{max}$, and threshold €.
2. Initialize the fuzzy partition $d^{(0)}$ randomly according to the constraints of the degree of membership.
3. At the t-step, calculate c cluster centroids $v^{(t)}$ according to eq$^n$ (8.1).
4. Calculate the objective function $J^{(t)}$ according to (8.1). If $|J^{(t)} - J^{(t-1)}| < €$ or $t > I_{max}$, then stop; otherwise continue to step-5.
5. Calculate $d^{(t+1)}$ according to eq$^n$ (8.2) and return to step-3.

---

As mentioned earlier, the number of clusters 'c' must be predefined. If 'c' is unknown, then the determination of an optimal c becomes an important issue. This issue is referred to as the "cluster validity" problem. The searching for fuzzy clustering validity index with excellent performance can

not only make a correct evaluation of the clustering result, and can but also automatically obtain the optimal cluster number. For a good structure of data sets with intra-cluster compactness and inter-cluster separation, calculating of validity index becomes mandatory.

**Cluster Validity Index:** A clustering validity index proposed by Xie and Beni is as follows -

$$V_{xie}(D,V)=\frac{(1/n)\sum_{i=1}^{c}\sum_{j=1}^{n} d_{ij}^{m}\|v_i-x_j\|^2}{min_{i\neq j}\|v_i-v_j\|^2} \qquad \text{..... (8.5)}$$

where $v_{XB}$ index focuses on two properties: compactness and separation. $\frac{1}{n}\sum_{i=1}^{C}\sum_{j=1}^{N} d_{ij}^{m}.\|v_i-x_j\|^2$ is used to measure the degree of intra-cluster compactness. The more bright the cluster divides, the lower its value gets. $\|v_i-v_j\|^2$ is used to measure the degree of separation between clusters. well-separated clusters will produce high value for the separation. Therefore, the most desirable partition is obtained by minimizing $v_{XB}$ for C = 2, 3, ..., $c_{max}$. That is the lower $v_{XB}$ value is, the better the obtained clustering result will be.

However, Bensaid found that the size of each cluster had a large influence on the Xie-Beni index and put forward a new index that was insensitive to the number of objects in each cluster. Bensaid index is defined as follows -

$$V_B(D,V)=\sum_{k=1}^{c}\frac{\sum_{i=1}^{n} d_{ik}^{m}\|x_i-v_k\|^2}{n_k\sum_{j=1}^{c}\|v_j-v_k\|^2} \qquad \text{..... (8.6)}$$

where $n_k$ is the fuzzy cardinality of the kth cluster and defined as -

$$n_k=\sum_{i=1}^{n} d_{ik} \qquad \text{..... (8.7)}$$

$\sum_{i=1}^{n} d_{ik}^{m}\|x_i-v_k\|^2$ shows the variation of the k-th fuzzy cluster. Then, the compactness is computed as -

$$\frac{1}{n_k}\sum_{i=1}^{n} d_{ik}^{m}\|x_i-v_k\|^2 \qquad \text{..... (8.8)}$$

$\sum_{j=1}^{c}\|v_j-v_k\|^2$ denotes the separation of the k-th fuzzy cluster, defined as the sum of the distances from its cluster centroid to the centroids of other c-1 clusters.

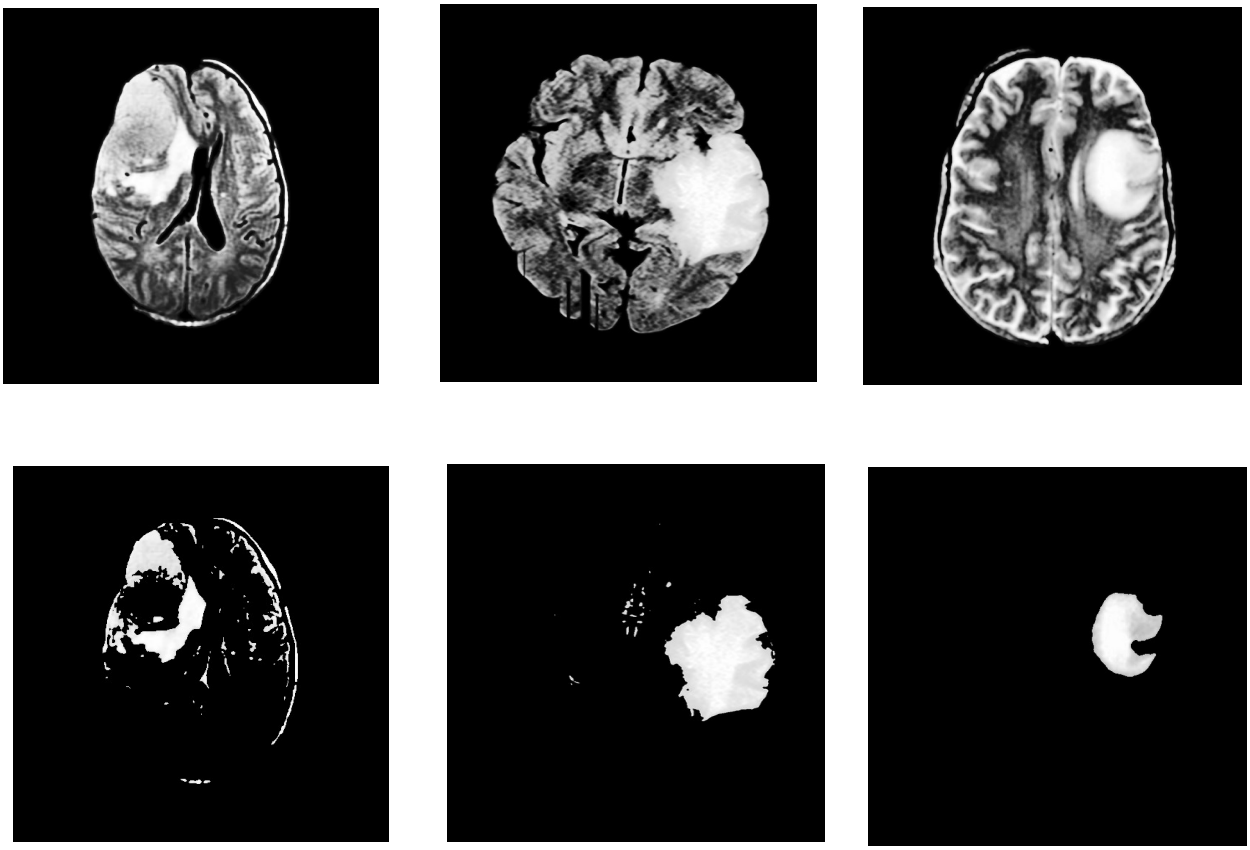

Fig 8.1: [Enhanced input image] [Segmented Ouput image]

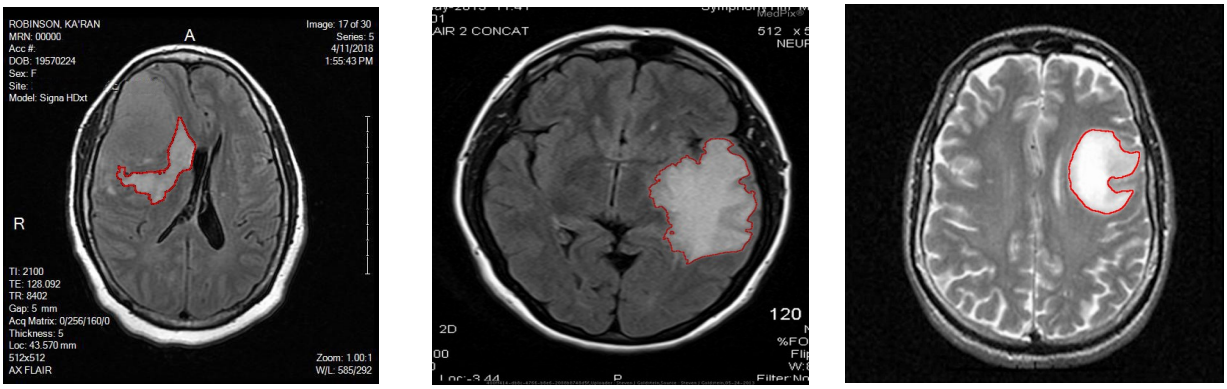


Fig8.2: [Identified tumor region bordered with red in the main image]

The outcomes of segmentation are those areas that are cancerous, then the tumor region is identified and surrounded by a red boundary.

# 9. Conclusion

In our project, we successfully detect most brain tumor region and surround the area with red pixels. In this project, we use all the conventional algorithms to detect tumor region though many modified and enhanced versions of those algorithms exist, especially the Fuzzy C-Means algorithm. We use conventional approaches instead of modified algorithms because we aimed to identify the problems associated with conventional algorithms to identify tumor region.

The future goal of this project is to apply an enhanced version of the algorithms used in this project and effectively implement them in real-life problems which will be much faster, more accurate. In recent research work, using chemical agents, tumor cells of breast cancer are successfully illuminated. In the future, other types of tumor cells may be illuminated and then applying simple algorithms, tumor regions can be more easily and perfectly-identified with less effort and time. Anyone can easily understand the approaches used to detect tumor region. For a more effective understanding of tumor position and size, a 3D model of the brain can be constructed from which size, position, volume, etc. data will be easy to collect.

# References


[1] Parasuraman Kumar, B. Vijay Kumar, “Brain Tumor MRI Segmentation and Classification Using Ensemble Classifier ”, International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8, Issue-1S4, June 2019.

[2] Sudipta Roy, Sanjay Nag, Indra Kanta Maitra, Prof. Samir Kumar Bandyopadhyay, “Artefact Removal and Skull Elimination from MRI of Brain Image”, International Journal of Scientific & Engineering Research, Volume 4, Issue 6, June-2013 ISSN 2229-5518.

[3] Aditya Goyal, Akhilesh Bijalwan, Mr. Kuntal Chowdhury, “A Comprehensive Review of Image Smoothing Techniques” issues in International Journal of Advanced Research in Computer Engineering & Technology Volume 1, Issue 4, June 2012 ISSN: 2278 – 1323.

[4] Nicholas Sia Pik Kong, Haidi Ibrahim, and Seng Chun Hoo, "A Literature Review on Histogram Equalization and Its Variations for Digital Image Enhancement", International Journal of Innovation, Management, and Technology, Vol. 4, No. 4, August 2013.

[5] Khaled Hamuda, " A Comparative study of data clustering technique", Department of System Design Engineering, University of Waterloo, Canada.

[6] Xunali Lisa Xie and Gerardo Beni, “A Validity Measure for Fuzzy Clustering”, IEEE TRANSACTION ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL 13, NO 8, AUGUST 1991.

[7] H, D, Cheng, X.H. Jiang, Y. Sun and J. Wang," Color image segmentation: advances and prospects," Pattern Recognition, 34:2259-2281.

[8] Min Ren, Peiyu Liu, Zhihao Wang, and Jing Yi, “A Self-Adaptive Fuzzy c-Means Algorithm for Determining the Optimal Number of Clusters”, Hindawi Publishing Corporation, Computational Intelligence and Neuroscience, Volume 2016, Article ID 2647389, 12 pages.

[9] Rajeshwar Dass, Priyanka, Swapna Devi, “Image Segmentation Techniques” International Journal of Electronics & Communication Technology (IJECT) 3.1 (2012): 2230-7109.

[10] Manoj Kumar Behera, “An approach for image segmentation using fuzzy c-means clustering.” International Journal of Multidisciplinary Research and Development 2.6 (2015): 349- 4182.

[11] J. C. Bezdek, “Numerical taxonomy with fuzzy sets,” Journal of Mathematical Biology, vol. 1, no. 1, pp. 57–71, 1974.

[12] M. Lee, “Fuzzy cluster validity index based on object proximities defined over fuzzy partition matrices,” in Proceedings of the IEEE International Conference on Fuzzy Systems and IEEE World Congress on Computational Intelligence (FUZZ-IEEE ’08), pp. 336–340, June 2008.

[13] H. Frigui and R. Krishnapuram, “A robust competitive clustering algorithm with applications in computer vision,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 450–465, 1999.

# Appendix: Implementation

**Bitmap.h:**

```
/*************************
/ Bitmap Digital Image Handling Library(v1.11)
/ Author:
/ Time: 7/Sep/2019/11:45 AM
/ Purpose: For Detecting Brain Tumor
/ Platform: Ubuntu 19.04 LTS
/*************************/



#include<iostream>
#include<cstdlib>
#include<fstream>
#include<cstring>
#include<cmath>
#include<iomanip>
#include<vector>

using namespace std;

#pragma pack(push)
#pragma pack(2)
struct BMPFileStructure
{
//BitMapFileHeader--:
uint16_t type{0x4D42};
uint32_t size1{0};
uint16_t reserved1{0};
uint16_t reserved2{0};
uint32_t ofBits{0};

//BitMapInfoHeader--:
uint32_t size2{0};
uint32_t width{0};
uint32_t height{0};
uint16_t planes{1};
uint16_t bitCount{0};
uint32_t compression{0};
uint32_t sizeImage{0};
uint32_t xPxlPerMeter{0};
uint32_t yPxlPerMeter{0};
uint32_t clrUsed{0};
uint32_t clrImportant{0};
};
```

```
#pragma pack(pop)

struct pixel
{
unsigned int B, G, R;
};

class Bitmap
{
BMPFileStructure bmp;
uint8_t *others = NULL;
pixel **pixelMatrix;
int e, rowsize;
vector<int> stack;
protected:
int xGradient(int x, int y);
int yGradient(int x, int y);
void check(int );
void calc(int ,int ,int ,int );
public:
Bitmap(void){ };
Bitmap(int ,int ,int );
Bitmap(string fname){ open(fname); };
//IMage information--:
void open(string );
void save(string ,int del);
void details(void);
int getHeight(void){ return(bmp.height); };
int getWidth(void){ return(bmp.width); };
void operator=(Bitmap );
pixel getPixel(int ,int );
void setPixel(int ,int ,pixel );
void savePixels(string ,int ,int ,int ,int );
//Basic Operation--:
void gray(void);
void complement(void);
void subtruct(Bitmap );
void multiply(Bitmap );
int standardDeviation(void);
void binarizeImage(int );
void connectedComponent(int component = 1);
void boundary(void);
void clap(Bitmap );
//Filter--:
void gauSmooth(void);
void histogram(string );
void histogramEqualisation(void);
void detectESobel(int );
//Sagmentation--:
void segFCM(void);
```

```
};

Bitmap :: Bitmap(int height, int width, int bitCount = 3)
{
if(height <= 0 || width <= 0)
{
cout <<"Height and Width must be positive integer--:\n";
exit(0);
}
//BitmapFileHeader--:
bmp.type = 0x4d42;
bmp.size1 = (height*width*bitCount)+54;
bmp.reserved1 = 0;
bmp.reserved2 = 0;
bmp.ofBits = 54;
//BitmapInfoHeader--:
bmp.size2 = 55;
bmp.width = width;
bmp.height = height;
bmp.planes = 1;
e = bmp.bitCount = bitCount;
bmp.compression = 0;
bmp.sizeImage = height*width*bitCount;

rowsize = bmp.width*bitCount;
pixelMatrix = new pixel *[height];
for(int i = 0;i < height;i++)
{
pixelMatrix[i] = new pixel[width];
for(int j = 0;j < width;j++)
{
pixelMatrix[i*rowsize+j] = 0;
}
}
}
//Opening, closing and pixels--:
void Bitmap :: open(string fname)
{
ifstream fin;
fin.open(fname, ios::binary);
fin.read((char *)&bmp, sizeof(bmp));
if(bmp.type != 0x4d42)
{
cout <<"Not a Bitmap file--:\n";
fin.close();
exit(1);
}
if((bmp.ofBits - 54) != 0)
{
```

```
others = new uint8_t[bmp.ofBits-54];
fin.read((char *)others, (bmp.ofBits-54));
}
e = (int)bmp.bitCount/8;
fin.seekg(bmp.ofBits);
pixelMatrix = new pixel *[bmp.height];
for(int i = 0;i < bmp.height;i++)
{
pixelMatrix[i] = new pixel[bmp.width];
for(int j = 0;j < bmp.width;j++)
{
if(fin.good())
{
pixelMatrix[i][j].B = fin.get();
pixelMatrix[i][j].G = fin.get();
pixelMatrix[i][j].R = fin.get();
}
else
{
cout <<"Can't read FILE["<<fname<<"]--:\n";
exit(1);
}
}
}
fin.close();
}
void Bitmap :: save(string fname, int del)
{
uint8_t tmp;
ofstream fout;
fout.open(fname, ios::binary);
fout.write((char *)&bmp, sizeof(bmp));
if((bmp.ofBits-54) != 0)
{
fout.write((char *)&others, (bmp.ofBits-54));
}
fout.seekp(bmp.ofBits);
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
tmp = (pixelMatrix[i][j].B > 255)?255:((pixelMatrix[i][j].B < 0)?0:pixelMatrix[i][j].B);
fout.put(tmp);
tmp = (pixelMatrix[i][j].G > 255)?255:((pixelMatrix[i][j].G < 0)?0:pixelMatrix[i][j].G);
fout.put(tmp);
tmp = (pixelMatrix[i][j].R > 255)?255:((pixelMatrix[i][j].R < 0)?0:pixelMatrix[i][j].R);
fout.put(tmp);
}
}
fout.close();
```

```
if(del)
{
for(int i = 0;i < bmp.height;i++)
{
delete(pixelMatrix[i]);
}
if(others != NULL)
{
delete(others);
}
}
}
void Bitmap :: details(void)
{
cout <<"BMPFIleHeader--:\n";
cout <<"\tType--:"<<hex<<bmp.type<<endl;
cout <<"\tSize--:"<<dec<<bmp.size1<<endl;
cout <<"\tOffset--:"<<dec<<bmp.ofBits<<endl;
cout <<"BMPInfoHeader--:\n";
cout <<"\tHeader_Size--:"<<bmp.size2<<endl;
cout <<"\tWidth--:"<<bmp.width<<endl;
cout <<"\tHeight--:"<<bmp.height<<endl;
cout <<"\tPlanes--:"<<bmp.planes<<endl;
cout <<"\tBit Count--:"<<bmp.bitCount<<endl;
cout <<"\tCompression--:"<<bmp.compression<<endl;
cout <<"\tSize of Image--:"<<bmp.sizeImage<<endl;
cout <<"\tPixels per meter--:"<<bmp.xPxlPerMeter<<endl;
}
void Bitmap :: operator=(Bitmap img)
{
bmp = img.bmp;
e = img.e;
if((img.bmp.ofBits - 54) != 0)
{
others = new uint8_t[img.bmp.ofBits-54];
others = img.others;
}
pixelMatrix = new pixel *[bmp.height];
for(int i = 0;i < bmp.height;i++)
{
pixelMatrix[i] = new pixel[bmp.width];
for(int j = 0;j < bmp.width;j++)
{
pixelMatrix[i][j] = img.pixelMatrix[i][j];
}
}
}
pixel Bitmap :: getPixel(int h, int w)
{
if((h > bmp.height || h < bmp.height) || (w > bmp.width || w < bmp.width))
```

```
{
pixel tmp;
tmp.R = tmp.G = tmp.B = -1;
return(tmp);
}
return(pixelMatrix[h][w]);
}
void Bitmap :: setPixel(int h, int w, pixel color)
{
if((h > bmp.height || h < bmp.height) || (w > bmp.width || w < bmp.width))
{
cout <<"Check height and width before assigning pixel--:\n";
return;
}
pixelMatrix[h][w] = color;
}
void Bitmap :: savePixels(string fname,int firstRow,int firstCol,int lastRow,int lastCol)
{
fstream file;
file.open(fname, ios::out);
if(!file.good())
{
cout <<"ERROR--:\n";
return;
}
file <<"Height--:"<<bmp.height<<endl;
file <<"Width--:"<<bmp.width<<endl;
file <<"Bit Count--:"<<bmp.bitCount<<endl;
file <<"{";
for(int i = firstRow;i < lastRow;i++)
{
file <<"{";
for(int j = firstCol;j < lastCol;j++)
{
file <<pixelMatrix[i][j].R;     //saving red pixel intensity--:
if(j != bmp.width-1)
{
file <<", ";
}
}
file <<"}";
if(i != bmp.height-1)
{
file <<", ";
}
file <<"\n";
}
file <<"};\n";
file.close();
}
```

```
//Basic Operation--:
void Bitmap :: gray(void)
{
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
int gray = 0.0722*pixelMatrix[i][j].B;
gray += 0.7152*pixelMatrix[i][j].G;
gray += 0.2126*pixelMatrix[i][j].R;
pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = gray;
}
}
}
void Bitmap :: complement(void)
{
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
pixelMatrix[i][j].B = 255 - pixelMatrix[i][j].B;
pixelMatrix[i][j].G = 255 - pixelMatrix[i][j].G;
pixelMatrix[i][j].R = 255 - pixelMatrix[i][j].R;
}
}
}
void Bitmap :: multiply(Bitmap img)
{
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
pixelMatrix[i][j].B *= img.pixelMatrix[i][j].B;
pixelMatrix[i][j].G *= img.pixelMatrix[i][j].G;
pixelMatrix[i][j].R *= img.pixelMatrix[i][j].R;
}
}
}
void Bitmap :: subtruct(Bitmap img)
{
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
pixelMatrix[i][j].B -= img.pixelMatrix[i][j].B;
pixelMatrix[i][j].G -= img.pixelMatrix[i][j].G;
pixelMatrix[i][j].R -= img.pixelMatrix[i][j].R;
}
}
}
```

```
int Bitmap :: standardDeviation(void)
{
float sum = 0, mean, sD = 0;
int tmp, dp = 0;
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
sum += pixelMatrix[i][j].B;
}
}
mean = sum/(bmp.height*bmp.width);
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
tmp = pixelMatrix[i][j].B;
sD += pow((tmp-mean), 2);
}
}
sum = sqrt(sD/(bmp.height*bmp.width));
return(sum);
}
void Bitmap :: binarizeImage(int threshold)
{
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
int tmp = pixelMatrix[i][j].R;
tmp = (tmp > threshold)?255:0;
pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = tmp;
}
}
}
void Bitmap :: connectedComponent(int component)
{
int *area, side[4], label = 0, tmp, t = 0, max = -1;
vector<int> equivlncArr;

//Storing value zero for label 0--:
equivlncArr.push_back(0);
//Labeling 'pixelMatrix' and storing labels in 'equivlncArr' vector--:
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
if(pixelMatrix[i][j].R != 0)    //if pixel value having value greater than 0--:
{
if(i != 0)                  //Upper row--:
```

```
{
side[0] = (j == 0)?0:pixelMatrix[i-1][j-1].R; //Upper row leftmost pixel--:
side[1] = pixelMatrix[i-1][j].R;                    //Upper row middle pixel--:
side[2] = (j == bmp.width)?0:pixelMatrix[i-1][j+1].R;       //Upper row right most pixel--:
}
else
{
side[0] = side[1] = side[2] = 0;          // When (i-1) < 0
}
side[3] = (j == 0)?0:pixelMatrix[i][j-1].R;    //Current row left most pixel--:
int min = 9999;
for(int k = 0;k < 4;k++)
{
//Finding minimum label which is also must be non-zero--:
if(side[k] > 0 && min > side[k])
{
min = side[k];
}
}
if(min != 9999)
{
//Updating with minimum label for label 'tmp'--:
for(int k = 0;k < 4;k++)
{
tmp = side[k];
equivlncArr[tmp] = equivlncArr[min];
}
}
else
{
//Creating new label--:
min = ++label;
equivlncArr.push_back(label);
}
pixelMatrix[i][j].R = pixelMatrix[i][j].G = pixelMatrix[i][j].B = min;       //Labling--:
}
}
}
//Allocating memory to area of size of 'label';
area = new int[label];
for(int i = 0;i < label;i++)
{
area[i] = 0;
}
//Relabeling 'pixelMatrix' and calculating area for each label--:
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
tmp = pixelMatrix[i][j].R;
```

```
if(tmp)
{
pixelMatrix[i][j].R = pixelMatrix[i][j].G = pixelMatrix[i][j].B = equivlncArr[tmp];
//Relabling--:
area[equivlncArr[tmp]]++;    //Calculating area--:
}
}
}
equivlncArr.clear();    //Deleting equivlncArr--:
//Finding maximum area and storing index of max area in 'tmp' variable--:
for(int k = 1;k < label;k++)
{
cout <<k<<"--: "<<area[k]<<endl;
if(max < area[k])
{
max = area[k];//number of pixel's in max area--:
t = k;              //Label of max area--:
}
}
cout <<"Max area--: "<<max<<"\tLabel--: "<<t<<endl;
//Labeling max area (indicated by label 't') to 1 and all others to zero--:
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
tmp = pixelMatrix[i][j].R;
tmp = (tmp == 1)?1:0;          //if pixel label equal to max area's label--:
pixelMatrix[i][j].R = pixelMatrix[i][j].G = pixelMatrix[i][j].B = tmp;

}
}
delete(area);
}
void Bitmap :: boundary(void)
{
int threshold = standardDeviation();
binarizeImage(threshold);
detectESobel(0);
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
if(pixelMatrix[i][j].R)
{
pixelMatrix[i][j].R = 255;
}
}
}
}
void Bitmap :: clap(Bitmap img)
```

```
{
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
if(img.pixelMatrix[i][j].R)
{
pixelMatrix[i][j].R = img.pixelMatrix[i][j].R;
pixelMatrix[i][j].G = pixelMatrix[i][j].B = 0;
}
}
}
}
void Bitmap :: gauSmooth(void)
{
int GConst = 1000;
int filter[7][7] = {{0,0,0,1,0,0,0},
                    {0,1,9,17,9,1,0},
                    {0,9,57,106,57,9,0},
                    {1,17,106,196,106,17,1},
                    {0,9,57,106,57,9,0},
                    {0,1,9,17,9,1,0},
                    {0,0,0,1,0,0,0}};
for(int i = 3;i < bmp.height-3;i++)
{
for(int j = 3;j < bmp.width-3;j++)
{
int sum = 0;
for(int k = -3;k <= 3;k++)
{
for(int l = -3;l <= 3;l++)
{
int gval = pixelMatrix[i+k][j+l].B;
int fval = filter[k+3][l+3];
sum += gval*fval;
}
}
sum /= GConst;
sum = (sum > 255)?255:((sum < 0)?0:sum);
pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = sum;
}
}
}
void Bitmap :: histogram(string fname)
{
//Bitmap img(512, 512);
int frequency[256] = {0}/*, xOrigin = 100, yOrigin = 100*/;
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
```

```
{
int tmp = pixelMatrix[i][j].B;
frequency[tmp]++;
}
}
cout <<"Histogram--:\n";
for(int i = 0;i < 256;i++)
{
cout <<"Histogram["<<i<<"]--:"<<frequency[i]<<endl;
}
}
void Bitmap :: histogramEqualisation(void)
{
float CN[256] = {0}, add = 0;
int frequency[256] = {0}, total = 0, tmp;
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
tmp = pixelMatrix[i][j].B;
frequency[tmp]++;
}
}
for(int i = 1;i < 256;i++)
{
if(frequency[i])
{
total += frequency[i];
}
}
for(int i = 1;i < 256;i++)
{
CN[i] = (float)frequency[i]/total;
add += CN[i];
CN[i] = add;
}
for(int i = 1;i < 256;i++)
{
CN[i] *= 255;
if(CN[i]-ceil(CN[i]) > 0.5)
{
CN[i] = ceil(CN[i])+1;
}
else
{
CN[i] = ceil(CN[i]);
}
}
for(int i = 0;i < bmp.height;i++)
{
```

```
for(int j = 0;j < bmp.width;j++)
{
tmp = pixelMatrix[i][j].B;
if(frequency[tmp])
{
frequency[tmp]--;
pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = (int)CN[tmp];
}
}
}
}
void Bitmap :: detectESobel(int threshold = 0)
{
int gx[bmp.height][bmp.width] = {0}, gy[bmp.height][bmp.width] = {0};
int max = -20, min = 2000;
for(int i = 1;i < bmp.height-1;i++)
{
for(int j = 1;j < bmp.width-1;j++)
{
//Horizontal--:
gx[i][j] = pixelMatrix[i-1][j-1].B
  +2*pixelMatrix[i-1][j].B
  +pixelMatrix[i-1][j+1].B
  -pixelMatrix[i+1][j-1].B
  -2*pixelMatrix[i+1][j].B
  -pixelMatrix[i+1][j+1].B;
gy[i][j] = pixelMatrix[i-1][j-1].B
  +2*pixelMatrix[i][j-1].B
  +pixelMatrix[i+1][j-1].B
  -pixelMatrix[i-1][j+1].B
  -2*pixelMatrix[i][j+1].B
  -pixelMatrix[i+1][j+1].B;
}
}
//Magnitude--:
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
int tmp = sqrt(pow(gx[i][j], 2)+pow(gy[i][j], 2));
pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = tmp;
if(tmp > max)
{
max = tmp;
}
if(min > tmp)
{
min = tmp;
}
}
```

```
}
int diff = max-min;
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
int tmp = 255*((pixelMatrix[i][j].B-min)/(diff*1.0));
//pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = tmp;
pixelMatrix[i][j].B = pixelMatrix[i][j].G = 0;
pixelMatrix[i][j].R = tmp;
}
}
}
void Bitmap :: segFCM(void)
{
int dP = 0, *cordinate, cluster, preCluster = 0;
float **DOM, **preDOM, **distance, *centroid, preCentroid[5] = {0};
float vIndex, pre_vIndex = 0, tmp, converge = 0;
//Counting number of data points--:
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
if(pixelMatrix[i][j].B)
{
dP++;
}
}
}
//Assigning memory for arrays--:
cordinate = new int[dP];
preDOM = new float *[dP];
DOM = new float *[dP];
distance = new float *[dP];
//assigning data points to cordinate for clustering--:
for(int i = 0, k = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
if(pixelMatrix[i][j].B)
{
cordinate[k++] = pixelMatrix[i][j].B;
}
}
}
for(cluster = 2;cluster < 6;cluster++)
{
cout <<"Cluster--:"<<cluster<<endl;
float sum, prev;
int itr = -1;
```

```
//Allocating memory a/c to cluster value to array--:
centroid = new float[cluster];
for(int i = 0;i < dP;i++)
{
preDOM[i] = new float[cluster];
DOM[i] = new float[cluster];
distance[i] = new float[cluster];
}
//[STEP-1]: Randomly initialize degreeOfmembership matrix--:
for(int i = 0;i < dP;i++)
{
sum = 0;
do {
for(int j = 0;j < cluster;j++)
{
DOM[i][j] = 0;
DOM[i][j] = (rand()%19)-1;
sum += pow(DOM[i][j], 2);
}
sum = sqrt(sum);
}while(sum == 0);
tmp = 0;
for(int j = 0;j < cluster;j++)
{
DOM[i][j] /= sum;
DOM[i][j] *= DOM[i][j];
}
}
//LOOP--:
do {
prev = converge;
//[STEP-2] : Find centroid for each clusters--:
for(int k = 0;k < cluster;k++)
{
sum = 0;
for(int i = 0;i < dP;i++)
{
sum += pow(DOM[i][k], 2);
}
centroid[k] = 0;
for(int i = 0;i < dP;i++)
{
centroid[k] += ((cordinate[i]*pow(DOM[i][k], 2))/sum);
}
//[STEP-3}: Find distance between cordinate & centroid--:
for(int i = 0;i < dP;i++)
{
distance[i][k] = pow(cordinate[i]-centroid[k], 2);
}
}
```

```
//Copying current degreeOfMembership to preDOM--:
for(int i = 0;i < dP;i++)
{
for(int j = 0;j < cluster;j++)
{
preDOM[i][j] = DOM[i][j];
}
}
//[STEP-4}: Update the degreeOfMembership matrix--:
converge = 0;
for(int i = 0;i < dP;i++)
{
sum = 0;
for(int k = 0;k < cluster;k++)
{
tmp = pow(distance[i][k], 2);
sum += (1/tmp);
}
for(int j = 0;j < cluster;j++)
{
tmp = pow(distance[i][j], 2);
DOM[i][j] = (1/tmp)/sum;
//Convergence equation--:
converge += pow(DOM[i][j]- preDOM[i][j], 2);
}
}
//[STEP-5]: Checking for convergence condition occurs or not--:
converge = sqrt(converge);
itr++;
//cout <<"\tIteration--:"<<itr<<endl;
}while(converge > 0.01 && itr < 1000 && fabs(prev-converge) != 0);
//Calculating Fuzzy Clustering Validity Index--:
float nk;
for(int k = 0;k < cluster;k++)
{
sum = nk = 0;
for(int i = 0;i < dP;i++)
{
tmp = pow(DOM[i][k], 2);
sum += (pow(cordinate[i]-centroid[k], 2) * tmp);    //Nominator term of equaltion;
nk += DOM[i][k];                                    //fuzzy cardinality;
}
tmp = 0;
for(int j = 0;j < cluster;j++)
{
tmp += pow(centroid[j]-centroid[k], 2);                    //additive term on denominator;
tmp *= nk;
}
//tmp *= nk;                                                //denominator term;
vIndex += sum/tmp;
```

```
}
if(vIndex > pre_vIndex)
{
for(int i = 0;i < cluster;i++)
{
preCentroid[i] = centroid[i];
}
pre_vIndex = vIndex;
preCluster = cluster;
}
delete(centroid);
for(int i = 0;i < dP;i++)
{
delete(DOM[i]);
delete(preDOM[i]);
delete(distance[i]);
}
cout <<"Cluster--:"<<cluster;
cout <<setprecision(9)<<"\tValidity Index--:"<<vIndex<<endl;
}
//[STEP-6]: Defuzzification--:
int max = 0;
cout <<"\n\nCluster--:"<<preCluster<<"\n\nPreCentroid[0]--:"<<preCentroid[0];
for(int i = 1;i < preCluster;i++)
{
cout <<"\nPreCentroid["<<i<<"]--:"<<preCentroid[i]<<endl;
if(preCentroid[max] < preCentroid[i])
{
max = i;
}
}
//cout <<"Max--:"<<max<<endl;
for(int i = 0;i < bmp.height;i++)
{
for(int j = 0;j < bmp.width;j++)
{
if(pixelMatrix[i][j].B)
{
float tmp = (pixelMatrix[i][j].B < preCentroid[max])?0:pixelMatrix[i][j].B;
pixelMatrix[i][j].B = pixelMatrix[i][j].G = pixelMatrix[i][j].R = (int)tmp;
}
}
}
}
```

**test.cc:**

```
#include<iostream>
#include<cstring>
#include"Bitmap.h"
using namespace std;

string Append(string , string data = "_Result_");

int main()
{
string fname;
cout <<"Give Filename--: ";
cin >>fname;

Bitmap mainImg, copyImg;
int threshold;
mainImg.open(fname);
mainImg.gray();
copyImg = mainImg;
threshold = mainImg.standardDeviation();
mainImg.binarizeImage(threshold);
mainImg.connectedComponent();
mainImg.multiply(copyImg);
mainImg.gauSmooth();
mainImg.histogramEqualisation();
mainImg.segFCM();
mainImg.connectedComponent();
mainImg.multiply(copyImg);
mainImg.boundary();
copyImg.clap(mainImg);

fname = Append(fname);

copyImg.save(fname, 1);

return 0;
}


string Append(string fname, string data)
{
data.append(fname);
data.append(".bmp");
return(data);
}
```