

AUTOMATED DETECTION AND ANALYSIS OF BRAIN TUMOR USING IMAGE PROCESSING FROM MRI IMAGES

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

By

**SANDEEP SRINIVAS (REG
NO: 39110889) RONGALI SASI
KUMAR (REG NO: 39110860)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC | 12B Status

By UGC | Approved by AICTE

**JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI - 600119**

APRIL– 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

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

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BONA FIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Sandeep Srinivas (39110889)** and **Rongali Sasi Kumar (39110860)** who carried out the Project Phase-1 entitled **“AUTOMATED DETECTION AND ANALYSIS OF BRAIN TUMOR USING IMAGE PROCESSING FROM MRI IMAGES”** under my supervision from June 2022 to November 2023.

Internal Guide
Ms. V. Dharani, M.E

Head of the Department
Dr. L. LAKSHMANAN, M.E., Ph.D.



Submitted for Viva voce Examination held on 20.04.2023

Internal Examiner

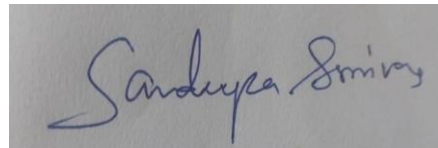
External Examiner

DECLARATION

I, **Sandeep Srinivas (Reg No- 39110889)**, hereby declare that the Project Phase-1 Report entitled “**AUTOMATED DETECTION AND ANALYSIS OF BRAIN TUMOR USING IMAGE PROCESSING FROM MRI IMAGES**” done by me under the guidance of **Ms. V. Dharani** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE:20/04/2023

PLACE: Chennai

A rectangular box containing a handwritten signature in blue ink. The signature is cursive and reads "Sandeep Srinivas".

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

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

I convey my thanks to **Dr. T.Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. V.Dharani M.E.** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-1 project work.

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

ABSTRACT

Brain Tumor is one of the leading causes of cancer death worldwide. Early detection of brain cancer is challenging because cancer-specific symptoms occur only at an advanced stage, and a reliable screening tool to identify high-risk patients is lacking. However, brain cancer can be cured if it is detected at an early stage. In this project, an attempt is made to detect brain tumor from CT images. It uses image processing techniques and a K-means Model Architecture to detect the tumor. After the image pre-processed, K-means Model Architecture is used to detect the tumorous area in the image. Accurate detection of size and location of brain tumor plays a vital role in the diagnosis of tumor. Image processing is an active research area in which medical image processing is a highly challenging field. Image segmentation plays a significant role in image processing as it helps in the extraction of suspicious regions from the medical images. In this paper an efficient algorithm is proposed for tumor detection based on segmentation of brain MRI images using K-Means clustering.

Chapter No	TITLE	Page No.
	ABSTRACT	5
	LIST OF FIGURES	7
1	INTRODUCTION	8
	1.1 Introduction to python	8
	1.2 Introduction to Machine Learning	9
	1.2.1 Supervised learning	10
	1.2.2 Unsupervised Learning	11
	1.2.3 Reinforcement Learning	11
2	LITERATURE SURVEY	12
3	SYSTEM ANALYSIS	16
	3.1 Existing System	16
	3.1.1 Disadvantages of Existing System	16
	3.2 Proposed System	16
4	SSYTEM ARCHITECTURE	18
	4.1 Proposed Architecture	18
	4.2 Input design	18
	4.3 Output design	18
	4.4 System requirements	19
5	IMPLEMENTATION	52
	5.1 Modules	52
	5..1.1 Modules Description	52
	5.2 K- Means Clustering	54
	5.2.1 Computer Vision	54

	5.3	Strided Convolutions	57
	5.4	One layer Of a Convolution Network	59
6		RESULT AND CONCLUSION	61
	6.1	Begnin tumor	61
	6.2	Malignant Tumor	61
	6.3	Conclusion	62
7		References	63
		APPENDIX	
	A.	SOURCE CODE	
	B.	SCREENSHOTS	
	C.	RESEARCH PAPER	

LIST OF FIGURES

S. No	Figure Name	Page No
1.2	Types of machine learning algorithm	11
4.4.1	Image Processing	22
4.4.2	Simple Grayscale level image	41
6.1	Benign Tumor output	61
6.2	Malignant Tumor output	61

CHAPTER – 1

INTRODUCTION

Brain Cancer is the 10th most commonly diagnosed cancer in men and the 9th in women, but the 4th leading cancer death for both men and women in the United States. Brain Cancer is only the major cancer with a five-year relative survival rate in the single digits. A recent report issued by brain cancer action in 2020, brain cancer is expected to become the second largest leading cancer cause of cancer death in the United States.

A major cause of this, is the late detection of the brain tumor because there are no effective early detection methods available. Also, most of the symptoms of brain tumor are vague and could be contributed to many other abdominal conditions. Also, if the cancer has spread to the other organs the treatment becomes very difficult. So, there is an urgent need of method that will help radiologists in diagnosis of the brain tumor at an early stage. There is not much work done on brain tumor detection. It is found from the literature survey that brain tumor detection is done by using the symptoms of the disease and taking patient history but not using image processing.

In this project an attempt is made to detect the brain tumor detection from CT scan images. These images are pre-processed using image processing and deep learning techniques and then a basic classifier is used to classify the tumor area in the image and detect whether it is in the normal condition or it is having any tumor in the pancreas.

1.1 INTRODUCTION TO PYTHON

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other language

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, and Unix shell and other scripting language

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL) and is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Apart from the above-mentioned terms Python has a big list of good features, few are listed below –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python is available on a wide variety of platforms including Linux and Mac OS X.

1.2 MACHINE LEARNING

Machine learning is the scientific field dealing with the ways in which machines learn from experience. For many scientists, the term “machine learning” is identical to the “artificial intelligence” given that the possibility of learning is the main characteristic of an entity called intelligent in the broadest sense of the word. The purpose of machine learning is the construction of computer systems that can adapt and learn from their experience. A more detailed and formal definition of machine learning is given by Mitchell: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E . With the rise of Machine Learning approaches, we have the ability to find a solution to this issue, we have developed a system using data mining which has the ability to predict whether the patient has diabetes or not. Furthermore, predicting the disease early leads to treating the patients before it becomes critical. Data mining has the ability to extract hidden knowledge from a huge amount of diabetes-related data.

Because of that, it has a significant role in diabetes research, now more than ever. The aim of this research is to develop a system which can predict the diabetic risk level of a patient with a higher accuracy. This research has focused on developing a system based on Random Forest Classifier Machine learning techniques.

1.2.1 SUPERVISED LEARNING

In supervised learning, the system must learn an expression of a model describing the data. The objective function is used to predict the value of a variable, called dependent variable or output variable, from a set of variables, called independent variables or input variables or characteristics or features. The set of possible input values of the function, i.e., its domain, are called instances. Each case is described by a set of characteristics (Attributes or features). A subset of all cases, for which the output variable value is known, is called training data or examples. In order to infer the best target function, the learning system, given a training set, takes into consideration alternative functions, called hypothesis and denoted by h . In supervised learning, there are two kinds of learning tasks: classification and regression. Classification

models try to predict distinct classes, such as e.g., blood groups, while regression models predict numerical values. Some of the most common techniques are Decision Trees (DT), Rule Learning, and Instance Based Learning (IBL), such as k-Nearest Neighbours (k-NN), Genetic Algorithms (GA), Artificial Neural Networks (ANN), and Support Vector Machines (SVM).

1.2.2 UNSUPERVISED LEARNING

In unsupervised learning, the system tries to discover the hidden structure of data or associations between variables. In that case, training data consists of instances without any corresponding labels. Association Rule Mining appeared much later than machine learning and is subject to greater influence from the research area of databases. Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

1.2.3 REINFORCEMENT LEARNING

The term Reinforcement Learning is a general term given to a family of techniques, in which the system attempts to learn through direct interaction with the environment so as to maximize some notion of cumulative reward. It is important to mention that the system has no prior knowledge about the behaviour of the environment and the only way to find out is through trial and failure (trial and error). Reinforcement learning is mainly applied to autonomous systems, due to its independence in relation to its environment.

Fig.No.1.2 Types of Machine Learning Algorithms

CHAPTER-2

2.1 LITERATURE SURVEY

**1. Bernstein M, Berger MS. Neuro-oncology, the essentials. Thieme. (2000)
ISBN:0865778809**

Neuro-Oncology: The Essentials, Third Edition, provides a thorough introduction to the fundamental scientific and clinical principles that underlie the successful multidisciplinary treatment of patients with brain and spine tumors. This new edition has eight new chapter topics, some of which include: Endoscopic Methodologies, Pediatric Supratentorial Cancers, and Neuro-Oncology in the Creating Scene. The book starts with basic science, moves on to clinical applications, and includes a substantial section on particular kinds of tumor.

Key Features:

A major revision with a 20 percent increase in content and a 35 percent increase in figures
Editors' Notes at the end of each chapter briefly summarize a topic, offer a historical perspective on a topic, add a current reference, or highlight important points. Demonstrates the

most recent surgical navigation techniques, such as stereotaxy and intraoperative MRI, that are used to access hard-to-reach tumors.

2. BW Stewart, P Kleihues "World Cancer Report" World Health Organization, International Agency for Research on Cancer.

The World Cancer Report provides a unique global view of cancer. It documents the frequency of cancer in different countries, trends in cancer incidence and mortality and it describes the known causes of human cancer. The molecular and cellular basis of the multistep process of malignant transformation is concisely summarized. The report contains an up to date over-view of cancer prevention, including screening programmes for early diagnosis, as well as advances in surgical and medical oncology, including novel drugs targeting tumour-specific signalling pathways. The efforts of the World Health Organization in the fight against cancer are detailed, together with strategies for cancer control.

3. Kaye AH. Essential neurosurgery. Wiley-Blackwell. (2005) ISBN:1405116412.

For junior surgical trainees and medical students, Essential Neurosurgery is a comprehensive introduction to neurosurgery. An understanding of neurology and the pathological basis of neurological disease are central to the book's focus on the fundamentals of neurosurgical diagnosis and treatment of common problems affecting the central nervous system. There is likewise inclusion of neurosurgical strategies and postoperative patient administration.

Many of the biological and technological advancements in neurosurgery that have improved surgical options and patient outcomes are included in this new edition, which brings the text up to date.

4. Yuehao Pan, Weimin Huang et.al "Brain Tumor Grading based on Neural networks and Convolutional Neural Networks",IEEE.

This paper concentrates on brain cancer reviewing utilizing multiphase X-ray pictures and contrasts the outcomes and different designs of profound learning construction and pattern

Brain Organizations. The learning machine uses the MRI images directly, with some combination operations performed on multiphase MRIs. The method that is utilized in this paper makes use of the learning capabilities of a deep learning machine, in contrast to other studies, which require additional effort to design and select feature sets. We present the sensitivity and specificity-based performance of the grading on the testing data. When compared to neural networks, the results demonstrate a maximum improvement of 18% in Convolutional Neural Networks' sensitivity and specificity-based grading performance. We also show some self-learned Convolutional Neural Network features and the kernels that were trained in various layers.

5. Darko Zikic, Yani Ioannou et.al “Segmentation of Brain tumor tissues with convolutional neural networks”, MICCAI, 2014, Brats challenge.

In this work, we investigate the possibility to directly apply convolutional neural networks (CNN) to segmentation of brain tumor tissues. As input to the network, we use multi-channel intensity information from a small patch around each point to be labelled. Only standard intensity pre-processing is applied to the input data to account for scanner differences. No post-processing is applied to the output of the CNN. We report promising preliminary results on the high-grade training data from the BraTS 2013 challenge. Work for the final submission will include architecture modifications, parameter tuning and training on the BraTS 2014 training corpus.

6. Karen, Zisserman “Very deep convolutional networks for large scale image recognition”, ICLR. [Published in 2014].

In this work, we investigate how the accuracy of a convolutional network in large-scale image recognition is affected by its depth. Our main contribution is an in-depth analysis of networks with increasing depth using an architecture with very small (3x3) convolution filters. This demonstrates that increasing the depth to 16-19 weight layers can significantly improve on existing configurations. Based on these findings, our ImageNet Challenge 2014 submission won first and second place, respectively, in the localization and classification tracks. In addition, we demonstrate that our representations yield cutting-edge results in a variety of other datasets when applied broadly. In order to make it easier for future research into the application of deep visual representations in computer vision, we have made the two ConvNet models that perform the best to the general public.

7. Taranjit Kaur, Tapan Kuamar Gandhi, “Automated brain image classification based on VGG-16 and transfer learning”, ICIT [Published in 2019].

From traditional to deep learning methods like convolutional neural networks (CNN), active research in the field of pathological brain image classification has taken place over the past few decades. Classification is accomplished by hand-crafted features using traditional machine learning techniques. CNN, on the other hand, uses image features directly extracted from raw images to perform classification. The size of the training data set has a significant impact on the features that CNN extracts. CNN is more likely to overfit if the size is small. Along these lines, profound CNN's (DCNN) with move learning has advanced. The primary objective of this paper is to investigate the use of transfer learning and a pre-trained DCNN VGG-16 model for the classification of pathological brain images. Just, the last couple of layers of the VGG-16 model were supplanted to oblige new picture classifications in the current application. The dataset from the Harvard Medical School repository, which includes both normal and abnormal MR images of various neurological diseases, was used to validate the pre-trained model with transfer learning. A 10-fold cross-validation method was then used to divide the data set. The pre-trained VGG-16 model with transfer learning performed the best when compared to the other current state-of-the-art works in the validation on the test set using sensitivity (Se), specificity (Sp), and accuracy (Acc). In addition, raw images can be categorised using the method's end-to-end structure without the need for custom attribute extraction.

8. Automated Brain Image Classification Based on VGG-16 and Transfer Learning

From traditional to deep learning methods like convolutional neural networks (CNN), active research in the field of pathological brain image classification has taken place over the past few decades. Classification is accomplished by hand-crafted features using traditional machine learning techniques. CNN, on the other hand, uses image features directly extracted from raw images to perform classification. The size of the training data set has a significant impact on the features that CNN extracts. CNN is more likely to over fit if the size is small. As a result, transfer learning-based deep CNNs (DCNNs) have evolved. The primary objective of this paper is to investigate the transfer learning capabilities of a pre trained DCNN VGG-16 model for the classification of pathological brain images. Just, the last not many layers of the VGG-

16 model were supplanted to oblige new picture classifications in the current application. The pre-prepared model with move learning has been approved on the dataset taken from the Harvard Clinical School storehouse, containing of typical as well as unusual MR pictures with various neurological illnesses. A 10-fold cross-validation method was then used to divide the data set.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- Numerous methods have been proposed to cope with noisy label classification for natural images. Ren et al. propose a technique to assign weights to training samples by using an additional clean validation set. Their intuition is to apply smaller weights to noisy samples and increase the weights of clean training samples to improve the gradient update. There is relatively limited work on the development and application of noisy label classification methods in medical imaging data.
- Dgani et al. model label noise as a part of the deep learning network to recover true labels of noisy samples for the task of classifying breast micro-calcifications in multi-view mammograms.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

- There is not much work done on brain tumor detection.
- It is found from the literature survey that brain tumor detection is done using the symptoms of the disease and by taking patient history but not using image processing.
- So, here we can't predict the disease in the early stage , it will become difficult when it goes to further states,

- Detecting the disease is very slow.
- It did not use any image processing techniques to predict disease at the early stage.

3.2 PROPOSED SYSTEM

In this project, an attempt is made to detect brain tumor from CT scan images. These images are preprocessed using image processing techniques and then K means model architecture is used to classify the tumor area in the image. The dataset is taken from the cancer imaging archive repository. These CT scan images of brain tumor are given as inputs to the system. These images are different format, so we have converted into jpg and also the image size is too huge, so we have used few of them only. Then classification is done using K-means model architecture. In this, the system is made learn according to the classes defined in the image. Once the system learns the classification done based on the features given to it, it can then classify the test data into one of those classes.

We trained a K-means using contrast enhanced-CT images of patients to distinguish brain cancer from healthy pancreases. K-means achieved excellent accuracy and improved sensitivity compared with radiologist interpretation in independent test sets, with acceptable performance in a test set obtained from patients of various types. These results provide the first solid proof of concept that K-means can capture the elusive CT features of brain cancer to assist and supplement radiologists in the detection and diagnosis of brain cancer.

CHAPTER 4

SYSTEM ARCHITECTURE

4.1 PROPOSED ARCHITECTURE:

- It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
- It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
- This shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

4.2 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the

process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy.

4.3 OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

4.4 SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

- System : Intel core i5 Processor.
- Hard Disk : 1000 GB.
- Monitor : 15'' LED
- Input Devices : Keyboard, Mouse
- Ram : 8 GB

SOFTWARE REQUIREMENTS:

- Operating system : Windows 10.
- Coding Language : Python

➤ Web Framework : Flask.

SYSTEM DESIGN:

IMAGE PROCESSING:

What is Image Processing?

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them. It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.

· Output is the last stage in which result can be altered image or report that is based on image analysis.

Purpose of Image processing

The purpose of image processing is divided into 5 groups. They are:

1. Visualization - Observe the objects that are not visible.
2. Image sharpening and restoration - To create a better image.
3. Image retrieval - Seek for the image of interest.
4. Measurement of pattern – Measures various objects in an image.
5. Image Recognition – Distinguish the objects in an image.

Types

The two types of methods used for Image Processing are Analog and Digital Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing. Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

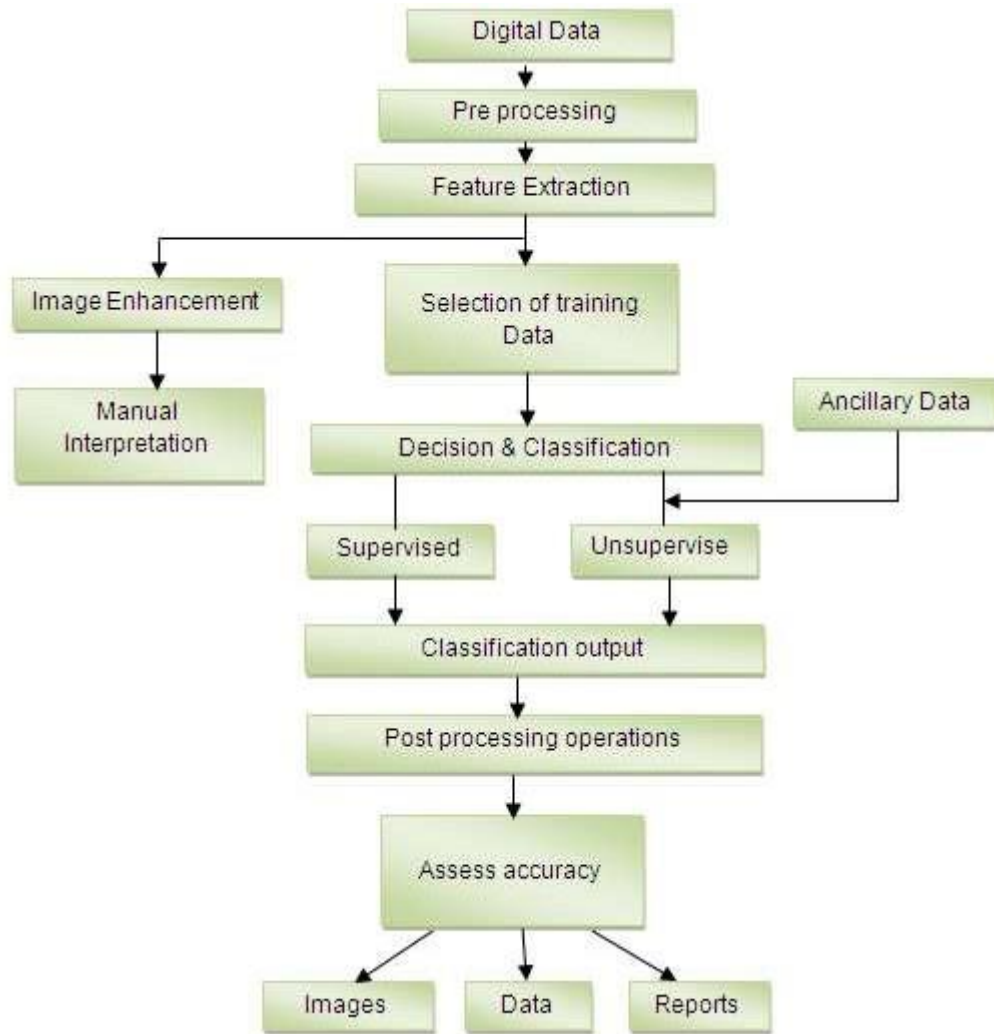


Fig.No 4.4.1 Image Processing

IMAGE PROCESSING CONCEPTS

Binary Images

Binary images are images whose pixels have only two possible intensity values. They are normally displayed as black and white. Numerically, the two values are often 0 for black, and either 1 or 255 for white.

Binary images are often produced by thresholding a grayscale or color image, in order to separate an object in the image from the background. The color of the object (usually white) is referred to as the foreground color. The rest (usually black) is referred to as the background

color. However, depending on the image which is to be thresholded, this polarity might be inverted, in which case the object is displayed with 0 and the background is with a non-zero value.

Some morphological operators assume a certain polarity of the binary input image so that if we process an image with inverse polarity the operator will have the opposite effect. For example, if we apply a closing operator to a black text on white background, the text will be opened.

Color Images

It is possible to construct (almost) all visible colors by combining the three primary colors red, green and blue, because the human eye has only three different color receptors, each of them sensible to one of the three colors. Different combinations in the stimulation of the receptors enable the human eye to distinguish approximately 350000 colors. A RGB color image is a multi-spectral image with one band for each color red, green and blue, thus producing a weighted combination of the three primary colors for each pixel.

A full 24-bit color image contains one 8-bit value for each color, thus being able to display $2^{24} = 16777216$ different colors.

However, it is computationally expensive and often not necessary to use the full 24-bit image to store the color for each pixel. Therefore, the color for each pixel is often encoded in a single byte, resulting in an 8-bit color image. The process of reducing the color representation from 24-bits to 8-bits, known as color quantization, restricts the number of possible colors to 256. However, there is normally no visible difference between a 24-bit color image and the same image displayed with 8 bits. An 8-bit color images are based on colormaps, which are look-up tables taking the 8-bit pixel value as index and providing an output value for each color.

8-bit Color Images

Full RGB color requires that the intensities of three color components be specified for each and every pixel. It is common for each component intensity to be stored as an 8-bit integer, and so each pixel requires 24 bits to completely and accurately specify its color. If this is done, then

the image is known as a 24-bit color image. However there are two problems with this approach:

- Storing 24 bits for every pixel leads to very large image files that with current technology are cumbersome to store and manipulate. For instance a 24-bit 512×512 image takes up 750KB in uncompressed form.
- Many monitor displays use colormaps with 8-bit index numbers, meaning that they can only display 256 different colors at any one time. Thus it is often wasteful to store more than 256 different colors in an image anyway, since it will not be possible to display them all on screen.

Because of this, many image formats (e.g. 8-bit GIF and TIFF) use 8-bit colormaps to restrict the maximum number of different colors to 256. Using this method, it is only necessary to store an 8-bit index into the colormap for each pixel, rather than the full 24-bit color value. Thus 8-bit image formats consist of two parts: a colormap describing what colors are present in the image, and the array of index values for each pixel in the image.

When a 24-bit full color image is turned into an 8-bit image, it is usually necessary to throw away some of the colors, a process known as color quantization. This leads to some degradation in image quality, but in practice the observable effect can be quite small, and in any case, such degradation is inevitable if the image output device (e.g. screen or printer) is only capable of displaying 256 colors or less.

The use of 8-bit images with colormaps does lead to some problems in image processing. First of all, each image has to have its own colormap, and there is usually no guarantee that each image will have exactly the same colormap. Thus on 8-bit displays it is frequently impossible to correctly display two different color images that have different colormaps at the same time. Note that in practice 8-bit images often use reduced size colormaps with less than 256 colors in order to avoid this problem.

Another problem occurs when the output image from an image processing operation contains different colors to the input image or images. This can occur very easily, as for instance when two color images are added together pixel-by-pixel. Since the output image contains different colors from the input images, it ideally needs a new colormap, different from those of the input images, and this involves further color quantization which will degrade the image quality.

Hence the resulting output is usually only an approximation of the desired output. Repeated image processing operations will continually degrade the image colors. And of course we still have the problem that it is not possible to display the images simultaneously with each other on the same 8-bit display.

Because of these problems it is to be expected that as computer storage and processing power become cheaper, there will be a shift away from 8-bit images and towards full 24-bit image processing.

24-bit Color Images

Full RGB color requires that the intensities of three color components be specified for each and every pixel. It is common for each component intensity to be stored as an 8-bit integer, and so each pixel requires 24 bits to completely and accurately specify its color. Image formats that store a full 24 bits to describe the color of each and every pixel are therefore known as 24-bit color images.

Using 24 bits to encode color information allows $2^{24} = 16777216$ different colors to be represented, and this is sufficient to cover the full range of human color perception fairly well.

The term 24-bit is also used to describe monitor displays that use 24 bits per pixel in their display memories, and which are hence capable of displaying a full range of colors.

There are also some disadvantages to using 24-bit images. Perhaps the main one is that it requires three times as much memory, disk space and processing time to store and manipulate 24-bit color images as compared to 8-bit color images. In addition, there is often not much point in being able to store all those different colors if the final output device (e.g. screen or printer) can only actually produce a fraction of them. Since it is possible to use colormaps to produce 8-bit color images that look almost as good, at the time of writing 24-bit displays are relatively little used. However it is to be expected that as the technology becomes

Color Quantization

Color quantization is applied when the color information of an image is to be reduced. The most common case is when a 24-bit color image is transformed into an 8-bit color image.

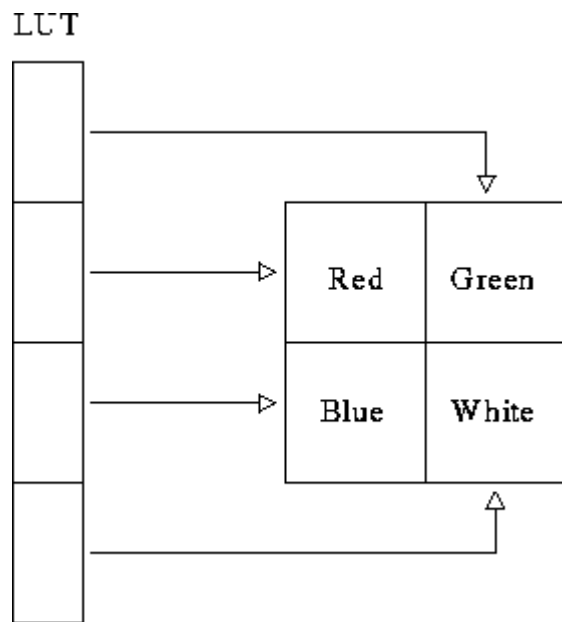
Two decisions have to be made:

1. which colors of the larger color set remain in the new image, and
2. how are the discarded colors mapped to the remaining ones.

The simplest way to transform a 24-bit color image into 8 bits is to assign 3 bits to red and green and 2 bits to blue (blue has only 2 bits, because of the eye's lower sensitivity to this color). This enables us to display 8 different shades of red and green and 4 of blue. However, this method can yield only poor results. For example, an image might contain different shades of blue which are all clustered around a certain value such that only one shade of blue is used in the 8-bit image and the remaining three blues are not used.

Alternatively, since 8-bit color images are displayed using a colormap, we can assign any arbitrary color to each of the 256 8-bit values and we can define a separate colormap for each image. This enables us perform a color quantization adjusted to the data contained in the image. One common approach is the popularity algorithm, which creates a histogram of all colors and retains the 256 most frequent ones. Another approach, known as the median-cut algorithm, yields even better results but also needs more computation time. This technique recursively fits a box around all colors used in the RGB colorspace which it splits at the median value of its longest side. The algorithm stops after 255 recursions. All colors in one box are mapped to the centroid of this box.

All above techniques restrict the number of displayed colors to 256. A technique of achieving additional colors is to apply a variation of half-toning used for gray scale images, thus increasing the color resolution at the cost of spatial resolution. The 256 values of the colormap are divided into four sections containing 64 different values of red, green, blue and white. As can be seen in Figure 1, a 2×2 pixel area is grouped together to represent one composite color, each of the four pixels displays either one of the primary colors or white. In this way, the number of possible colors is increased from 256 to 64^4 .



A 2×2 pixel area displaying one composite color

cheaper, their use in image processing will grow.

Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of 'multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image processing context, one of the input arrays is normally just a graylevel image. The second array is usually much smaller, and is also two-dimensional (although it may be just a single pixel thick), and is known as the kernel. Figure 1 shows an example image and kernel that we will use to illustrate convolution.

I₁₁	I₁₂	I₁₃	I₁₄	I₁₅	I₁₆	I₁₇	I₁₈	I₁₉
I₂₁	I₂₂	I₂₃	I₂₄	I₂₅	I₂₆	I₂₇	I₂₈	I₂₉
I₃₁	I₃₂	I₃₃	I₃₄	I₃₅	I₃₆	I₃₇	I₃₈	I₃₉
I₄₁	I₄₂	I₄₃	I₄₄	I₄₅	I₄₆	I₄₇	I₄₈	I₄₉
I₅₁	I₅₂	I₅₃	I₅₄	I₅₅	I₅₆	I₅₇	I₅₈	I₅₉
I₆₁	I₆₂	I₆₃	I₆₄	I₆₅	I₆₆	I₆₇	I₆₈	I₆₉

K₁₁	K₁₂	K₁₃
K₂₁	K₂₂	K₂₃

Figure 1 An example small image (left) and kernel (right) to illustrate convolution. The labels within each grid square are used to identify each square.

The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. (Note that implementations differ in what they do at the edges of images, as explained below.) Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together.

So, in our example, the value of the bottom right pixel in the output image will be given by:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$

If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will have M - m + 1 rows, and N - n + 1 columns.

Mathematically we can write the convolution as:

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i + k - 1, j + l - 1)K(k, l)$$

where i runs from 1 to $M - m + 1$ and j runs from 1 to $N - n + 1$.

Note that many implementations of convolution produce a larger output image than this because they relax the constraint that the kernel can only be moved to positions where it fits entirely within the image. Instead, these implementations typically slide the kernel to all positions where just the top left corner of the kernel is within the image. Therefore the kernel 'overlaps' the image on the bottom and right edges. One advantage of this approach is that the output image is the same size as the input image. Unfortunately, in order to calculate the output pixel values for the bottom and right edges of the image, it is necessary to invent input pixel values for places where the kernel extends off the end of the image. Typically pixel values of zero are chosen for regions outside the true image, but this can often distort the output image at these places. Therefore in general if you are using a convolution implementation that does this, it is better to clip the image to remove these spurious regions. Removing $n - 1$ pixels from the right hand side and $m - 1$ pixels from the bottom will fix things.

Convolution can be used to implement many different operators, particularly spatial filters and feature detectors. Examples include Gaussian smoothing and the Sobel edge detector.

Distance Metrics

It is often useful in image processing to be able to calculate the distance between two pixels in an image, but this is not as straightforward as it seems. The presence of the pixel grid makes several so-called distance metrics possible which often give different answers to each other for the distance between the same pair of points. We consider the three most important ones.

Euclidean Distance

This is the familiar straight line distance that most people are familiar with. If the two pixels that we are considering have coordinates (x_1, y_1) and (x_2, y_2) , then the Euclidean distance is given by:

$$D_{Euclid} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

City Block Distance

Also known as the Manhattan distance. This metric assumes that in going from one pixel to the other it is only possible to travel directly along pixel grid lines. Diagonal moves are not allowed. Therefore the 'city block' distance is given by:

$$D_{City} = |x_2 - x_1| + |y_2 - y_1|$$

Chessboard Distance

This metric assumes that you can make moves on the pixel grid as if you were a King making moves in chess, *i.e.* a diagonal move counts the same as a horizontal move. This means that the metric is given by:

$$D_{Chess} = \max(|x_2 - x_1|, |y_2 - y_1|)$$

Note that the last two metrics are usually much faster to compute than the Euclidean metric and so are sometimes used where speed is critical but accuracy is not too important.

Dithering

Dithering is an image display technique that is useful for overcoming limited display resources. The word dither refers to a random or semi-random perturbation of the pixel values.

Two applications of this techniques are particularly useful:

🟡 Low quantization display: When images are quantized to a few bits (e.g. 3) then only a limited number of graylevels are used in the display of the image. If the scene is smoothly shaded, then the image display will generate rather distinct boundaries around the edges of image regions when the original scene intensity moves from one quantization level to the next. To eliminate this effect, one dithering technique adds random noise (with a small range of values) to the input signal before quantization into the output range. This randomizes the quantization of the pixels at the original quantization boundary, and thus pixels make a more gradual transition from neighborhoods containing 100% of the first quantization level to neighborhoods containing 100% of the second quantization level.

🟡 Limited color display: When fewer colors are able to be displayed (e.g. 256) than are present in the input image (e.g. 24 bit color), then patterns of adjacent pixels are used to simulate the appearance of the unrepresented colors.

Edge Detectors

Edges are places in the image with strong intensity contrast. Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation when we want to divide the image into areas corresponding to different objects. Representing an image by its edges has the further advantage that the amount of data is reduced significantly while retaining most of the image information.

Since edges consist of mainly high frequencies, we can, in theory, detect edges by applying a highpass frequency filter in the Fourier domain or by convolving the image with an appropriate kernel in the spatial domain. In practice, edge detection is performed in the spatial domain, because it is computationally less expensive and often yields better results.

Since edges correspond to strong illumination gradients, we can highlight them by calculating the derivatives of the image. This is illustrated for the one-dimensional case in Figure 1.

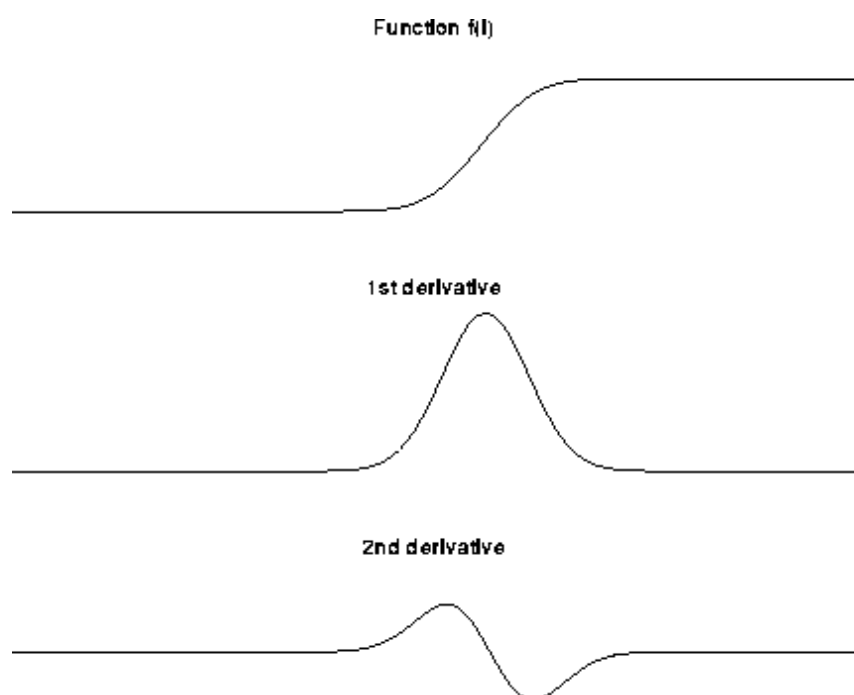


Figure 1 1st and 2nd derivative of an edge illustrated in one dimension.

We can see that the position of the edge can be estimated with the maximum of the 1st derivative or with the zero-crossing of the 2nd derivative. Therefore we want to find a technique to calculate the derivative of a two-dimensional image. For a discrete one-dimensional function $f(i)$, the first derivative can be approximated by

$$\frac{df(i)}{di} = f(i+1) - f(i)$$

Calculating this formula is equivalent to convolving the function with $[-1 \ 1]$. Similarly the 2nd derivative can be estimated by convolving $f(i)$ with $[1 \ -2 \ 1]$.

Different edge detection kernels which are based on the above formula enable us to calculate either the 1st or the 2nd derivative of a two-dimensional image. There are two common approaches to estimate the 1st derivative in a two-dimensional image, Prewitt compass edge detection and gradient edge detection.

Prewitt compass edge detection involves convolving the image with a set of (usually 8) kernels, each of which is sensitive to a different edge orientation. The kernel producing the maximum response at a pixel location determines the edge magnitude and orientation. Different sets of kernels might be used: examples include Prewitt, Sobel, Kirsch and Robinson kernels.

Gradient edge detection is the second and more widely used technique. Here, the image is convolved with only two kernels, one estimating the gradient in the x-direction, G_x , the other the gradient in the y-direction, G_y . The absolute gradient magnitude is then given by

$$|G| = \sqrt{G_x^2 + G_y^2}$$

and is often approximated with

$$|G| = |G_x| + |G_y|$$

In many implementations, the gradient magnitude is the only output of a gradient edge detector, however the edge orientation might be calculated with

The most common kernels used for the gradient edge detector are the Sobel, Roberts Cross and Prewitt operators.

After having calculated the magnitude of the 1st derivative, we now have to identify those pixels corresponding to an edge. The easiest way is to threshold the gradient image, assuming that all pixels having a local gradient above the threshold must represent an edge. An alternative technique is to look for local maxima in the gradient image, thus producing one pixel wide edges. A more sophisticated technique is used by the Canny edge detector. It first applies a gradient edge detector to the image and then finds the edge pixels using non-maximal suppression and hysteresis tracking.

An operator based on the 2nd derivative of an image is the Marr edge detector, also known as zero crossing detector. Here, the 2nd derivative is calculated using a Laplacian of Gaussian (LoG) filter. The Laplacian has the advantage that it is an isotropic measure of the 2nd derivative of an image, i.e. the edge magnitude is obtained independently from the edge orientation by convolving the image with only one kernel. The edge positions are then given by the zero-crossings in the LoG image. The scale of the edges which are to be detected can be controlled by changing the variance of the Gaussian.

A general problem for edge detection is its sensitivity to noise, the reason being that calculating the derivative in the spatial domain corresponds to accentuating high frequencies and hence magnifying noise. This problem is addressed in the Canny and Marr operators by convolving the image with a smoothing operator (Gaussian) before calculating the derivative.

Frequency Domain

For simplicity, assume that the image I being considered is formed by projection from scene S (which might be a two- or three-dimensional scene, etc.).

The frequency domain is a space in which each image value at image position F represents the amount that the intensity values in image I vary over a specific distance related to F . In the

frequency domain, changes in image position correspond to changes in the spatial frequency, (or the rate at which image intensity values) are changing in the spatial domain image I.

For example, suppose that there is the value 20 at the point that represents the frequency 0.1 (or 1 period every 10 pixels). This means that in the corresponding spatial domain image I the intensity values vary from dark to light and back to dark over a distance of 10 pixels, and that the contrast between the lightest and darkest is 40 gray levels (2 times 20).

The spatial frequency domain is interesting because: 1) it may make explicit periodic relationships in the spatial domain, and 2) some image processing operators are more efficient or indeed only practical when applied in the frequency domain.

In most cases, the Fourier Transform is used to convert images from the spatial domain into the frequency domain and vice-versa.

A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances.

Grayscale Images

A grayscale (or graylevel) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. In fact a 'gray' color is one in which the red, green and blue components all have equal intensity in RGB space, and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full color image.

Often, the grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. If the levels are evenly spaced then the difference between successive graylevels is significantly better than the graylevel resolving power of the human eye.

Grayscale images are very common, in part because much of today's display and image capture hardware can only support 8-bit images. In addition, grayscale images are entirely sufficient

for many tasks and so there is no need to use more complicated and harder-to-process color images.

Image Editing Software

There is a huge variety of software for manipulating images in various ways. Much of this software can be grouped under the heading image processing software, and the bulk of this reference is concerned with that group.

Another very important category is what we call image editing software. This group includes painting programs, graphic art packages and so on. They are often useful in conjunction with image processing software packages, in situations where direct immediate interaction with an image is the easiest way of achieving something. For instance, if a region of an image is to be masked out for subsequent image processing, it may be easiest to create the mask using an art package by directly drawing on top of the original image. The mask used in the description of the AND operator was created this way for instance. Art packages also often allow the user to move sections of the images around and brighten or darken selected regions interactively. Few dedicated image processing packages offer the same flexibility and ease of use in this respect.

Idempotence

Some operators have the special property that applying them more than once to the same image produces no further change after the first application. Such operators are said to be idempotent. Examples include the morphological operators opening and closing.

Isotropic Operators

An isotropic operator in an image processing context is one which applies equally well in all directions in an image, with no particular sensitivity or bias towards one particular set of directions (e.g. compass directions). A typical example is the zero crossing edge detector which responds equally well to edges in any orientation. Another example is Gaussian smoothing. It should be borne in mind that although an operator might be isotropic in theory, the actual implementation of it for use on a discrete pixel grid may not be perfectly isotropic. An example of this is a Gaussian smoothing filter with very small standard deviation on a square grid.

Kernel

A kernel is a (usually) smallish matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers give rise to different results under convolution. For instance, Figure 1 shows a 3×3 kernel that implements a mean filter.

1	1	1
1	1	1
1	1	1

Set of coordinate points =

{ (-1, -1), (0, -1), (1, -1),
(-1, 0), (0, 0), (1, 0),
(-1, 1), (0, 1), (1, 1) }

Figure 1 Convolution kernel for a mean filter with 3×3 neighborhood.

The word 'kernel' is also commonly used as a synonym for 'structuring element', which is a similar object used in mathematical morphology. A structuring element differs from a kernel in that it also has a specified origin. This sense of the word 'kernel' is not used in HIPR.

Logical Operators

Logical operators are generally derived from Boolean algebra, which is a mathematical way of manipulating the truth values of concepts in an abstract way without bothering about what the concepts actually mean. The truth value of a concept in Boolean value can have just one of two possible values: true or false. Boolean algebra allows you to represent things like:

The block is both red and large

by something like:

A AND B

where A represents 'The block is red', and B represents 'The block is large'. Now each of these sub-phrases has its own truth value in any given situation: each sub-phrase is either true or false. Moreover, the entire composite phrase also has a truth value: it is true if both of the sub-phrases are true, and false in any other case. We can write this AND combination rule (and its dual operation NAND) using a truth-table as shown in Figure 1, in which we conventionally represent true by 1, and false by zero.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Figure 1 Truth-tables for AND and NAND

The left hand table shows each of the possible combinations of truth values of A and B, and the the resulting truth value of A AND B. Similar truth-tables can be set up for the other logical operators: NAND, OR, NOR, XOR, XNOR and NOT.

Turning now to an image processing context, the pixel values in a binary image, which are either 0 or 1, can be interpreted as truth values as above. Using this convention we can carry out logical operations on images simply by applying the truth-table combination rules to the pixel values from a pair of input images (or a single input image in the case of NOT). Normally, corresponding pixels from each of two identically sized binary input images are compared to produce the output image, which is another binary image of the same size. As with other image

arithmetic operations, it is also possible to logically combine a single input image with a constant logical value, in which case each pixel in the input image is compared to the same constant in order to produce the corresponding output pixel. See the individual logical operator descriptions for examples of these operations.

Logical operations can also be carried out on images with integer pixel values. In this extension the logical operations are normally carried out in bitwise fashion on binary representations of those integers, comparing corresponding bits with corresponding bits to produce the output pixel value. For instance, suppose that we wish to XOR the integers 47 and 255 together using 8-bit integers. 47 is 00101111 in binary and 255 is 11111111. XORing these together in bitwise fashion, we have 11010000 in binary or 208 in decimal.

Note that not all implementations of logical operators work in such bitwise fashion. For instance some will treat zero as false and any non-zero value as true and will then apply the conventional 1-bit logical functions to derive the output image. The output may be a simple binary image itself, or it may be a graylevel image formed perhaps by multiplying what would be the binary output image (containing 0's and 1's) with one of the input images.

Look-up Tables and Colormaps

Look-Up Tables or LUTs are fundamental to many aspects of image processing. An LUT is simply a table of cross-references linking index numbers to output values. The most common use is to determine the colors and intensity values with which a particular image will be displayed, and in this context the LUT is often called simply a colormap.

The idea behind the colormap is that instead of storing a definite color for each pixel in an image, for instance in 24-bit RGB format, each pixel's value is instead treated as an index number into the colormap. When the image is to be displayed or otherwise processed, the colormap is used to look up the actual colors corresponding to each index number. Typically, the output values stored in the LUT would be RGB color values.

There are two main advantages to doing things this way. Firstly, the index number can be made to use fewer bits than the output value in order to save storage space. For instance an 8-bit index number can be used to look up a 24-bit RGB color value in the LUT. Since only the 8-bit index number needs to be stored for each pixel, such 8-bit color images take up less space

than a full 24-bit image of the same size. Of course the image can only contain 256 different colors (the number of entries in an 8-bit LUT), but this is sufficient for many applications and usually the observable image degradation is small.

Secondly the use of a color table allows the user to experiment easily with different color labeling schemes for an image.

One disadvantage of using a colormap is that it introduces additional complexity into an image format. It is usually necessary for each image to carry around its own colormap, and this LUT must be continually consulted whenever the image is displayed or processed.

Another problem is that in order to convert from a full color image to (say) an 8-bit color image using a color image, it is usually necessary to throw away many of the original colors, a process known as color quantization. This process is lossy, and hence the image quality is degraded during the quantization process. Additionally, when performing further image processing on such images, it is frequently necessary to generate a new colormap for the new images, which involves further color quantization, and hence further image degradation.

As well as their use in colormaps, LUTs are often used to remap the pixel values within an image. This is the basis of many common image processing point operations such as thresholding, gamma correction and contrast stretching. The process is often referred to as anamorphosis.

Masking

A mask is a binary image consisting of zero- and non-zero values. If a mask is applied to another binary or to a grayscale image of the same size, all pixels which are zero in the mask are set to zero in the output image. All others remain unchanged.

Masking can be implemented either using pixel multiplication or logical AND, the latter in general being faster.

Masking is often used to restrict a point or arithmetic operator to an area defined by the mask. We can, for example, accomplish this by first masking the desired area in the input image and processing it with the operator, then masking the original input image with the inverted mask to obtain the unprocessed area of the image and finally recombining the two partial images

using image addition. An example can be seen in the worksheet on the logical AND operator. In some image processing packages, a mask can directly be defined as an optional input to a point operator, so that automatically the operator is only applied to the pixels defined by the mask .

Mathematical Morphology

The field of mathematical morphology contributes a wide range of operators to image processing, all based around a few simple mathematical concepts from set theory. The operators are particularly useful for the analysis of binary images and common usages include edge detection, noise removal, image enhancement and image segmentation.

The two most basic operations in mathematical morphology are erosion and dilation. Both of these operators take two pieces of data as input: an image to be eroded or dilated, and a structuring element (also known as a kernel). The two pieces of input data are each treated as representing sets of coordinates in a way that is slightly different for binary and grayscale images.

For a binary image, white pixels are normally taken to represent foreground regions, while black pixels denote background. (Note that in some implementations this convention is reversed, and so it is very important to set up input images with the correct polarity for the implementation being used). Then the set of coordinates corresponding to that image is simply the set of two-dimensional Euclidean coordinates of all the foreground pixels in the image, with an origin normally taken in one of the corners so that all coordinates have positive elements.

For a grayscale image, the intensity value is taken to represent height above a base plane, so that the grayscale image represents a surface in three-dimensional Euclidean space. Figure 1 shows such a surface. Then the set of coordinates associated with this image surface is simply the set of three-dimensional Euclidean coordinates of all the points within this surface and also all points below the surface, down to the base plane. Note that even when we are only considering points with integer coordinates, this is a lot of points, so usually algorithms are employed that do not need to consider all the points.

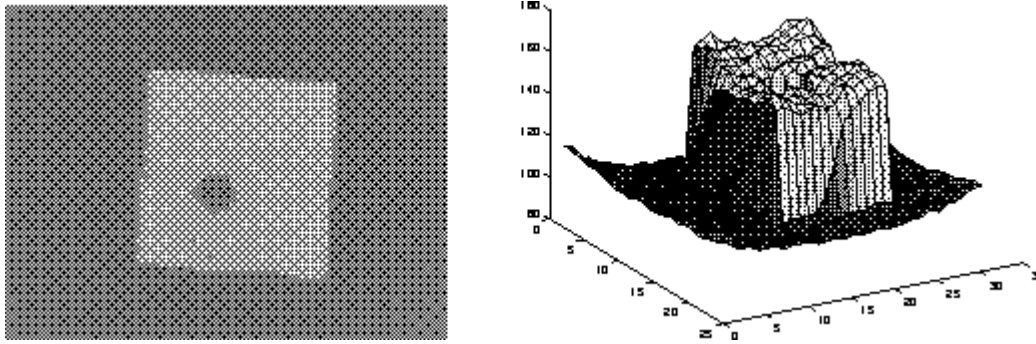


Figure 4.4.2 Simple graylevel image and the corresponding surface in image space

The structuring element is already just a set of point coordinates (although it is often represented as a binary image). It differs from the input image coordinate set in that it is normally much smaller, and its coordinate origin is often not in a corner, so that some coordinate elements will have negative values. Note that in many implementations of morphological operators, the structuring element is assumed to be a particular shape (e.g. a 3×3 square) and so is hardwired into the algorithm.

Binary morphology can be seen as a special case of graylevel morphology in which the input image has only two graylevels at values 0 and 1.

Erosion and dilation work (at least conceptually) by translating the structuring element to various points in the input image, and examining the intersection between the translated kernel coordinates and the input image coordinates. For instance, in the case of erosion, the output coordinate set consists of just those points to which the origin of the structuring element can be translated, while the element still remains entirely 'within' the input image.

Virtually all other mathematical morphology operators can be defined in terms of combinations of erosion and dilation along with set operators such as intersection and union. Some of the more important are opening, closing and skeletonization.

Multi-spectral Images

A multi-spectral image is a collection of several monochrome images of the same scene, each of them taken with a different sensor. Each image is referred to as a band. A well known multi-spectral (or multi-band image) is a RGB color image, consisting of a red, a green and a blue image, each of them taken with a sensor sensitive to a different wavelength. In image processing, multi-spectral images are most commonly used for Remote Sensing applications. Satellites usually take several images from frequency bands in the visual and non-visual range. Landsat 5, for example, produces 7 band images with the wavelength of the bands being between 450 and 1250 nm.

All the standard single-band image processing operators can also be applied to multi-spectral images by processing each band separately. For example, a multi-spectral image can be edge detected by finding the edges in each band and then ORing the three edge images together. However, we would obtain more reliable edges, if we associate a pixel with an edge based on its properties in all three bands and not only in one.

To fully exploit the additional information which is contained in the multiple bands, we should consider the images as one multi-spectral image rather than as a set of monochrome graylevel images. For an image with k bands, we can then describe the brightness of each pixel as a point in a k -dimensional space represented by a vector of length k .

Special techniques exist to process multi-spectral images. For example, to classify a pixel as belonging to one particular region, its intensities in the different bands are said to form a feature vector describing its location in the k -dimensional feature space. The simplest way to define a class is to choose a upper and lower threshold for each band, thus producing a k -dimensional 'hyper-cube' in the feature space. Only if the feature vector of a pixel points to a location within this cube, is the pixel classified as belonging to this class. A more sophisticated classification method is described in the corresponding worksheet.

The disadvantage of multi-spectral images is that, since we have to process additional data, the required computation time and memory increase significantly. However, since the speed of the hardware will increase and the costs for memory will decrease in the future, it can be expected that multi-spectral images will become more important in many fields of computer vision.

Non-linear Filtering

Suppose that an image processing operator F acting on two input images A and B produces output images C and D respectively. If the operator F is linear, then

$$F(a \times A + b \times B) = a \times C + b \times D$$

where a and b are constants. In practice, this means that each pixel in the output of a linear operator is the weighted sum of a set of pixels in the input image.

By contrast, non-linear operators are all the other operators. For example, the threshold operator is non-linear, because individually, corresponding pixels in the two images A and B may be below the threshold, whereas the pixel obtained by adding A and B may be above threshold. Similarly, an absolute value operation is non-linear:

$$|-1 + 1| \neq |-1| + |1|$$

as is the exponential operator:

$$\exp(1 + 1) \neq \exp(1) + \exp(1)$$

Pixels

In order for any digital computer processing to be carried out on an image, it must first be stored within the computer in a suitable form that can be manipulated by a computer program. The most practical way of doing this is to divide the image up into a collection of discrete (and usually small) cells, which are known as pixels. Most commonly, the image is divided up into a rectangular grid of pixels, so that each pixel is itself a small rectangle. Once this has been done, each pixel is given a pixel value that represents the color of that pixel. It is assumed that the whole pixel is the same color, and so any color variation that did exist within the area of the pixel before the image was discretized is lost. However, if the area of each pixel is very small, then the discrete nature of the image is often not visible to the human eye.

Other pixel shapes and formations can be used, most notably the hexagonal grid, in which each pixel is a small hexagon. This has some advantages in image processing, including the fact that pixel connectivity is less ambiguously defined than with a square grid, but hexagonal grids are not widely used. Part of the reason is that many image capture systems (e.g. most CCD cameras and scanners) intrinsically discretize the captured image into a rectangular grid in the first instance.

Pixel Connectivity

The notation of pixel connectivity describes a relation between two or more pixels. For two pixels to be connected they have to fulfill certain conditions on the pixel brightness and spatial adjacency.

First, in order for two pixels to be considered connected, their pixel values must both be from the same set of values V . For a grayscale image, V might be any range of graylevels, e.g. $V=\{22,23,...40\}$, for a binary image we simple have $V=\{1\}$.

To formulate the adjacency criterion for connectivity, we first introduce the notation of neighborhood. For a pixel p with the coordinates (x,y) the set of pixels given by:

$$N_4(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$$

is called its 4-neighbors. Its 8-neighbors are defined as

$$N_8(p) = N_4 \cup \{(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}$$

From this we can infer the definition for 4- and 8-connectivity:

Two pixels p and q , both having values from a set V are 4-connected if q is from the set $N_4(p)$ and 8-connected if q is from $N_8(p)$.

General connectivity can either be based on 4- or 8-connectivity; for the following discussion we use 4-connectivity.

A pixel p is connected to a pixel q if p is 4-connected to q or if p is 4-connected to a third pixel which itself is connected to q . Or, in other words, two pixels q and p are connected if there is a path from p and q on which each pixel is 4-connected to the next one.

A set of pixels in an image which are all connected to each other is called a connected component. Finding all connected components in an image and marking each of them with a distinctive label is called connected component labeling.

An example of a binary image with two connected components which are based on 4-connectivity can be seen in Figure 1. If the connectivity were based on 8-neighbors, the two connected components would merge into one.

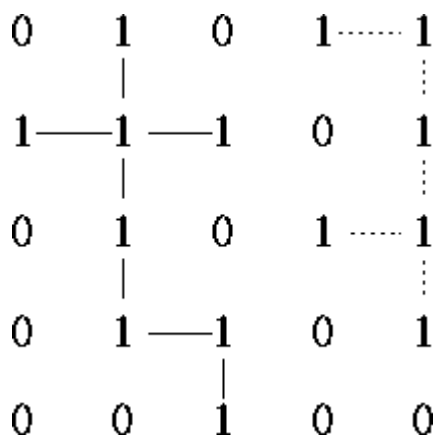


Figure 1 Two connected components based on 4-connectivity.

Pixel Values

Each of the pixels that represents an image stored inside a computer has a pixel value which describes how bright that pixel is, and/or what color it should be. In the simplest case of binary images, the pixel value is a 1-bit number indicating either foreground or background. For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.

To represent color images, separate red, green and blue components must be specified for each pixel (assuming an RGB colorspace), and so the pixel 'value' is actually a vector of three numbers. Often the three different components are stored as three separate 'grayscale' images known as color planes (one for each of red, green and blue), which have to be recombined when displaying or processing.

Multi-spectral images can contain even more than three components for each pixel, and by extension these are stored in the same kind of way, as a vector pixel value, or as separate color planes.

The actual grayscale or color component intensities for each pixel may not actually be stored explicitly. Often, all that is stored for each pixel is an index into a colormap in which the actual intensity or colors can be looked up.

Although simple 8-bit integers or vectors of 8-bit integers are the most common sorts of pixel values used, some image formats support different types of value, for instance 32-bit signed integers or floating point values. Such values are extremely useful in image processing as they allow processing to be carried out on the image where the resulting pixel values are not necessarily 8-bit integers. If this approach is used then it is usually necessary to set up a colormap which relates particular ranges of pixel values to particular displayed colors.

Primary Colors

It is a useful fact that the huge variety of colors that can be perceived by humans can all be produced simply by adding together appropriate amounts of red, blue and green colors. These colors are known as the primary colors. Thus in most image processing applications, colors are represented by specifying separate intensity values for red, green and blue components. This representation is commonly referred to as RGB.

The primary color phenomenon results from the fact that humans have three different sorts of color receptors in their retinas which are each most sensitive to different visible light wavelengths.

The primary colors used in painting (red, yellow and blue) are different. When paints are mixed, the 'addition' of a new color paint actually subtracts wavelengths from the reflected visible light.

RGB and Colorspaces

A color perceived by the human eye can be defined by a linear combination of the three primary colors red, green and blue. These three colors form the basis for the RGB-colorspace. Hence, each perceivable color can be defined by a vector in the three-dimensional colorspace. The intensity is given by the length of the vector, and the actual color by the two angles describing the orientation of the vector in the colorspace.

The RGB-space can also be transformed into other coordinate systems, which might be more useful for some applications. One common basis for the color space is IHS. In this coordinate system, a color is described by its intensity, hue (average wavelength) and saturation (the amount of white in the color). This color space makes it easier to directly derive the intensity and color of perceived light and is therefore more likely to be used by human beings.

Spatial Domain

For simplicity, assume that the image I being considered is formed by projection from scene S (which might be a two- or three-dimensional scene, etc.).

The spatial domain is the normal image space, in which a change in position in I directly projects to a change in position in S . Distances in I (in pixels) correspond to real distances (e.g. in meters) in S .

This concept is used most often when discussing the frequency with which image values change, that is, over how many pixels does a cycle of periodically repeating intensity variations occur. One would refer to the number of pixels over which a pattern repeats (its periodicity) in the spatial domain.

In most cases, the Fourier Transform will be used to convert images from the spatial domain into the frequency domain.

A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances.

Another term used in this context is spatial derivative, which refers to how much the image intensity values change per change in image position.

Structuring Elements

The field of mathematical morphology provides a number of important image processing operations, including erosion, dilation, opening and closing. All these morphological operators take two pieces of data as input. One is the input image, which may be either binary or grayscale for most of the operators. The other is the structuring element. It is this that determines the precise details of the effect of the operator on the image.

The structuring element is sometimes called the kernel, but we reserve that term for the similar objects used in convolutions.

The structuring element consists of a pattern specified as the coordinates of a number of discrete points relative to some origin. Normally cartesian coordinates are used and so a convenient way of representing the element is as a small image on a rectangular grid. Figure 1 shows a number of different structuring elements of various sizes. In each case the origin is marked by a ring around that point. The origin does not have to be in the center of the structuring element, but often it is. As suggested by the figure, structuring elements that fit into a 3×3 grid with its origin at the center are the most commonly seen type.

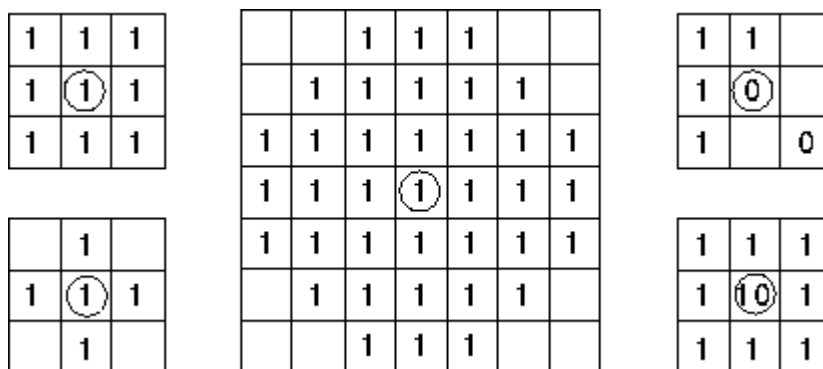


Figure 1 Some example structuring elements.

Note that each point in the structuring element may have a value. In the simplest structuring elements used with binary images for operations such as erosion, the elements only have one value, conveniently represented as a one. More complicated elements, such as those used with thinning or grayscale morphological operations, may have other pixel values.

An important point to note is that although a rectangular grid is used to represent the structuring element, not every point in that grid is part of the structuring element in general. Hence the elements shown in Figure 1 contain some blanks. In many texts, these blanks are represented as zeros, but this can be confusing and so we avoid it here.

When a morphological operation is carried out, the origin of the structuring element is typically translated to each pixel position in the image in turn, and then the points within the translated structuring element are compared with the underlying image pixel values. The details of this comparison, and the effect of the outcome depend on which morphological operator is being used.

Wrapping and Saturation

If an image is represented in a byte or integer pixel format, the maximum pixel value is limited by the number of bits used for the representation, e.g. the pixel values of a 8-bit image are limited to 255.

However, many image processing operations produce output values which are likely to exceed the given maximum value. In such cases, we have to decide how to handle this pixel overflow.

One possibility is to wrap around the overflowing pixel values. This means that if a value is greater than the possible maximum, we subtract the pixel value range so that the value starts again from the possible minimum value. Figure 1 shows the mapping function for wrapping the output values of some operation into an 8-bit format.

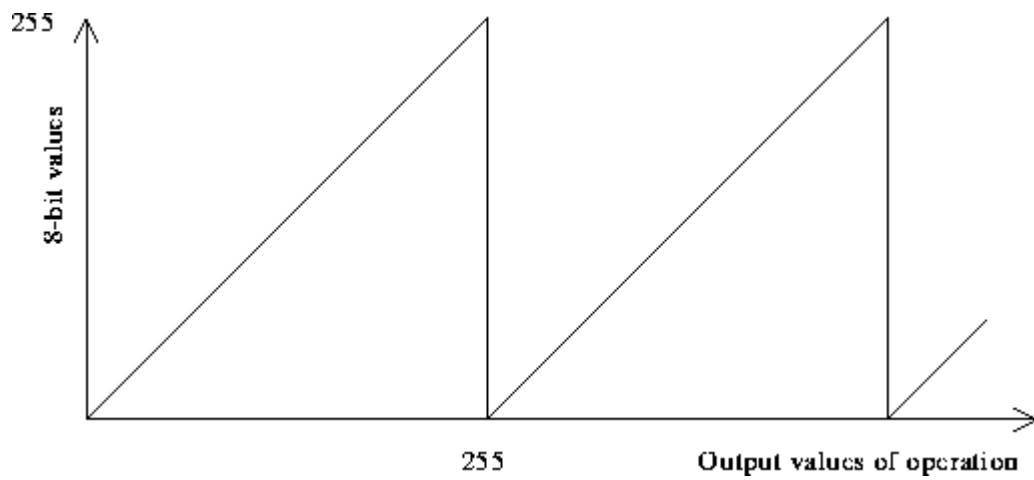


Figure 1 Mapping function for wrapping the pixel values of an 8-bit image.

Another possibility is to set all overflowing pixels to the maximum possible values --- an effect known as saturation. The corresponding mapping function for an 8-bit image can be seen in Figure 2.

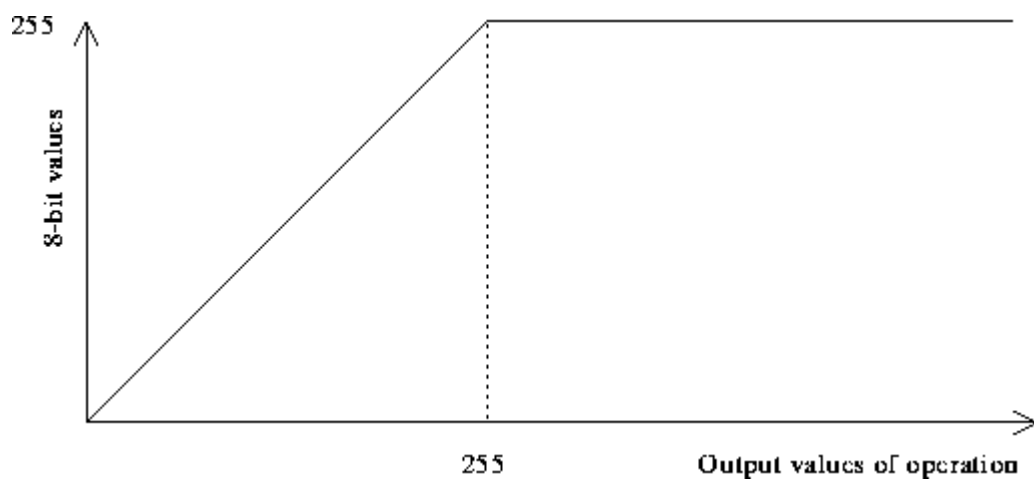


Figure 2 Mapping function for saturating an 8-bit image.

If only a few pixels in the image exceed the maximum value it is often better to apply the latter technique, especially if we use the image for display purposes. However, by setting all overflowing pixels to the same value we lose an essential amount of information. In the worst case, when all pixels exceed the maximum value, this would lead to an image of constant pixel values. Wrapping around overflowing pixel retains the differences between values. On the other hand, it might cause the problem that pixel values passing the maximum 'jump' from the maximum to the minimum value. Examples for both techniques can be seen in the worksheets of various point operators.

If possible, it is easiest to change the image format, for example to float format, so that all pixel values can be represented. However, we should keep in mind that this implies an increase in processing time and memory.

CHAPTER 5

IMPLEMENTATION

5.1 MODULES

- ❖ Dataset
- ❖ Importing the necessary libraries
- ❖ Retrieving the images
- ❖ Splitting the dataset
- ❖ Building the model
- ❖ Apply the model and plot the graphs for accuracy and loss
- ❖ Accuracy on test set
- ❖ Saving the Trained Model

5.1.1 MODULES DESCRIPTION:

Dataset:

In the first module, we developed the system to get the input dataset for the training and testing purpose. We have taken the dataset from Brain Tumor Detection Using Image Processing

Link — <https://www.cancerimagingarchive.net/collections/>

The dataset consists of 1411 brain tumour images

Importing the necessary libraries:

We will be using Python language for this. First we will import the necessary libraries such as keras for building the main model, sklearn for splitting the training and test data, PIL for converting the images into array of numbers and other libraries such as pandas, numpy ,matplotlib and tensorflow.

Retrieving the images:

We will retrieve the images and their labels. Then resize the images to (224,224) as all images should have same size for recognition. Then convert the images into numpy array.

Splitting the dataset:

Split the dataset into train and test. 80% train data and 20% test data.

Building the model:

For building the we will use sequential model from keras library. Then we will add the layers to make convolutional neural network. In the first 2 Conv2D layers we have used 32 filters and the kernel size is (5,5).

In the MaxPool2D layer we have kept pool size (2,2) which means it will select the maximum value of every 2 x 2 area of the image. By doing this dimensions of the image will reduce by factor of 2. In dropout layer we have kept dropout rate = 0.25 that means 25% of neurons are removed randomly.

We apply these 3 layers again with some change in parameters. Then we apply flatten layer to convert 2-D data to 1-D vector. This layer is followed by dense layer, dropout layer and dense layer again. The last dense layer outputs 2 nodes as the brain tumour or not. This layer uses the softmax activation function which gives probability value and predicts which of the 2 options has the highest probability.

Apply the model and plot the graphs for accuracy and loss:

We will compile the model and apply it using fit function. The batch size will be 2. Then we will plot the graphs for accuracy and loss. We got average validation accuracy of 100% and average training accuracy of 98.7%.

Accuracy on test set:

We got a accuracy of 100% on test set

Saving the Trained Model:

Once you're confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or .pkl file using a library like pickle .

5.2 K-means Clustering

- K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.
- K-means is a centroid-based clustering algorithm, where we calculate the distance between each data point and a centroid to assign it to a cluster. The goal is to identify the K number of groups in the dataset.
- Here, we divide a data space into K clusters and assign a mean value to each. The data points are placed in the clusters closest to the mean value of that cluster. There are several distance metrics available that can be used to calculate the distance.

5.2.1 Computer Vision

Some of the computer vision problems which we will be solving in this article are:

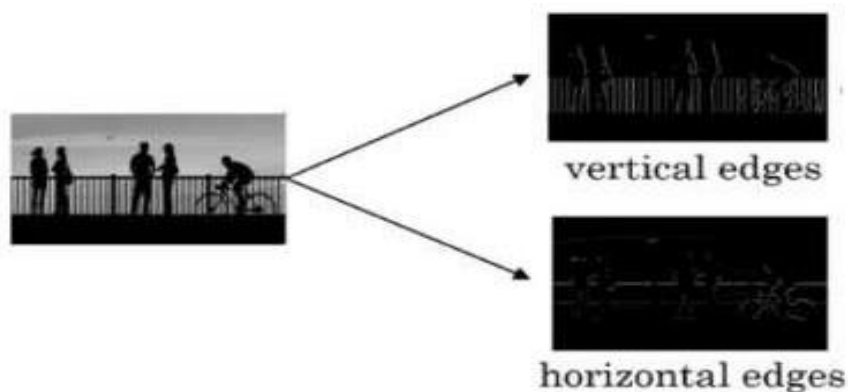
1. Image classification
2. Object detection
3. Neural style transfer

One major problem with computer vision problems is that the input data can get really big. Suppose an image is of the size 68 X 68 X 3. The input feature dimension then becomes 12,288. This will be even bigger if we have larger images (say, of size 720 X 720 X 3). Now, if we pass such a big input to a neural network, the number of parameters will swell up to a HUGE

number (depending on the number of hidden layers and hidden units). This will result in more computational and memory requirements – not something most of us can deal with.

Edge Detection Example

In the previous article, we saw that the early layers of a neural network detect edges from an image. Deeper layers might be able to detect the cause of the objects and even more deeper layers might detect the cause of complete objects (like a person's face). In this section, we will focus on how the edges can be detected from an image. Suppose we are given the below image: As you can see, there are many vertical and horizontal edges in the image. The first thing to do is to detect these edges:



But how do we detect these edges? To illustrate this, let's take a 6 X 6 grayscale image (i.e. only one channel):

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Next, we convolve this 6 X 6 matrix with a 3 X 3 filter:

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6 image



1	0	-1
1	0	-1
1	0	-1

3 X 3 filter

After the convolution, we will get a 4 X 4 image. The first element of the 4 X 4 matrix will be calculated as:

3 ¹	0 ⁰	1 ⁻¹
1 ¹	5 ⁰	8 ⁻¹
2 ¹	7 ⁰	2 ⁻¹

Similarly, we will convolve over the entire image and get a 4 X 4 output:

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

So, convolving a 6 X 6 input with a 3 X 3 filter gave us an output of 4 X 4. Consider one more example:

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

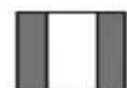
1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



*



Note: Higher pixel values represent the brighter portion of the image and the lower pixel values represent the darker portions. This is how we can detect a vertical edge in an image.

More Edge Detection

The type of filter that we choose helps to detect the vertical or horizontal edges. We can use the following filters to detect different edges:

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

Some of the commonly used filters are:

1	0	-1
2	0	-2
1	0	-1

Sobel
filter

3	0	-3
10	0	-10
3	0	-3

Scharr
filter

The Sobel filter puts a little bit more weight on the central pixels. Instead of using these filters, we can create our own as well and treat them as a parameter which the model will learn using backpropagation.

5.3 Strided Convolutions

Suppose we choose a stride of 2. So, while convoluting through the image, we will take two steps – both in the horizontal and vertical directions separately. The dimensions for stride s will be:

- **Input:** $n \times n$
- **Padding:** p
- **Stride:** s
- **Filter size:** $f \times f$
- **Output:** $[(n+2p-f)/s+1] \times [(n+2p-f)/s+1]$

Stride helps to reduce the size of the image, a particularly useful feature.

Convolutions Over Volume

Suppose, instead of a 2-D image, we have a 3-D input image of shape $6 \times 6 \times 3$. We will use a $3 \times 3 \times 3$ filter instead of a 3×3 filter. Let's look at an example:

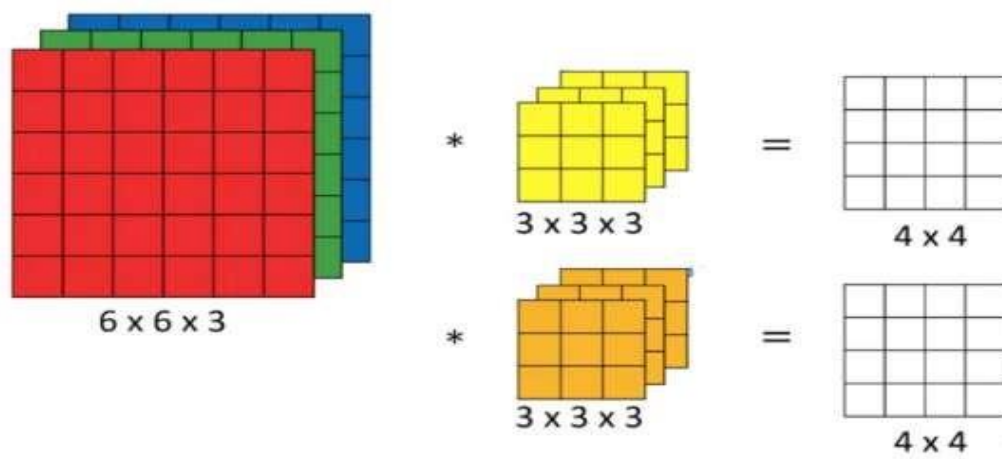
Input: $6 \times 6 \times 3$

Filter: $3 \times 3 \times 3$

The dimensions above represent the height, width and channels in the input and filter. Keep in mind that the number of channels in the input and filter should be same. This will result in an output of 4×4 . Let's understand it visually:

Since there are three channels in the input, the filter will consequently also have three channels. After convolution, the output shape is a 4×4 matrix. So, the first element of the output is the sum of the element-wise product of the first 27 values from the input (9 values from each channel) and the 27 values from the filter. After that we convolve over the entire image.

Instead of using just a single filter, we can use multiple filters as well. How do we do that? Let's say the first filter will detect vertical edges and the second filter will detect horizontal edges from the image. If we use multiple filters, the output dimension will change. So, instead of having a 4×4 output as in the above example, we would have a $4 \times 4 \times 2$ output (if we have used 2 filters):



Generalized dimensions can be given as:

- **Input:** $n \times n \times n_c$
- **Filter:** $f \times f \times n_c$
- **Padding:** p

- **Stride:** s
- **Output:** $[(n+2p-f)/s+1] \times [(n+2p-f)/s+1] \times n_c'$

Here, n_c is the number of channels in the input and filter, while n_c' is the number of filters.

5.4 One Layer of a Convolutional Network

Once we get an output after convolving over the entire image using a filter, we add a bias term to those outputs and finally apply an activation function to generate activations. This is one layer of a convolutional network. Recall that the equation for one forward pass is given by:

$$z^{[1]} = w^{[1]} * a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

In our case, input (6 X 6 X 3) is $a^{[0]}$ and filters (3 X 3 X 3) are the weights $w^{[1]}$. These activations from layer 1 act as the input for layer 2, and so on. Clearly, the number of parameters in case of convolutional neural networks is independent of the size of the image. It essentially depends on the filter size. Suppose we have 10 filters, each of shape 3 X 3 X 3. What will be the number of parameters in that layer? Let's try to solve this:

- Number of parameters for each filter = $3*3*3 = 27$
- There will be a bias term for each filter, so total parameters per filter = 28
- As there are 10 filters, the total parameters for that layer = $28*10 = 280$

No matter how big the image is, the parameters only depend on the filter size. Awesome, isn't it? Let's have a look at the summary of notations for a convolution layer:

- $f^{[l]}$ = filter size
- $p^{[l]}$ = padding
- $s^{[l]}$ = stride
- $n_{[c]}^{[l]}$ = number of filters

In a convolutional network (ConvNet), there are basically three types of layers:

1. Convolution layer
2. Pooling layer
3. Fully connected layer

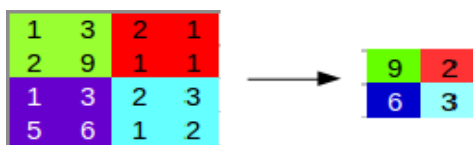
Let's understand the pooling layer in the next section.

Pooling Layers

Pooling layers are generally used to reduce the size of the inputs and hence speed up the computation. Consider a 4 X 4 matrix as shown below:

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

Applying max pooling on this matrix will result in a 2 X 2 output:



For every consecutive 2 X 2 block, we take the max number. Here, we have applied a filter of size 2 and a stride of 2. These are the hyperparameters for the pooling layer. Apart from max pooling, we can also apply average pooling where, instead of taking the max of the numbers, we take their average. In summary, the hyperparameters for a pooling layer are:

1. Filter size
2. Stride
3. Max or average pooling

If the input of the pooling layer is $n_h \times n_w \times n_c$, then the output will be $\{(n_h - f) / s + 1\} \times \{(n_w - f) / s + 1\} \times n_c$.

We'll take things up a notch now. Let's look at how a convolution neural network with convolutional and pooling layer works. Suppose we have an input of shape 32 X 32 X3 : There are a combination of convolution and pooling layers at the beginning, a few fully connected layers at the end and finally a softmax classifier to classify the input into various categories. There are a lot of hyperparameters in this network which we have to specify as well.

CHAPTER-6

RESULT AND CONCLUSION

6.1 For benign tumor detection

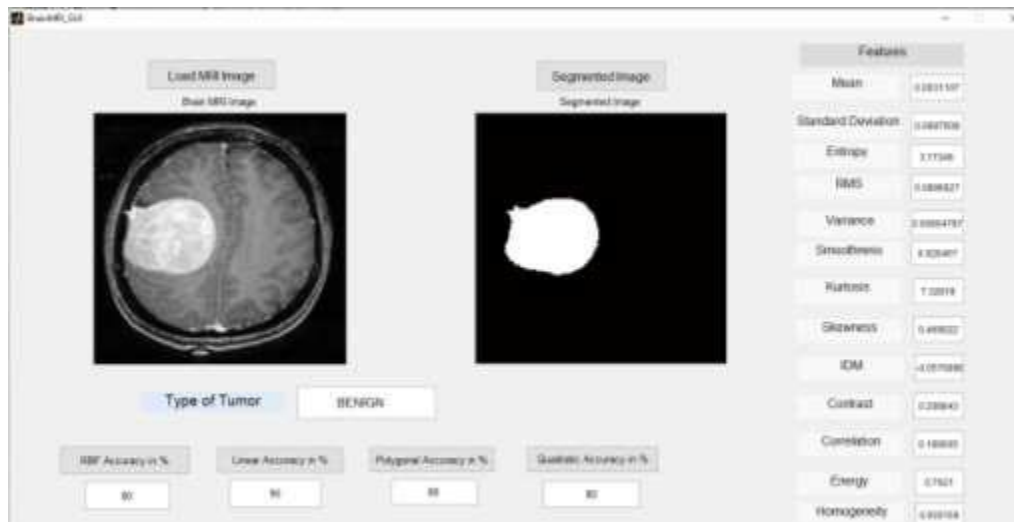


Fig 6.1 benign tumor

6.2 For malignant tumor

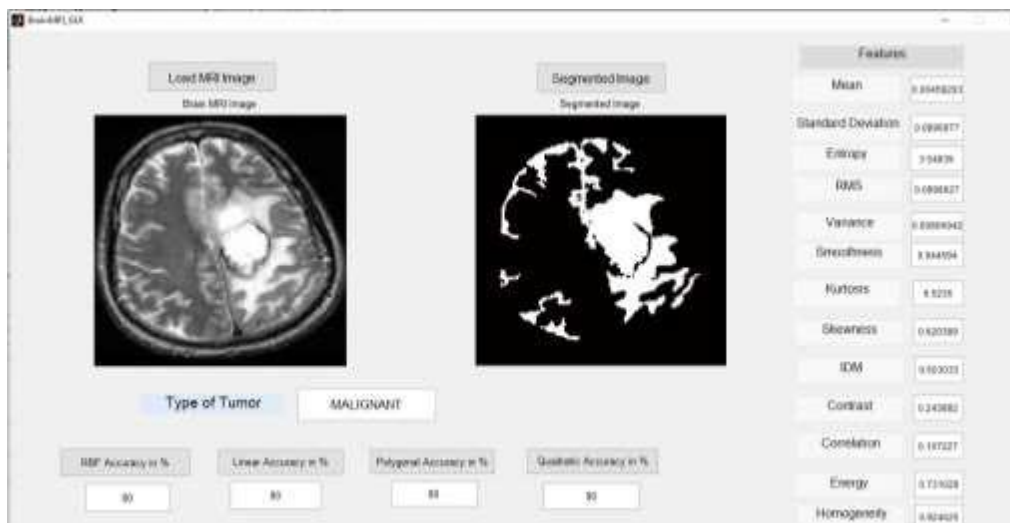


Fig 6.2 malignant tumor

CONCLUSION

- K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.
- K-means is a centroid-based clustering algorithm, where we calculate the distance between each data point and a centroid to assign it to a cluster. The goal is to identify the K number of groups in the dataset.
- Here, we divide a data space into K clusters and assign a mean value to each. The data points are placed in the clusters closest to the mean value of that cluster. There are several distance metrics available that can be used to calculate the distance.

CHAPTER-7

REFERENCES

1. Rahib L, Smith BD, Aizenberg R, Rosenzweig AB, Fleshman JM, Matrisian LM. Projecting cancer incidence and deaths to 2030: the unexpected burden of thyroid, liver, and pancreas cancers in the United States. *Cancer Res* 2014; 74: 2913–21.
2. Siegel RL, Miller KD, Jemal A. Cancer statistics, 2019. *CA Cancer J Clin* 2019; 69: 7–34.
3. Ryan DP, Hong TS, Bardeesy N. Brain adenocarcinoma. *N Engl J Med* 2014; 371: 1039–49.
4. Al-Hawary MM, Francis IR, Chari ST, et al. Brain ductal adenocarcinoma radiology reporting template: consensus statement of the Society of Abdominal Radiology and the American Brain Association. *Radiology* 2014; 270: 248–60.
5. Dewitt J, Devereaux BM, Lehman GA, Sherman S, Imperiale TF. Comparison of endoscopic ultrasound and computed tomography for the preoperative evaluation of brain cancer: a systematic review. *Clin Gastroenterol Hepatol* 2006; 4: 717–25; quiz 664.
6. Yasaka K, Abe O. Deep learning and artificial intelligence in radiology: current applications and future directions. *PLoS Med* 2018; 15: e1002707.
7. Esteva A, Kuprel B, Novoa RA, et al. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 2017; 542: 115–18.

APPENDIX

A. SOURCE CODE

```
function varargout = BrainMRI_GUI(varargin)

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;

gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @BrainMRI_GUI_OpeningFcn, ...
                  'gui_OutputFcn', @BrainMRI_GUI_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback', []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT
```

```

% --- Executes just before BrainMRI_GUI is made visible.

function BrainMRI_GUI_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject    handle to figure

% eventdata  reserved - to be defined in a future version of MATLAB

% handles     structure with handles and user data (see GUIDATA)

% varargin    command line arguments to BrainMRI_GUI (see VARARGIN)


% Choose default command line output for BrainMRI_GUI

handles.output = hObject;

ss = ones(200,200);

axes(handles.axes1);

imshow(ss);

axes(handles.axes2);

imshow(ss);

% Update handles structure

guidata(hObject, handles);


% UIWAIT makes BrainMRI_GUI wait for user response (see UIRESUME)

% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.

```

```
function varargout = BrainMRI_GUI_OutputFcn(hObject, eventdata, handles)
```

```
% varargout cell array for returning output args (see VARARGOUT);
```

```
% hObject handle to figure
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
```

```
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
```

```
function pushbutton1_Callback(hObject, eventdata, handles)
```

```
% hObject handle to pushbutton1 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
[FileName,PathName] = uigetfile('*.jpg;*.png;*.bmp','Pick an MRI Image');
```

```
if isequal(FileName,0)||isequal(PathName,0)
```

```
    warndlg('User Press Cancel');
```

```
else
```

```
    P = imread([PathName,FileName]);
```

```
    P = imresize(P,[200,200]);
```

```
    % input =imresize(a,[512 512]);
```

```
axes(handles.axes1)
```

```

imshow(P);title('Brain MRI Image');

% helpdlg(' Multispectral Image is Selected ');


% set(handles.edit1,'string',Filename);

% set(handles.edit2,'string',Pathname);

handles.ImgData = P;

% handles.FileName = FileName;


guidata(hObject,handles);

end


% --- Executes on button press in pushbutton2.

function pushbutton2_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if isfield(handles,'ImgData')

    %if isfield(handles,'imgData')

    I = handles.ImgData;


    gray = rgb2gray(I);

    % Otsu Binarization for segmentation

    level = graythresh(I);

    % gray = gray>80;

```

```

img = im2bw(I,6);

img = bwareaopen(img,80);

img2 = im2bw(I);

% Try morphological operations

% gray = rgb2gray(I);

% tumor = imopen(gray,strel('line',15,0));


axes(handles.axes2)

imshow(img);title('Segmented Image');

%imshow(tumor);title('Segmented Image');


handles ImgData2 = img2;

guidata(hObject,handles);


signal1 = img2(:,,:);

%Feat = getmswpfeat(signal,winsize,wininc,J,'matlab');

%Features = getmswpfeat(signal,winsize,wininc,J,'matlab');


[cA1,cH1,cV1,cD1] = dwt2(signal1,'db4');

[cA2,cH2,cV2,cD2] = dwt2(cA1,'db4');

[cA3,cH3,cV3,cD3] = dwt2(cA2,'db4');


DWT_feat = [cA3,cH3,cV3,cD3];

G = pca(DWT_feat);

```

```

whos DWT_feat

whos G

g = graycomatrix(G);

stats = graycoprops(g,'Contrast Correlation Energy Homogeneity');

Contrast = stats.Contrast;

Correlation = stats.Correlation;

Energy = stats.Energy;

Homogeneity = stats.Homogeneity;

Mean = mean2(G);

Standard_Deviation = std2(G);

Entropy = entropy(G);

RMS = mean2(rms(G));

%Skewness = skewness(img)

Variance = mean2(var(double(G)));

a = sum(double(G(:)));

Smoothness = 1-(1/(1+a));

Kurtosis = kurtosis(double(G(:)));

Skewness = skewness(double(G(:)));

% Inverse Difference Movement

m = size(G,1);

n = size(G,2);

in_diff = 0;

for i = 1:m

    for j = 1:n

```

```

        temp = G(i,j)./(1+(i-j).^2);

        in_diff = in_diff+temp;

    end

end

IDM = double(in_diff);

feat = [Contrast,Correlation,Energy,Homogeneity, Mean, Standard_Deviation, Entropy,
RMS, Variance, Smoothness, Kurtosis, Skewness, IDM];

load Trainset.mat

xdata = meas;

group = label;

svmStruct1 = svmtrain(xdata,group,'kernel_function', 'linear');

species = svmclassify(svmStruct1,feat,'showplot',false);

if strcmpi(species,'MALIGNANT')

    helpdlg(' Malignant Tumor ');

    disp(' Malignant Tumor ');

else

    helpdlg(' Benign Tumor ');

    disp(' Benign Tumor ');

end

set(handles.edit4,'string',species);

% Put the features in GUI

set(handles.edit5,'string',Mean);

```

```

set(handles.edit6,'string',Standard_Deviation);

set(handles.edit7,'string',Entropy);

set(handles.edit8,'string',RMS);

set(handles.edit9,'string',Variance);

set(handles.edit10,'string',Smoothness);

set(handles.edit11,'string',Kurtosis);

set(handles.edit12,'string',Skewness);

set(handles.edit13,'string',IDM);

set(handles.edit14,'string',Contrast);

set(handles.edit15,'string',Correlation);

set(handles.edit16,'string',Energy);

set(handles.edit17,'string',Homogeneity);

end

```

```

% --- Executes on button press in pushbutton3.

```

```

function pushbutton3_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to pushbutton3 (see GCBO)

```

```

% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles    structure with handles and user data (see GUIDATA)

```

```

load Trainset.mat

```

```

%data = [meas(:,1), meas(:,2)];

```

```

Accuracy_Percent= zeros(200,1);

```

```

itr = 80;

```



```

hWaitBar = waitbar(0,'Evaluating Maximum Accuracy with 100 iterations');

for i = 1:itr

data = meas;

%groups = ismember(label,'BENIGN ');

groups = ismember(label,'MALIGNANT');

[train,test] = crossvalind('HoldOut',groups);

cp = classperf(groups);

%svmStruct =
svmtrain(data(train,:),groups(train),'boxconstraint',Inf,'showplot',false,'kernel_function','rbf');

svmStruct_RBF =
svmtrain(data(train,:),groups(train),'boxconstraint',Inf,'showplot',false,'kernel_function','rbf');

classes2 = svmclassify(svmStruct_RBF,data(test,:), 'showplot',false);

classperf(cp,classes2,test);

%Accuracy_Classification_RBF = cp.CorrectRate.*100;

Accuracy_Percent(i) = cp.CorrectRate.*100;

sprintf('Accuracy of RBF Kernel is: %g%%',Accuracy_Percent(i))

waitbar(i/itr);

end

delete(hWaitBar);

Max_Accuracy = max(Accuracy_Percent);

sprintf('Accuracy of RBF kernel is: %g%%',Max_Accuracy)

set(handles.edit1,'string',Max_Accuracy);

guidata(hObject,handles);

```

```

% --- Executes on button press in pushbutton4.

function pushbutton4_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load Trainset.mat

%data = [meas(:,1), meas(:,2)];

Accuracy_Percent= zeros(200,1);

itr = 100;

hWaitBar = waitbar(0,'Evaluating Maximum Accuracy with 100 iterations');

for i = 1:itr

    data = meas;

    %groups = ismember(label,'BENIGN ');

    groups = ismember(label,'MALIGNANT');

    [train,test] = crossvalind('HoldOut',groups);

    cp = classperf(groups);

    svmStruct = svmtrain(data(train,:),groups(train),'showplot',false,'kernel_function','linear');

    classes = svmclassify(svmStruct,data(test,:), 'showplot',false);

    classperf(cp,classes,test);

    %Accuracy_Classification = cp.CorrectRate.*100;

    Accuracy_Percent(i) = cp.CorrectRate.*100;

    sprintf('Accuracy of Linear Kernel is: %g%%',Accuracy_Percent(i))

    waitbar(i/itr);

end

```

```

delete(hWaitBar);

Max_Accuracy = max(Accuracy_Percent);

sprintf('Accuracy of Linear kernel is: %g%%',Max_Accuracy)

set(handles.edit2,'string',Max_Accuracy);

% --- Executes on button press in pushbutton5.

function pushbutton5_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load Trainset.mat

%data = [meas(:,1), meas(:,2)];

Accuracy_Percent= zeros(200,1);

itr = 100;

hWaitBar = waitbar(0,'Evaluating Maximum Accuracy with 100 iterations');

for i = 1:itr

data = meas;

groups = ismember(label,'BENIGN ');

groups = ismember(label,'MALIGNANT');

[train,test] = crossvalind('HoldOut',groups);

cp = classperf(groups);

svmStruct_Poly =
svmtrain(data(train,:),groups(train),'Polyorder',2,'Kernel_Function','polynomial');

classes3 = svmclassify(svmStruct_Poly,data(test,:), 'showplot',false);

```

```

classperf(cp,classes3,test);

Accuracy_Percent(i) = cp.CorrectRate.*100;

sprintf('Accuracy of Polynomial Kernel is: %g%%',Accuracy_Percent(i))

waitbar(i/itr);

end

delete(hWaitBar);

Max_Accuracy = max(Accuracy_Percent);

%Accuracy_Classification_Poly = cp.CorrectRate.*100;

sprintf('Accuracy of Polynomial kernel is: %g%%',Max_Accuracy)

set(handles.edit3,'string',Max_Accuracy);

```

```

function edit1_Callback(hObject, eventdata, handles)

% hObject    handle to edit1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

```

```

% --- Executes during object creation, after setting all properties.

```

```

function edit1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit1 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

%    See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function edit2_Callback(hObject, eventdata, handles)

% hObject    handle to edit2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.

function edit2_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

```

```
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
%    See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function edit3_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit3 as text
```

```
%    str2double(get(hObject,'String')) returns contents of edit3 as a double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function edit3_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called
```

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

% --- Executes during object creation, after setting all properties.

```
function pushbutton4_CreateFcn(hObject, eventdata, handles)
```

% hObject handle to pushbutton4 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

```
function edit4_Callback(hObject, eventdata, handles)
```

% hObject handle to edit4 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text

% str2double(get(hObject,'String')) returns contents of edit4 as a double

% --- Executes during object creation, after setting all properties.

function edit4_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit4 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function edit5_Callback(hObject, eventdata, handles)

% hObject handle to edit5 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text

% str2double(get(hObject,'String')) returns contents of edit5 as a double


```

% --- Executes during object creation, after setting all properties.

function edit5_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit5 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

%    See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

```

```

function edit6_Callback(hObject, eventdata, handles)

% hObject    handle to edit6 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit6 as text

%    str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.

```

```

function edit6_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit6 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

%    See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

```

```

function edit7_Callback(hObject, eventdata, handles)

% hObject    handle to edit7 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit7 as text

%    str2double(get(hObject,'String')) returns contents of edit7 as a double


% --- Executes during object creation, after setting all properties.

function edit7_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to edit7 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

%    See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

```

```

function edit8_Callback(hObject, eventdata, handles)

% hObject    handle to edit8 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit8 as text
%    str2double(get(hObject,'String')) returns contents of edit8 as a double


% --- Executes during object creation, after setting all properties.

function edit8_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit8 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

%    See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function edit9_Callback(hObject, eventdata, handles)

% hObject    handle to edit9 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a double


% --- Executes during object creation, after setting all properties.

function edit9_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit9 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

```

```
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
%    See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function edit10_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit10 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit10 as text
```

```
%    str2double(get(hObject,'String')) returns contents of edit10 as a double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function edit10_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit10 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called
```

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function edit11_Callback(hObject, eventdata, handles)
```

% hObject handle to edit11 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text

% str2double(get(hObject,'String')) returns contents of edit11 as a double

% --- Executes during object creation, after setting all properties.

```
function edit11_CreateFcn(hObject, eventdata, handles)
```

% hObject handle to edit11 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

```
% Hint: edit controls usually have a white background on Windows.
```

```
%     See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function edit12_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit12 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit12 as text
```

```
%     str2double(get(hObject,'String')) returns contents of edit12 as a double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function edit12_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit12 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end
```

```
function edit13_Callback(hObject, eventdata, handles)

% hObject handle to edit13 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
% str2double(get(hObject,'String')) returns contents of edit13 as a double


% --- Executes during object creation, after setting all properties.

function edit13_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit13 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.
```



```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end

```

```

function edit14_Callback(hObject, eventdata, handles)

% hObject    handle to edit14 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%        str2double(get(hObject,'String')) returns contents of edit14 as a double


% --- Executes during object creation, after setting all properties.

function edit14_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit14 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

%        See ISPC and COMPUTER.

```

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))  
  
    set(hObject,'BackgroundColor','white');  
  
end
```

```
function edit15_Callback(hObject, eventdata, handles)  
  
% hObject    handle to edit15 (see GCBO)  
  
% eventdata reserved - to be defined in a future version of MATLAB  
  
% handles    structure with handles and user data (see GUIDATA)  
  
% Hints: get(hObject,'String') returns contents of edit15 as text  
%        str2double(get(hObject,'String')) returns contents of edit15 as a double  
  
% --- Executes during object creation, after setting all properties.  
  
function edit15_CreateFcn(hObject, eventdata, handles)  
  
% hObject    handle to edit15 (see GCBO)  
  
% eventdata reserved - to be defined in a future version of MATLAB  
  
% handles    empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
  
%        See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))  
  
    set(hObject,'BackgroundColor','white');  
  
end
```

```
function edit16_Callback(hObject, eventdata, handles)  
  
% hObject    handle to edit16 (see GCBO)  
  
% eventdata reserved - to be defined in a future version of MATLAB  
  
% handles    structure with handles and user data (see GUIDATA)  
  
% Hints: get(hObject,'String') returns contents of edit16 as text  
%        str2double(get(hObject,'String')) returns contents of edit16 as a double  
  
% --- Executes during object creation, after setting all properties.  
  
function edit16_CreateFcn(hObject, eventdata, handles)  
  
% hObject    handle to edit16 (see GCBO)  
  
% eventdata reserved - to be defined in a future version of MATLAB  
  
% handles    empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
  
%        See ISPC and COMPUTER.
```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');
end

```

```

function edit17_Callback(hObject, eventdata, handles)

% hObject    handle to edit17 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%        str2double(get(hObject,'String')) returns contents of edit17 as a double


% --- Executes during object creation, after setting all properties.

function edit17_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit17 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

%        See ISPC and COMPUTER.

```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

    set(hObject,'BackgroundColor','white');

end

```

```

% --- Executes on button press in pushbutton6.

```

```

function pushbutton6_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to pushbutton6 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB

```

```

% handles    structure with handles and user data (see GUIDATA)

```

```

load Trainset.mat

```

```

%data = [meas(:,1), meas(:,2)];

```

```

Accuracy_Percent= zeros(200,1);

```

```

itr = 100;

```

```

hWaitBar = waitbar(0,'Evaluating Maximum Accuracy with 100 iterations');

```

```

for i = 1:itr

```

```

    data = meas;

```

```

    groups = ismember(label,'BENIGN ');

```

```

    groups = ismember(label,'MALIGNANT');

```

```

    [train,test] = crossvalind('HoldOut',groups);

```

```

    cp = classperf(groups);

```

```

    svmStruct4 =

```

```

    svmtrain(data(train,:),groups(train),'showplot',false,'kernel_function','quadratic');

```

```

    classes4 = svmclassify(svmStruct4,data(test,:), 'showplot',false);

```

```

classperf(cp,classes4,test);

%Accuracy_Classification_Quad = cp.CorrectRate.*100;

Accuracy_Percent(i) = cp.CorrectRate.*100;

sprintf('Accuracy of Quadratic Kernel is: %g%%',Accuracy_Percent(i))

waitbar(i/itr);

end

delete(hWaitBar);

Max_Accuracy = max(Accuracy_Percent);

sprintf('Accuracy of Quadratic kernel is: %g%%',Max_Accuracy)

set(handles.edit19,'string',Max_Accuracy);

```

```

function edit19_Callback(hObject, eventdata, handles)

% hObject    handle to edit19 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a double

% --- Executes during object creation, after setting all properties.

function edit19_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit19 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

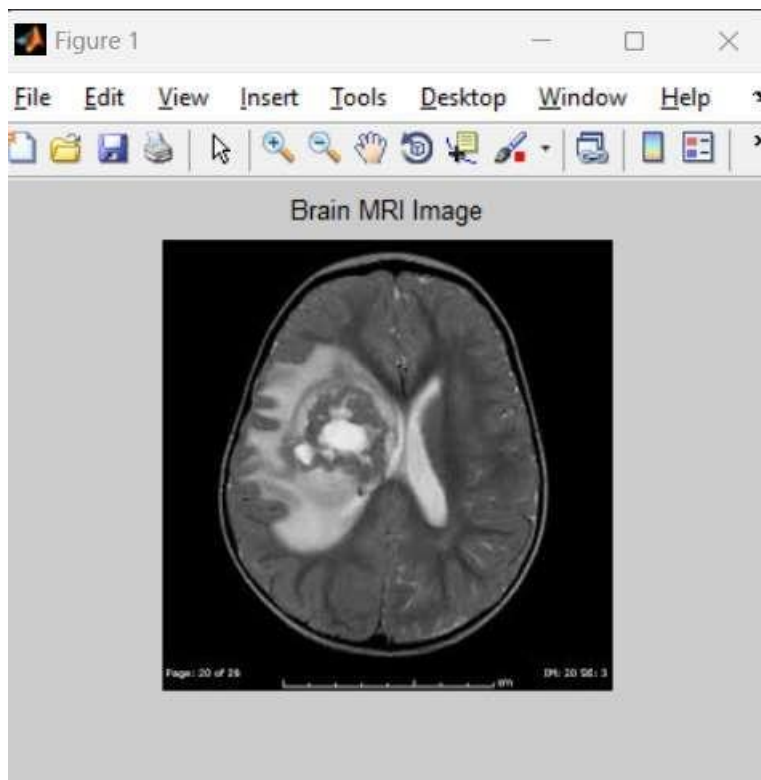

% Hint: edit controls usually have a white background on Windows.

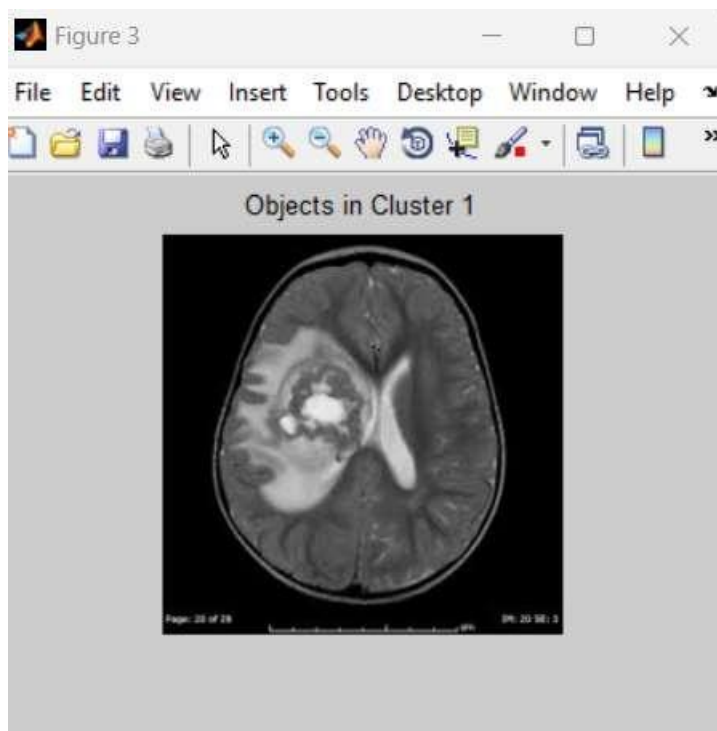
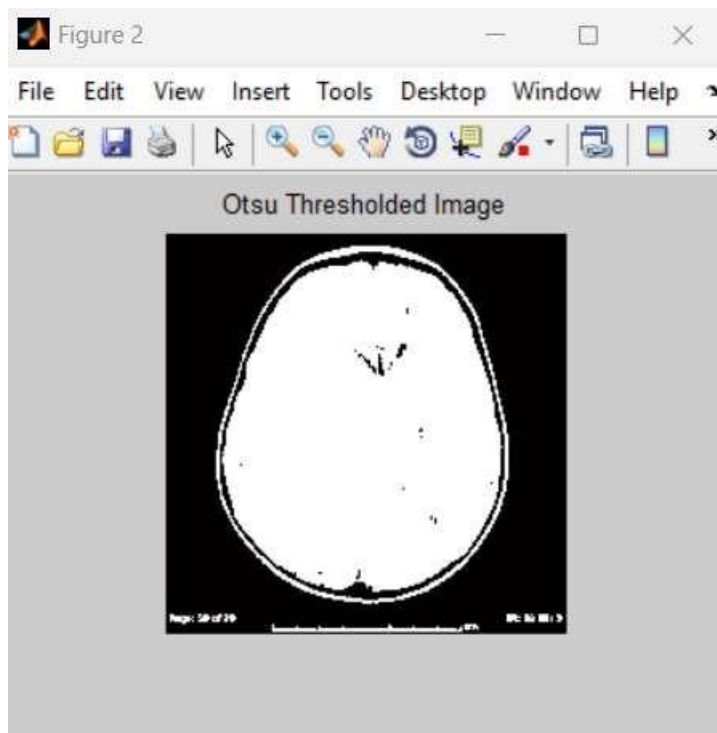
% See ISPC and COMPUTER.

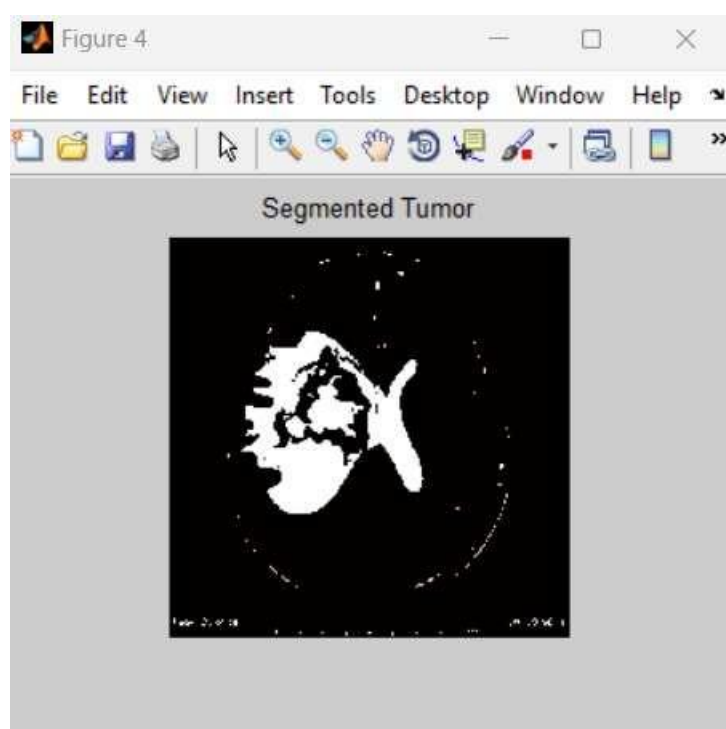
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

B. SCREENSHOTS







C. RESEARCH PAPER

Automated Detection and Analysis of Brain Tumor Using Image Processing from MRI Images

Sandeep Srinivas Batta

Computer Science and
Engineering

Sathyabama Institute of Science
and Technology

Chennai, Tamil Nadu

Rongali Sasi Kumar

Computer Science and
Engineering

Sathyabama Institute of Science
and Technology

Chennai, Tamil Nadu

V. Dharani

Sathyabama Institute of Science
and Technology

ABSTRACT:

A brain tumor is a type of abnormal development caused by unregulated cell reproduction. Magnetic resonance imaging (MRI) is the most often utilized diagnostic tool. The amount of data in MR images is too large for human interpretation and analysis. Brain tumor segmentation in Magnetic Resonance Imaging (MRI) is one of the major principal studies that has emerged as an investigation in the field of medical imaging systems. One of the major traits in tumor diagnosis is the precise recognition of the area and whereabouts of a brain tumor. Image processing in medical field is a highly competing in discipline. Image segmentation is important in image processing because it aids in the extraction of questionable areas from medical pictures. An effective technique for tumor identification based on segmentation of brain MRI images utilizing K-Means grouping the tumorous region in the image is suggested in this study. In this research, we will discuss a tumor picture segmentation approach that is more accurate and reproducible than traditional manual segmentation. This approach further decreases image analysis time by extracting the tumor from the MR image and determining its exact form and location. Based on the area calculated from the cluster, we can also identify the tumor stage and present it. If such a system is established and employed, it will reduce the strain on radiologists and clinicians working in this sector while also improving the identification and diagnosis of brain tumors.

Keywords: K-means, clustering, segmentation, Magnetic Resonance Imaging

INTRODUCTION

A brain tumor, also known as an intracranial tumor, is an uncontrollable growth of tissue in which cells proliferate and reproduce rapidly, appearing unfettered by normal cell regulatory processes. Brain tumors are of two types one is malignant (cancerous) or benign (noncancerous), which can afflict both children and adults. Brain tumors, whether malignant or not, can impair brain function if they get out of hand that is if they multiply fast enough to impinge on adjacent tissues. Brain tumors can be treated in a variety of ways. All forms of brain tumors can cause symptoms that vary based on the tumor's size and location in the brain. Headaches, seizures, eye difficulties, vomiting, and mental abnormalities are all possible symptoms. Brain tumors can harm both children and adults. One major reason for this is that there aren't any effective early diagnosis methods for brain tumors. Brain tumor can be detected by some of the common symptoms and could be lead to a variety of other abdominal conditions. In addition, if the tumor has spread to other organs, treatment becomes extremely difficult. For helping radiologists for detecting the tumor in early stage an safe and reliable method need to be developed, so that even a patient where the expert or doctor cannot reach or help the patient can detect the tumor and can make appropriate care about the tumor. There hasn't been a lot of research done on brain tumor detection. According to the literature review, brain tumor identification occurs by using disease symptoms and taking a patient history, rather than using image processing.

This project makes an attempt to detect brain tumors from images obtained from CT scans. These photos are which was before using image analysis and deep learning methods, before a clustering is used to identify the tumor region in the image and determine whether it is normal or has a tumor in the pancreas.

LITERATURE REVIEW

Literature Review is one of the most important step in software development. Prior to building the instrument, the time factor, economics, and corporate strength must be determined. So now we have to choose which OS and languages, tools, hardware to construct the project. By moving further in the process of the project a lot of help is required from senior programmers, professors, assistant etc who will help to cover a great deal of work.

These rules are taken into account when creating the suggested system. The development sector thoroughly surveys the requirements of the project. One of the most important is the literature review. Before constructing the tools and related design, it is vital to assess and study the time element, resource need, manpower, economy, and corporate strength. Once these requirements have been met and thoroughly assessed, when the requirements are met and processed the next step is to identify the software requirements of the system such as which software we are using to build the project

According to the literature review, brain tumour identification is done utilising illness symptoms and patient history rather than image processing. Hence, while we can't foresee the disease in its early stages, it will get more difficult as it progresses. The sickness is difficult to detect. It did not employ any methods of image processing to predict illness early on.

Detection and analysis of brain tumor from MRI by Integrated Thresholding and Morphological Process with Histogram based method by the authors: Md. Rezwanul Islam, Md.Reezbhan Imteaz, MariumE-Jannat

Here we can discuss about brain tumor detection and analysis the type of tumor in an precise manner.We used image based processing techniques to improve the accuracy of the tumor.

Image Mining Methodology for Detection of Brain Tumor: A Review by the authors Mrs. Shinde Apurva Swapnil, Ms. Vengurlekar Samidha Girish

Here they used filtering , edge detection and clustering algorithms to detect and classify the type of tumor

This study attempts to identify brain tumours using CT scan pictures. These photos are pre-processed using algorithms for image processing before being classified using the K-means model architecture. The dataset was obtained from the medical image achievement repository. These brain tumour CT scan pictures are sent into the system as inputs. These photographs are in multiple formats, so we converted them to jpg, and the size of the image is too large, so we just utilised a handful of them. The K-means model architecture is then used for categorization. In this case, the system is taught to learn based on the special way in the picture.

EXISTING SYSTEM

Several strategies for dealing with noisy label categorization in natural photos have been presented. Suggest a method for assigning weights to training images using an extra clean validation set. Their concept is to assign smaller values to noisy examples and enhance the values of clean training examples to enhance the gradient update. There has been little research into the development and implementation of noisy labeling classification techniques in data from medical imaging. Dani et al. use label noise as part of a deep learning network to recover genuine labels from noisy data for the task of categorizing breast microcalcifications in multi-view mammograms.

There hasn't been much research on brain tumor detection. According to the literature review, brain tumor identification is done utilizing illness symptoms and patient history rather than image processing. Consequently, we can't foresee the disease in its early stages, and it will grow more difficult as it progresses. Identifying the condition is very sluggish. It did not employ any methods of image processing to predict illness early on.

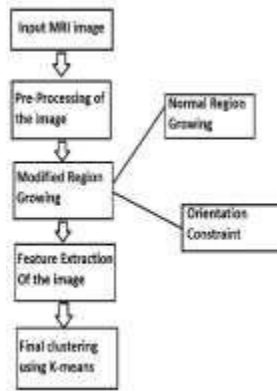
Even after thresholding, the resulting picture is not clear enough to establish a tumor diagnosis since we can't extract tumor information from the image, showing to be a significant disadvantage of existing threshold-based systems. To address some of these limitations in existing methods, we propose an image segmentation approach that will assist doctors in improved tumor identification and diagnosis.

PROPOSED SYSTEM

The goal of this study is to detect brain tumors using CT scan pictures. These photos are pre-processed with image processing techniques before being classified using the K-means model architecture. The dataset was obtained from the tumor screening success repository. The machine is fed these CT scan photos of brain tumors. These photographs are in a various format, so we converted them to jpg. The size of the image is also too large, so we just utilized a handful of them. The K-means planning and process is then used to do categorization. The system is created to learn in accordance with the special way in the picture in this case.

Using contrast based on image of patients, we created a K-means algorithm to identify cancer from healthy pancreas. In independent test sets, K-means demonstrated high accuracy and enhanced sensitivity compared to radiologist interpretation, with adequate accuracy in a test set gathered from patients of varied kinds. Our findings give the first strong evidence to suggest that K-means may capture the cryptic CT characteristics of brain cancer and help radiologists detect and diagnose brain cancer.

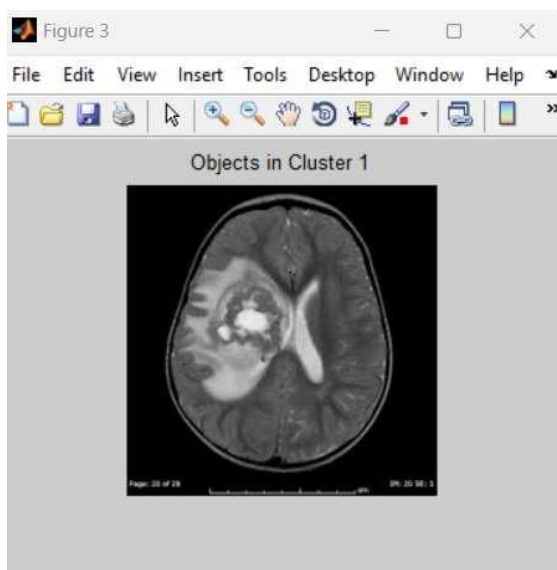
ARCHITECTURE OF THE SYSTEM



ALGORITHM

K-means clustering is a conventional, simple machine learning approach that is learned on a test data set after that then it is able to divide the data set to different sets like a fresh data set using an a priori which is determined prime, k number of clusters. Machine learning aims to develop algorithms for unsupervised learning on data. Many applications require a data analysis tool to sort the raw data so this becomes handy for many applications, like finding out the tumor cells in our project. We partition a data space into K clusters and provide each a mean value. The data points are allocated to the clusters with the highest mean value.

K-means clustering is a method of calculating the centroid of the whole dataset and then repeat the whole process of calculating the centroid for every data in the dataset and then determining the optimal centroid. It is arrogant to assume that the number of clusters is already known. The letter 'K' in K-means represents the total number of clusters discovered from data by the approach.



1. Image processing

Image Processing is a technique of improving the original images or raw images that are taken from cameras/CC TV on satellites, traffic lights, security office etc. During the recent times image processing has taken a great leap forward in improving and developing new techniques. So these techniques are mainly useful to magnify or strengthen the image according to our own use or output which are taken by space satellites, aircrafts, military jets. Because of the ready availability of strong people, image analysis systems are growing in popularity. Image Processing is utilized in a variety of techniques, including computers, large-scale graphics software, and so on.

2. Pre- processing

Pre- Processing tells us how the actions at abstraction stage involves in processing the image, here the input and output are intensity images. We can use this pre-processing to improve the image and extract the essential information and also, we can remove the noise present in the image

In machine learning, Data Preprocessing is taking the raw data and convert it into useful information for model creation. This step should be done at the start of the machine learning model process

Because excellent data is without a doubt an important thing to acquire. It is more significance than the model and quality of the data should also be checked. As a result businesses and people spend their time cleaning and getting ready the data. The raw data that is available in the actual world is not great as it is noisy, unreliable and incomplete to use. It also might not have precise, pertinent properties and it might also be false. Preprocessing is crucial to enhancing the caliber of the data. By removing any duplicates or anomalies, standardizing the data so that it can be compared, and enhancing the quality of the findings, preprocessing aids in ensuring that the data is consistent.

3. Feature Extraction

The matrix factorization method includes the feature extraction stage, which splits and condenses an initial collection of raw data into smaller, easier-to-manage categories. These qualities properly and uniquely describe the actual data gathering while being straightforward to manage.

Every data set has a subset of data, this subset selection is the difficult part for better algorithm result. A learning algorithm's performance (both accuracy and computation time) may be greatly improved by feature extraction, but it is not simple. Problems can be worked on using extremely high-dimensional feature-spaces.

A group of pixels that are having same spectral, spatial, and/or textural characteristics are referred to as an object (also known as a segment) in the object-based approach used by Feature Extraction to identify images. The spectral information included in each pixel is used to categories images using traditional pixel-based classification techniques.

4. Edge Detection

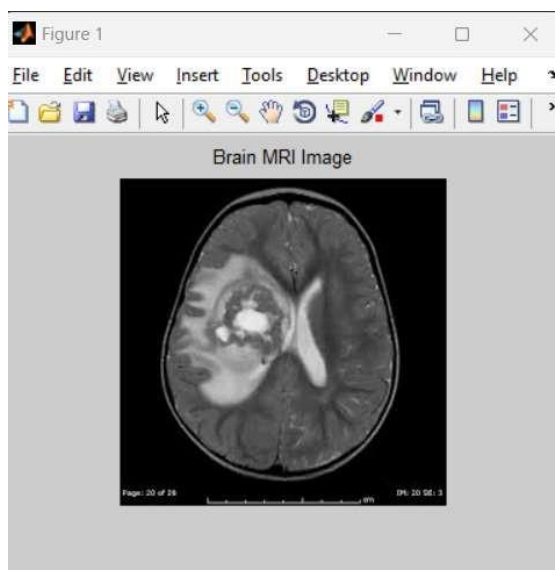
Identification of edges in an image via edge detection is a method that is essential to understanding picture features. It is believed that edges contain significant information and relevant characteristics. The image should be reduced to less size so that the analysis can be done and removing unnecessary information will be easy. Then the crucial structural components of an image that are important to the problem will be retained and focused.

The method relies on identifying bridges and significant changes in the picture. Points where colors and light shift are described here as bridges and sudden transitions. Sharp lines in the processed image may be seen when the color and lighting shift. Edge segmentation works by finding the edges in an image by various factors like grey level, brightness, saturation etc.

RESULTS AND DISCUSSIONS

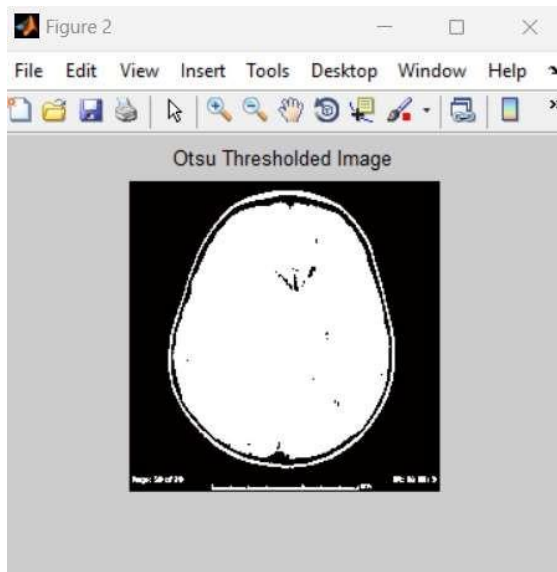
RESULT

The experiment was carried out using the Matlab application version 2018b and a machine with 8GB of Memory and a Core i7 1.8 GHz Processor. When we train the network, we save it so that we don't have to repeat the training process and waste time, after that we test the dataset and show the accuracy of the components of the system, in the last stage the brain tumor that observed and regarded, a sample image was selected, here we have opted an example image that filled into the system the network that was programmed previously, when clicking on the detect button, the brain tumor can be seen.

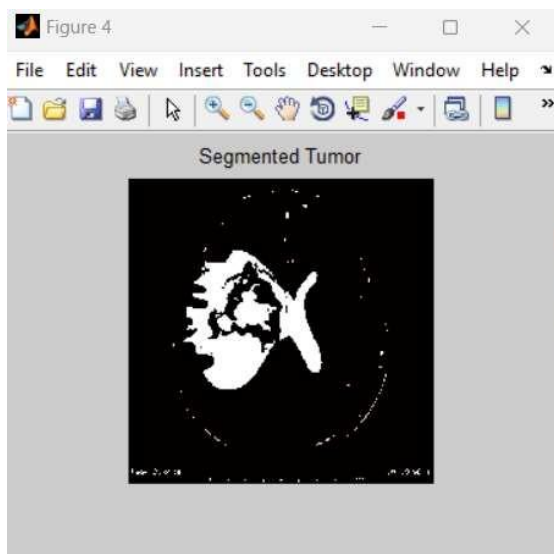


Accuracy = number of correctly classified images / total number of images \times 100%

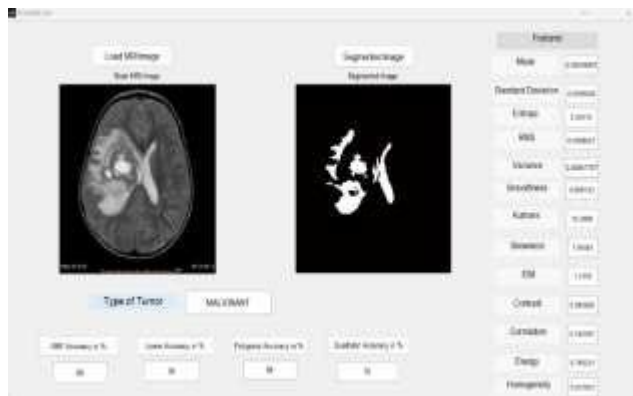
The user must extract tumor information from the picture after it has been shown. This option displays the numerous clusters that result from K-means clustering. Once this is completed, the tumor stage may be estimated.



The difference in accuracy is attributable to the collection and these photographs, as well as differences in image parameters such as brightness, brightness, quality, and so on. An essential point to notice is the procedure of partitioning the information, which was used in our work.



This is the final stage in the tumor detection procedure. After calculating the area, hit this button on the GUI to reveal the brain tumor stage. There are normally three conventional tumor stages, which can be shown based on the computed area.



CONCLUSION

In the field of medicine, like treating any brain problems or conducting any surgery will require the need of image segmentation of the brain images. Here we utilized it and performed K-means clustering technique and developed a system that aids in segmenting the MRI images to help the doctors. We used image processing, edge detection and feature extraction techniques to extract a great amount of data from the image and find the tumor part of the brain.

We hope this will help many people without need for any expert to find out whether they have tumor or not and take necessary step to take care of themselves

REFERENCES

Rahib L, Smith BD, Aizenberg R, Rosenzweig AB, Fleshman JM, Matrisian LM. Projecting cancer incidence and deaths to 2030: the unexpected burden of thyroid, liver, and pancreas cancers in the United States. *Cancer Res* 2014; 74: 2913–21.

- Siegel RL, Miller KD, Jemal A. Cancer statistics, 2019. *CA Cancer J Clin* 2019; 69: 7–34.
- Ryan DP, Hong TS, Bardeesy N. Brain adenocarcinoma. *N Engl J Med* 2014; 371: 1039–49.
- Al-Hawary MM, Francis IR, Chari ST, et al. Brain ductal adenocarcinoma radiology reporting template: consensus statement of the Society of Abdominal Radiology and the American Brain Association. *Radiology* 2014; 270: 248–60.
- Dewitt J, Devereaux BM, Lehman GA, Sherman S, Imperiale TF. Comparison of endoscopic ultrasound and computed tomography for the preoperative evaluation of brain cancer: a systematic review. *Clin Gastroenterol Hepatol* 2006; 4: 717–25; quiz 664.