

A CORONA RECOGNITION METHOD USING VISIBLE LIGHT COLOR AND MACHINE LEARNING

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

By

NAGIREDDY DIVYA SREE(37110482)

**MOPURI HARIKA KRISHNAVENI
(37110634)**



**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

JEPPIAAR NAGAR, RAJIV GANDHI

SALAI, CHENNAI – 600 119

MARCH 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with “A” grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bona fide work of **MOPURI HARIKA KRISHNAVENI (37110634)** and **NAGIREDDY DIVYASREE (37110482)** who carried out the project entitled “**A CORONA RECOGNITION METHOD BASED ON VISIBLE LIGHT COLOR AND MACHINE LEARNING**” under my supervision from **NOVEMBER 2020** to **MARCH 2021**.

Internal Guide

Dr. R.AROUL CANESSANE ,M.E., Ph.D

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **NAGIREDDY DIVYASREE and MOPURI HARIKA KRISHNAVENI**
hereby

declare that the Project Report entitled "**A CORONA RECOGNITION
METHOD BASED ON VISIBLE LIGHT COLOR AND MACHINE LEARNING**" is
done by me

under the guidance of **Dr. R. AROUL CANESSANE, M.E., Ph. D** Department
of Computer Science and Engineering at Sathyabama Institute of
Science and Technology is submitted in partial fulfillment of the
requirements for the award of Bachelor of Engineering degree in
Computer Science and Engineering.

DATE:

PLACE: CHENNAI

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. S. Vigneswari, M.E., Ph.D., and 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 **Dr. R. Aroul Canessane, M.E., Ph.D Assistant Professor**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my 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

COVID-19 is a worldwide epidemic, as announced by the world Health Organization in March 2020. Machine Learning methods can play vital roles in identifying COVID-19 patients by visually analyzing their chest x-ray images. In this a paper a new ML- method proposed to classify the chest x-ray images into two classes, COVID-19 patient or non-COVID-19 person. This is a highly infectious disease and spreads rapidly in the community worldwide. In this pandemic situation where the novel corona virus continuously spreading day by day worldwide, the researchers are focused on finding new ways to destroy the coronavirus.

The air borne transmission appears to plays an important role in the spread of coronavirus, UV light would be able to destroy the novel coronavirus when exposure to UVC irradiation stronger than $90\text{W}/\text{cm}^2$. Devices using UV light, emits high-intensity rays could help to kill the viruses and protect from disinfection areas contaminated from the novel corona virus. We will also highlight the basic study on coronavirus and how it was outbreak worldwide in the year 2020.

TABLE OF CONTENTS

ABSTRACT LIST OF FIGURES

v
viii

CHAPTER NO	TITLE	PAGENO
1	INTRODUCTION	1
	1.1 INTRODUCTION	2
2	LITERATURE SURVEY	3
	2.1 LITERATURE SURVEY	3
	<i>2.1.1 Survey of COVID-19 detection using medical images</i>	3
	<i>2.1.2 Corona virus classification using CT images by Machine Learning Methods</i>	4
	<i>2.1.3 Detection of COVID-19 using CXR and CT images</i>	4
	<i>2.1.4 Automatic Detection of corona virus Disease using X-ray images and deep convolutional Neural Networks</i>	4
3	SYSTEM DESIGN & METHODOLOGY	5
	3.1 EXISTING SYSTEM	5
	3.2 PROPOSED SYSTEM	6
	3.3 IMAGE PROCESSING	7
	3.3.1 Purpose of image processing	8
	3.4 TYPES OF IMAGE PROCESSING	9
	3.5 IMAGE PROCESSING CONCEPTS	10
	3.5.1 Binary images	11
	3.5.2 color images	12
	3.5.3 8-Bit color images	14

	3.5.4 24-Bit color images	16
	3.5.5 Color Quantization	17
	3.5.6 Convolution	18
	3.6 DISTANCE METRICS	19
	3.6.1 Euclidean distance	19
	3.6.2 City block distance	19
	3.6.3 Chessboard distance	19
	3.6.4 Dithering	19
	3.7 EDGE DETECTORS	20
	3.7.1 1 ST and 2 ND derivative of an edge illustrated in one dimension	20
	3.8 FREQUENCY DOMAIN	21
	3.9 GRAYSCALE IMAGES	21
4	SOFTWARE DEVELOPMENT METHODOLOGY	22
	4.1 IMAGE EDITING SOFTWARE	22
	4.2 KERNEL	23
	4.3 MASKING	25
	4.4 MATHEMATICAL MORPHOLOGY	27
	4.5 MULTI-SPECTRAL IMAGES	28
	4.6 PIXELS	29
	4.6.1-pixel connectivity	30
	4.7 PIXEL VALUES	31
	4.7.1 RGB and color spaces	32
	4.8 WRAPPING AND SATURATION	33
	4.9 SOFTWARE DESCRIPTION	34
5	RESULTS AND DISCUSSIONS	39
6	CONCLUSION AND FUTURE WORK	40
	REFERENCES	41
	APPENDIX	
	A. SOURCE CODE	54

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
3.2.1	BLOCK DIAGRAM	6
3.4.1	IMAGE PROCESSING BLOCK DIAGRAM	8
3.5.6.1	PIXEL AREA DISPLAY ONE COMPOSITE COLOR	13
3.5.7.1	CONVOLUTION DIAGRAM	14
3.7.1	1 ST AND 2 ND DERIVATIVE OF AN EDGE ILLUSTRATED IN ONE DIMENSION	18
4.2.1	CONVOLUTION AND KERNEL FOR A MEAN FILTER WITH 3*3 NEIGHBORHOOD	23
4.4.1	SIMPLE GRAY LEVEL IMAGE	26
4.6.1.1	TWO CONNECTED COMPONENTS BASED ON 4- CONNECTIVITY	28
4.8.1	MAPPING FUNCTION FOR WRAPPING THE PIXEL VALUES	30
4.8.2	MAPPING FUNCTION FOR SATURATING AN 8-BIT IMAGE	31
5.1	HISTOGRAM	35
5.2	EQUALIZATION	36
5.3	SEGMENTATION PROCESS	39
5.4	FILTRATION PROCESS	40

DILATION PROCESS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION:

A cluster of mysterious cases of pneumonia were detected in December 2019 in the province of Wuhan, China, which eventually spread to the rest of the world [2 - 5]. Initially, a few cases were reported in the European Union in countries such as France and Germany, which later escalated at an alarming rate. The outbreak has spread to several cruise ships and cruise operators have begun either to cancel or to change their routes as countries around the world have introduced travel restrictions to control the spread of disease [1]. As of 14 April 2020, 128,000 people died of COVID-19, while 1.99 million cases in 210 countries and territories were reported in 219,747 cases. Figure 1 represents the statistics about the confirmed cases across the globe. Figure 2 shows the statistics about the deaths across the globe. COVID 19 is an infectious illness triggered by the recently identified coronavirus virus. It was not understood until the outbreak in Wuhan, China, started in December 2019 [6]. The most frequent signs of COVID-19 include fever, tiredness, and dry cough. Most individuals (about 80 percent) are healing from the disease without needing extra treatment. Approximately 1 of every 6 people that have COVID-19 are seriously ill and experience trouble breathing. Senior citizens and those with chronic medical issues such as elevated blood pressure, cardiac disorders and diabetes are more prone to experience severe illnesses [7]. Evidence to date has demonstrated that the virus that triggers COVID-19 is mainly spread by interaction with respiratory droplets rather than through the air. Pneumonia is a type of acute respiratory infection that affects the lungs [8 – 10]. The lungs are made up of little pockets called alveoli, which are packed with oxygen as a stable person breathes. When anyone has pneumonia, these alveoli are packed with pus and blood, rendering ventilation painful and therefore raising the absorption of oxygen.

Symptoms of pneumonia can involve fever, trouble breathing, and tiredness. Pneumonia can spread in a variety of ways [11, 12]. Fig. 1. Image representing the number of cases of COVID-

19 across the world [31]. 2020 International Conference for Emerging Technology

Authorized licensed use limited to: Carleton University. Downloaded on August 06,2020 at 15:06:43 UTC from IEEE Xplore. Restrictions apply. Fig. 2. Image representing the number of deaths due to COVID-19 across the world [31]. The respiratory system is the part of the human body most affected by the virus, so the use of X-rays of the chest may prove to be a more efficient way than the thermal screening of the human body. Image Recognition tasks have progressed in recent years due to advances in algorithms, the availability of large datasets and powerful GPUs that allow deep architecture training. Convolution Neural Networks (CNNs), Residual Networks and training methods such as transfer learning have produced state-of-the-art performance for tasks such as image recognition and classification. As of April 4, 2020, under-developed antibody tests were adopted. As of 15 April 2020, the screening protocol adopted for mass control is focused on thermal imaging and wide scale testing using nRT-PCR test kits to identify potential patients with COVID-19 virus. The standard testing procedure may generate results within a few hours to two days. There is therefore a significant need for advanced technologies to screen future COVID-19 patients. In section 5, we applied an approach to classifying COVID19 positive patients based on X-Ray Images. The symptoms of pneumonia and COVID-19 are somewhat similar and the respiratory system is the most affected area of the human body, so screening based on the respiratory system using XRay would be a safer way.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY:

Survey of COVID-19 Detection Using Medical Images.

The outbreak of the COVID-19 pandemic caused the death of a large number of people. Millions of people are infected by this virus and are still getting infected day by day. As the cost and required time of conventional RT-PCR tests to detect COVID-19, researchers are trying to use medical images like X-Ray and Computed Tomography (CT) images to detect it with the help of Artificial Intelligence (AI) based systems. In this paper, we reviewed some of these newly emerging AI-based models that can detect COVID-19 from medical images using X-Ray or CT of lung images.

Coronavirus Classification using CT Images by Machine Learning Methods

This study presents early phase detection of Coronavirus (COVID-19), which is named by World Health Organization (WHO), by machine learning methods. The detection process was implemented on abdominal Computed Tomography (CT) images. The expert radiologists detected from CT images that COVID-19 shows different behaviours from other viral pneumonia. Therefore, the clinical experts specify that COVID-19 virus needs to be diagnosed in early phase. For detection of the COVID-19, four different datasets were formed by taking patches sized as 16x16, 32x32, 48x48, 64x64 from 150 CT images. The feature extraction process was applied to patches to increase the classification performance.

Detection of COVID-19 using CXR and CT images using Transfer Learning and Haralick features

Recognition of COVID-19 is a challenging task which consistently requires taking a gander at clinical images of patients. In this paper, the transfer

learning technique has been applied to clinical images of different types of pulmonary diseases, including COVID-19. It is found that COVID-19 is very much similar to pneumonia lung disease. Further findings are made to identify the type of pneumonia similar to COVID-19. Transfer Learning makes it possible for us to find out that viral pneumonia is same as COVID-19.

Automatic Detection of Coronavirus Disease (COVID-19) Using X-ray Images and Deep Convolutional Neural Networks

The 2019 novel coronavirus disease (COVID-19), with a starting point in China, has spread rapidly among people living in other countries, and is approaching approximately 34,986,502 cases worldwide according to the statistics of European Centre for Disease Prevention and Control. There are a limited number of COVID-19 test kits available in hospitals due to the increasing cases daily. Therefore, it is necessary to implement an automatic detection system as a quick alternative diagnosis option to prevent COVID-19 spreading among people.

CHAPTER 3

SYSTEM DESIGN & METHODOLOGY

EXISTING SYSTEM

In existing system saliva of the covid-19 patient is directly taken for testing. And it takes so much of time to get the result. The system doesn't give the accurate values or results i.e they are showing positive even for the negative patients. Corona discharges observed on the corona camera screen at a test voltage of 90 kV. Yellow marks signify the corona pulses, which were observed on the corona camera screen. The amount of the yellow marks indicates the intensity of corona discharge, so does the value of the corona current. Note that the discharge level at a test voltage of 90 kV is different for each conductor configuration

PROPOSED SYSTEM:

Image moments defined as projections of image functions onto a polynomial basis where the image moments used to extract global and local features from these images. Generally, projection of digital images using orthogonal polynomials with fractional orders results in orthogonal moments of fractional orders which are able to extract both coarse and fine features from the input digital images. In this paper, new orthogonal Exponent moments of fractional-orders are derived. Then, these moments are utilized to extract high accurate 961 features from each COVID-19 input image. The intrinsic properties of the new image moments are:

- These orthogonal moments are successfully able to represent digital images for low and high orders
- The orthogonal moments are invariants to geometric transformations,

which is an essential property for classification and recognition applications.

- The orthogonal moments are robust to noise.

Fast and inexpensive computation requirements make them favorable for real-time applications.

Block diagram:

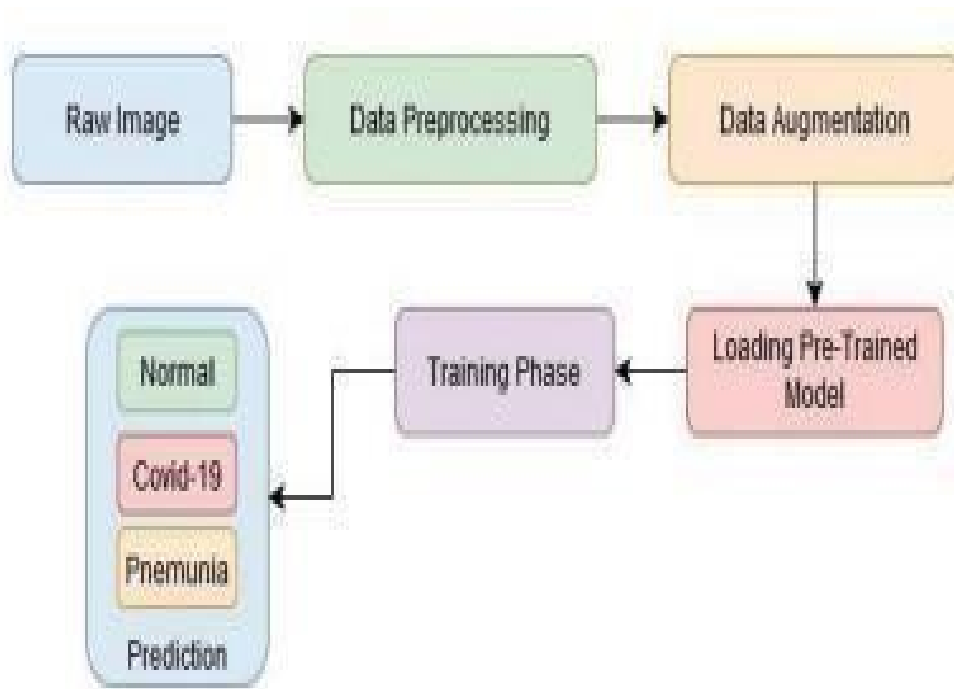


Fig 3.2.1:BLOCK DIAGRAM

IMAGE PROCESSING

Image Processing is a technique to enhance raw images received from

cameras/sensors placed on satellites, space probes and aircrafts or pictures taken in normal day-to-day life for various applications. Various techniques have been developed in Image Processing during the last four to five decades. Most of the techniques are developed for enhancing images obtained from unmanned spacecraft's, space probes and military reconnaissance flights. Image Processing systems are becoming popular due to easy availability of powerful personnel. Computers, large size graphics software's etc. Image Processing is used in various Techniques. *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:

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

TYPES OF IMAGE PROCESSING

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.

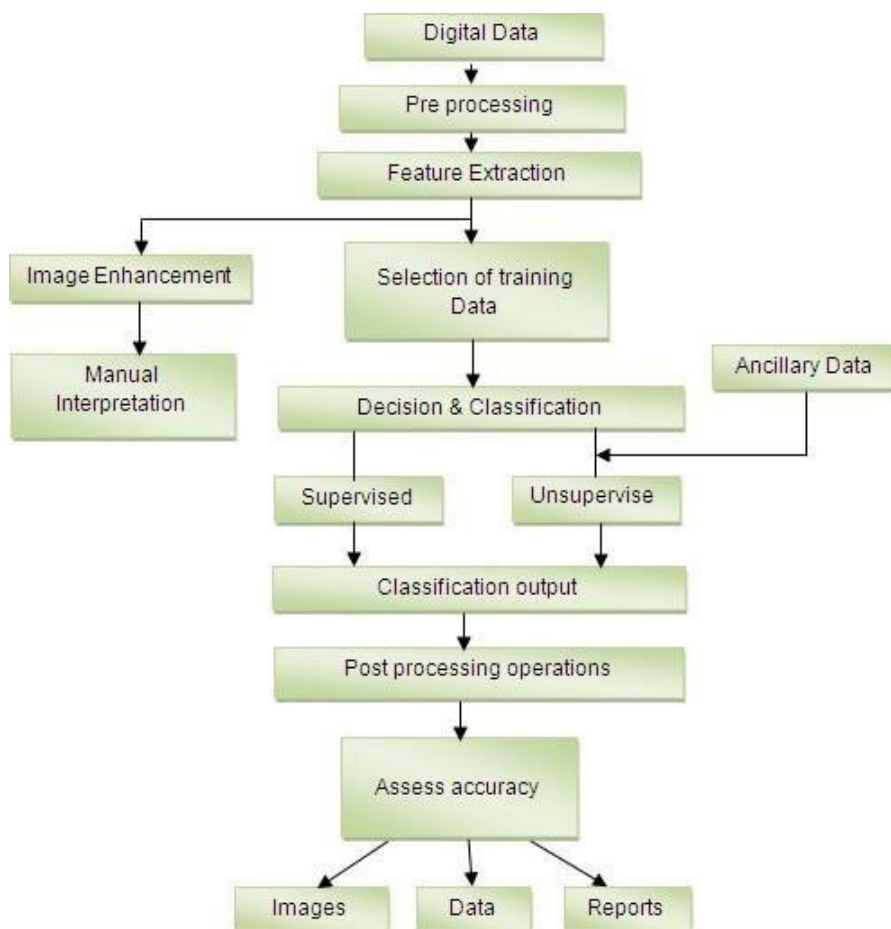


Fig 3.4.1 image processing block diagram

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 $2^{24} = 16777216$ different colors for each color, thus being able to display

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-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

$$2^{24} = 16,777,216$$

allows 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.

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 .

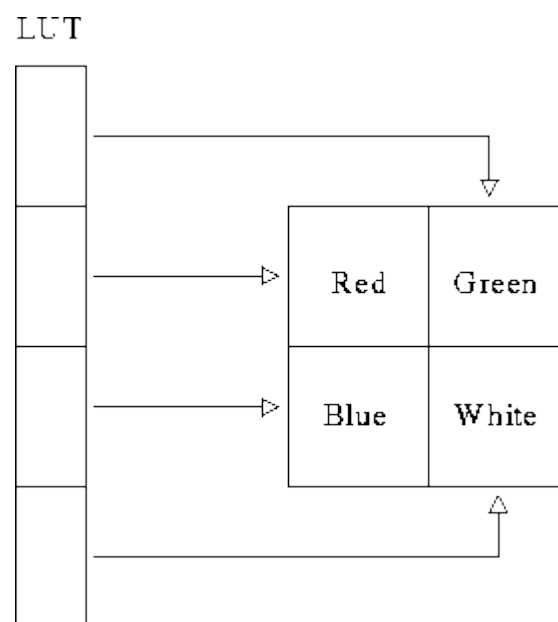


FIG 3.5.6.1 A 2×2 PIXEL AREA DISPLAYING ONE COMPOSITE COLOR

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

Fig3.5.7.1 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.

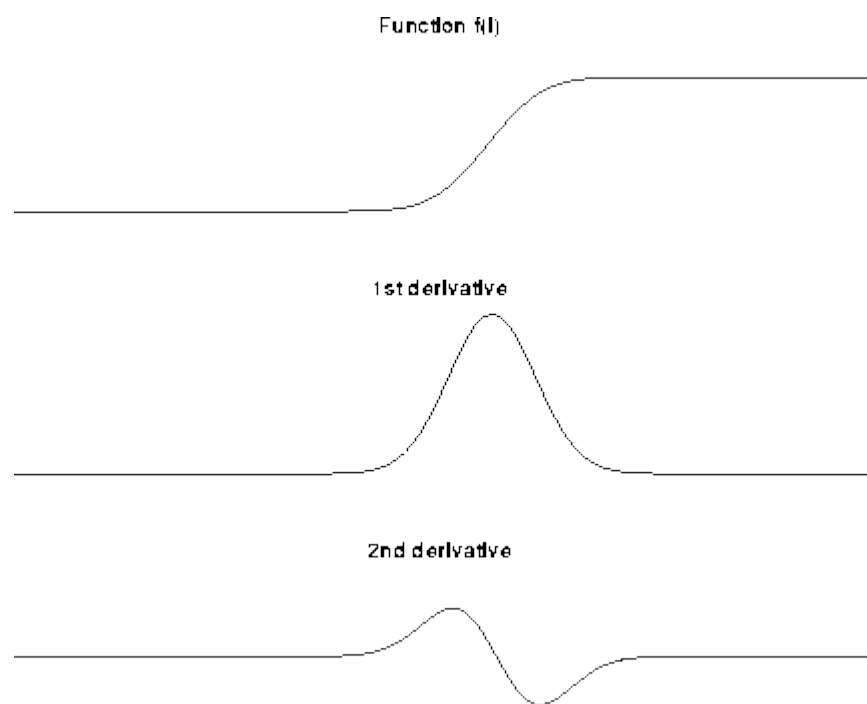


FIG 3.7.1 1ST AND 2ND DERIVATIVE OF AN EDGE ILLUSTRATED IN ONE DIMENSION.

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

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

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.

CHAPTER 4

SOFTWARE DEVELOPMENT METHODOLOGY

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.

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) }

Fig4.2.1 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.

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.

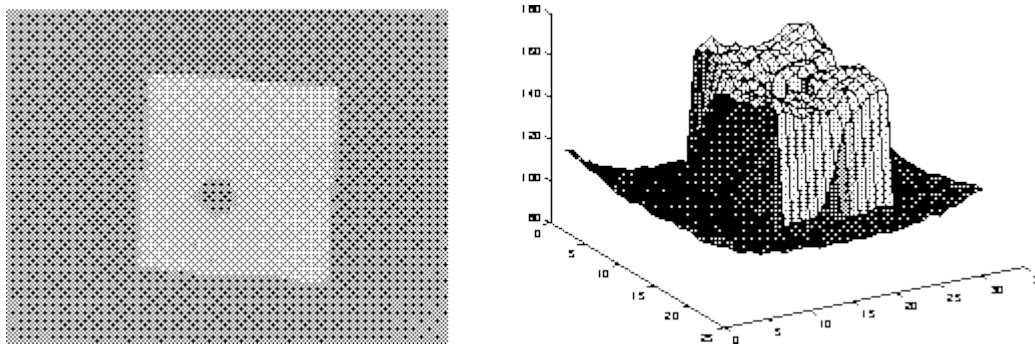


Fig4.4.1 simple graylevel image and the corresponding surface in image space

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 an 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. $V = \{22, 23, \dots, 40\}$, 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.

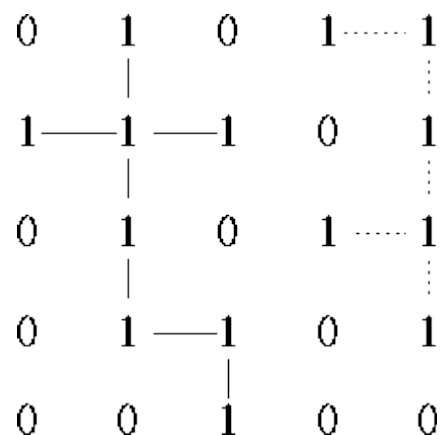


Fig4.6.1.1 Two connected components based on 4-connectivity.

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.

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.

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.

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.

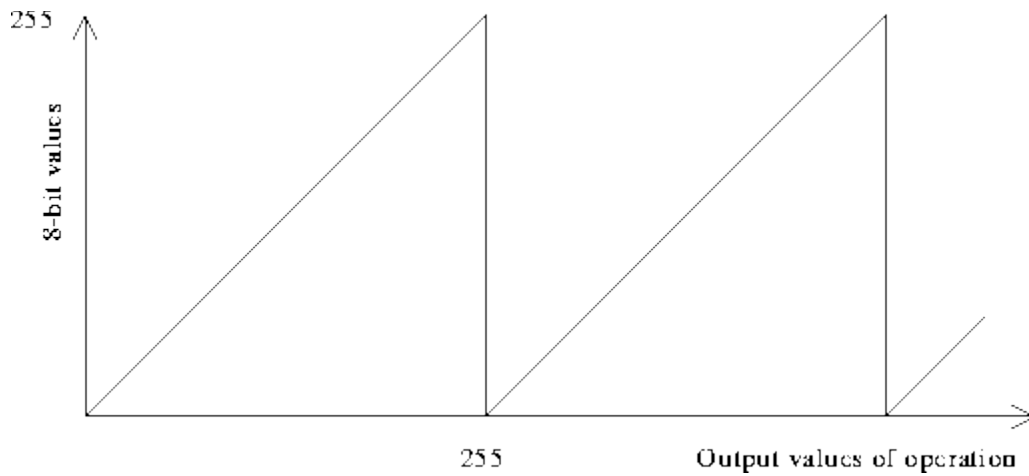


Fig4.8.1 Mapping function for wrapping the pixel values of an 8-bit image.

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.

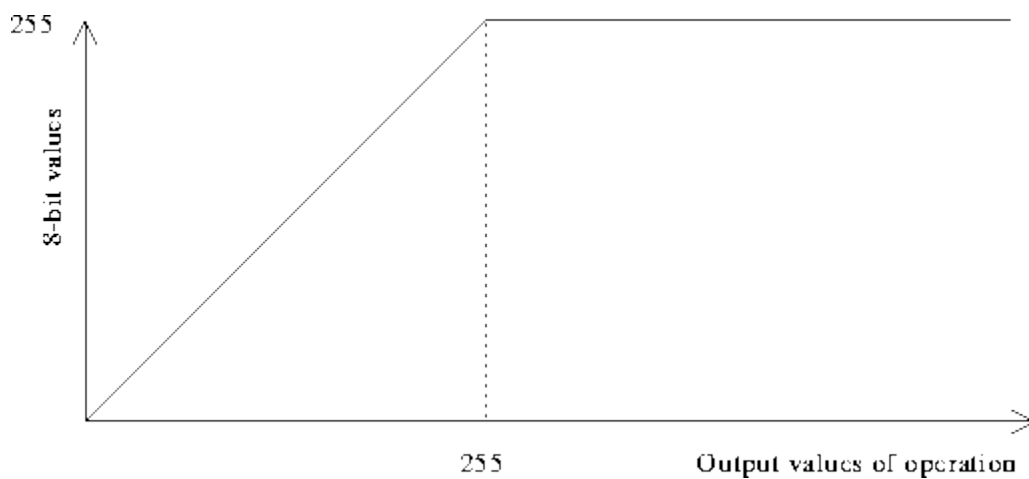


Fig4.8.2 Mapping function for saturating an 8-bit image.

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 [pointoperators](#).

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.

SOFTWARE DESCRIPTION:

MATLAB® is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation. Using MATLAB, you can solve technical computing problems faster than with traditional programming languages, such as C, C++, and Fortran.

Matlab is a data analysis and visualization tool which has been designed with powerful support for matrices and matrix operations. As well as this, Matlab has excellent graphics capabilities, and its own powerful programming language. One of the reasons that Matlab has become such an important tool is through the use of sets of Matlab programs designed to support a particular task. These sets of programs are called toolboxes, and the particular toolbox of interest to us is the image processing toolbox. Rather than give a description of all of Matlab's capabilities, we shall restrict

ourselves to just those aspects concerned with handling of images. We shall introduce functions, commands and techniques as required. A Matlab function is a keyword which accepts various parameters, and produces some sort of output: for example a matrix, a string, a graph. Examples of such functions are sin, imread, imclose. There are many functions in

Matlab, and as we shall see, it is very easy (and sometimes necessary) to write our own. When you start up Matlab, you have a blank window called the Command Window_ in which you enter commands. Given the vast number of Matlab's functions, and the different parameters they can take, a command line style interface is in fact much more efficient than a complex sequence of pull-down menus.

You can use MATLAB in a wide range of applications, including signal and image processing, communications, control design, test and measurement financial modeling and analysis. Add-on toolboxes (collections of special-purpose MATLAB functions) extend the MATLAB environment to solve particular classes of problems in these application areas.

MATLAB provides a number of features for documenting and sharing your work. You can integrate your MATLAB code with other languages and applications, and distribute your MATLAB algorithms and applications.

When working with images in Matlab, there are many things to keep in mind such as loading an image, using the right format, saving the data as different data types, how to display an image, conversion between different image formats.

Image Processing Toolbox provides a comprehensive set of reference-standard algorithms and graphical tools for image processing, analysis, visualization, and algorithm development. You can perform image enhancement, image deblurring, feature detection, noise reduction, image segmentation, spatial transformations, and image registration. Many functions in the toolbox are multithreaded to take advantage of multicore and multiprocessor computers. MATLAB and images

- The help in MATLAB is very good, use it!
- An image in MATLAB is treated as a matrix
- Every pixel is a matrix element

- All the operators in MATLAB defined on
- MATLAB can import/export several image formats

Outside Matlab images may be of three types i.e. black & white, grey scale and colored. In Matlab, however, there are four types of images. Black & White images are called binary images, containing 1 for white and 0 for black. Grey scale images are called intensity images, containing numbers in the range of 0 to 255 or 0 to 1. Colored images may be represented as RGB Image or Indexed Image.

In RGB Images there exist three indexed images. First image contains all the red portion of the image, second green and third contains the blue portion. So for a 640 x 480 sized image the matrix will be 640 x 480 x 3. An alternate method of colored image representation is Indexed Image. It actually exist of two matrices namely image matrix and map matrix. Each color in the image is given an index number and in image matrix each color is represented as an index number. Map matrix contains the database of which index number belongs to which color.

IMAGE TYPE CONVERSION

- RGB Image to Intensity Image (rgb2gray)
- RGB Image to Indexed Image (rgb2ind)
- RGB Image to Binary Image (im2bw)
- Indexed Image to RGB Image (ind2rgb)
- Indexed Image to Intensity Image (ind2gray)
- Indexed Image to Binary Image (im2bw)
- Intensity Image to Indexed Image (gray2ind)
- Intensity Image to Binary Image (im2bw)
- Intensity Image to RGB Image (gray2ind, ind2rgb)

CHAPTER 5

RESULTS AND DISCUSSION

The prediction shows that the color features perform the best among all the three characteristics information and the KNN algorithm performs the best among all four algorithms. The model shows consistently good performance with different cameras and camera settings as well. Discharge produces radiation of UV, visible, and near infrared wavelengths. Past studies focused mostly on the UV spectrum, yet the measurement of the light spectrum demonstrates that radiation intensity of the visible spectrum can be high as well. Objects that can produce radiation of visible spectrum is affected by radiation across all spectra. Because of this, even though our RGB-GLH method uses information only from the visible spectrum, it is still able to encompass discharge status- related information across spectra, thus enabling us to build a more successful model. The RGB color information characteristics method can also be applied to other discharge types other than corona.

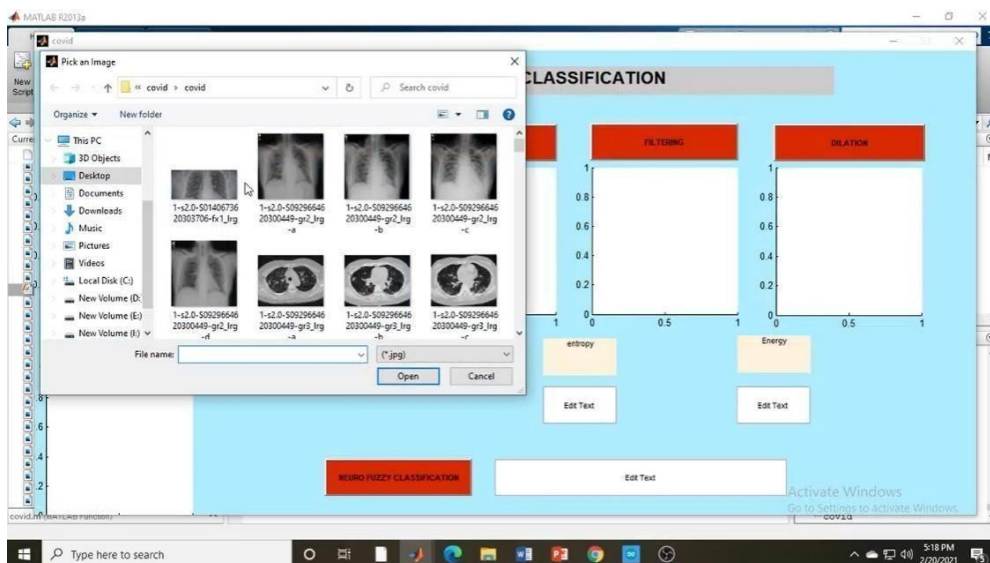


FIG 5.1 HISTOGRAM EQUALIZATION

It is the method to process images in order to adjust the contrast of an image by modifying intensity. It gives brightness to the image. Histogram is a graph showing no of

pixels in an image at different intensities.

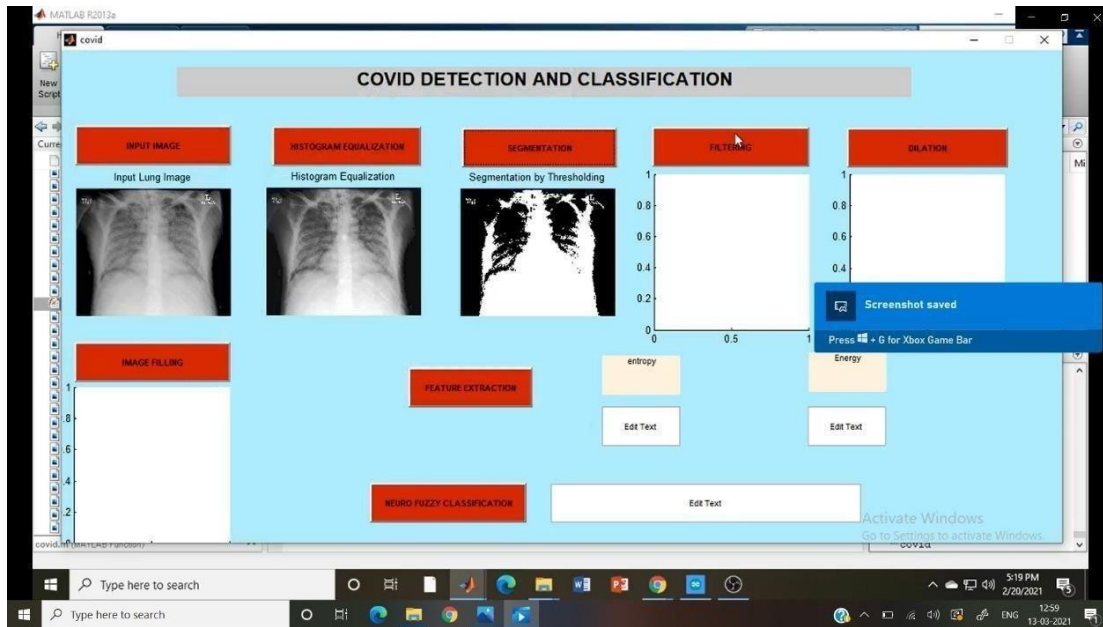


FIG 5.2 SEGMENTATION PROCESS

It is an important stage of image recognition system, because it extracts the objects of our interest. These techniques are used to isolate the desired object from image to perform analysis of image objects. There are some types of segmentation techniques

- Threshold method
- Clustering based Segmentation
- Edge based Segmentation
- Artificial Neural Network based Segmentation

Finally, it is the process of partitioning a digital image into multiple segments. Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labeled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image. A common technique is to look for abrupt discontinuities in pixel values, which typically indicate edges that define a region.

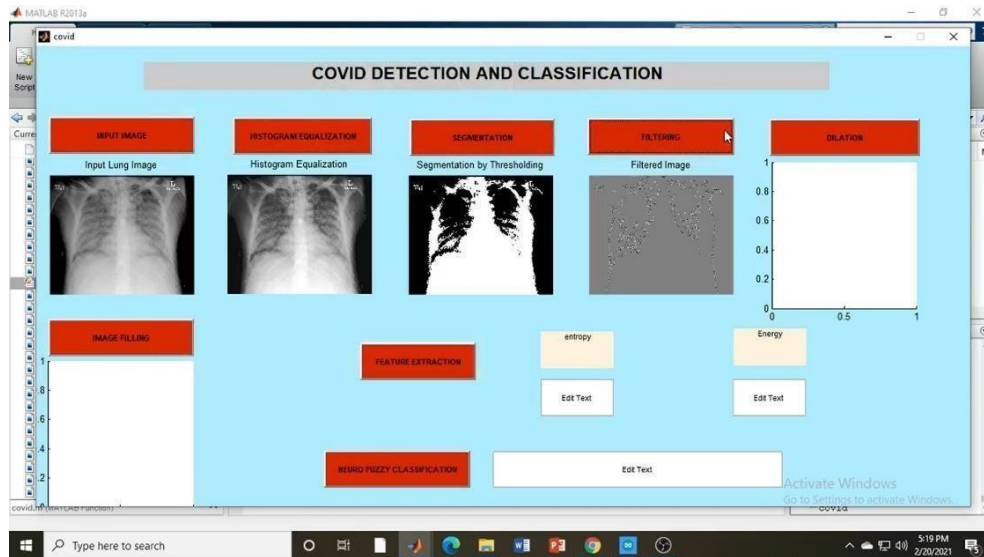


FIG 5.3 FILTRATION PROCESS

It is used to suppress either the high frequencies in the image i.e., smoothing the image or low frequencies detecting the edges in a image. It helps in removing the unwanted features. In **image processing filters** are mainly used to suppress either the high frequencies in the **image**, i.e. smoothing the **image**, or the low frequencies, i.e. enhancing or detecting edges in the **image**. An **image** can be **filtered** either in the frequency or in the spatial domain. In signal processing, a **filter** is a device or process that removes some unwanted components or features from a signal. **Filtering** is a class of signal processing, the defining feature of **filters** being the complete or partial suppression of some aspect of the signal.

Low pass filtering (aka smoothing), is employed to remove high spatial frequency noise from a digital image. The low-pass filters usually employ moving window operator which affects one pixel of the image at a time, changing its value by some function of a local region (window) of pixels. The operator moves over the image to affect all the pixels in the image. A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image - the opposite of the low-pass filter. High-pass filtering works in the same way as low-pass filtering; it just uses a different convolution

kernel.

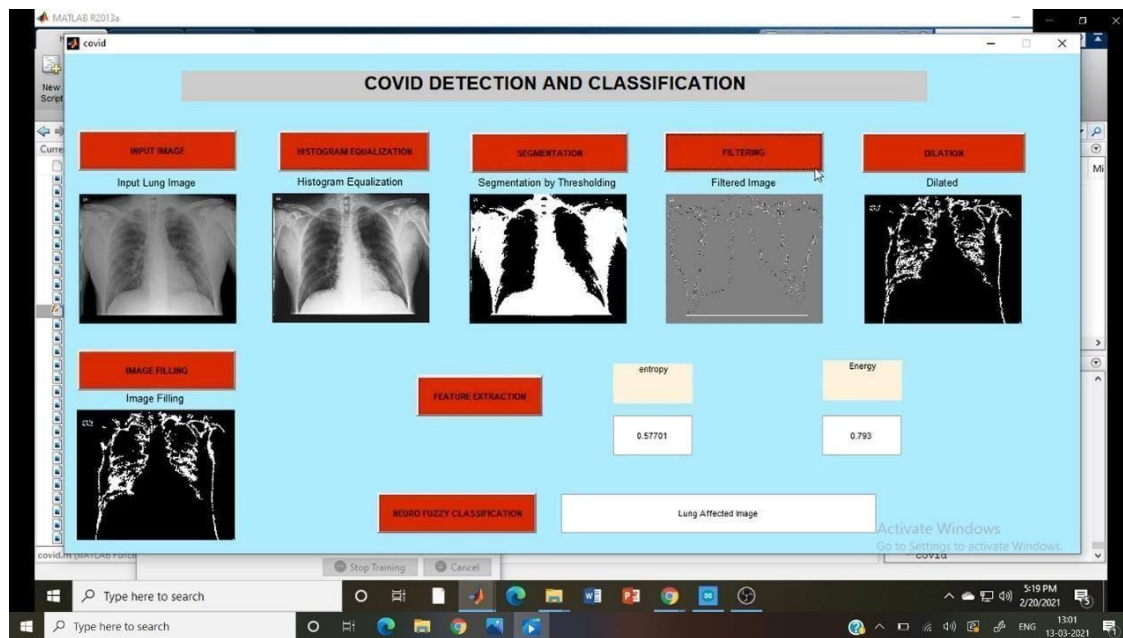


FIG 5.4 DILATION PROCESS

It is a basic operation in mathematical morphology. Morphology means image processing based on the shapes. Dilation and erosion are often used in combination to implement image processing operations. For example, the definition of a morphological *opening* of an image is an erosion followed by a dilation, using the same structuring element for both operations. You can combine dilation and erosion to remove small objects from an image and smooth the border of large objects. The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion. This table lists the rules for both dilation and erosion. Dilation is one of the two basic operators in the area of [mathematical morphology](#), the other being [erosion](#). It is typically applied to [binary images](#), but there are versions that work on

[grayscaleimages](#). The basic effect of the operator on a binary image is to gradually

enlarge the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels grow in size while holes within those regions become smaller.

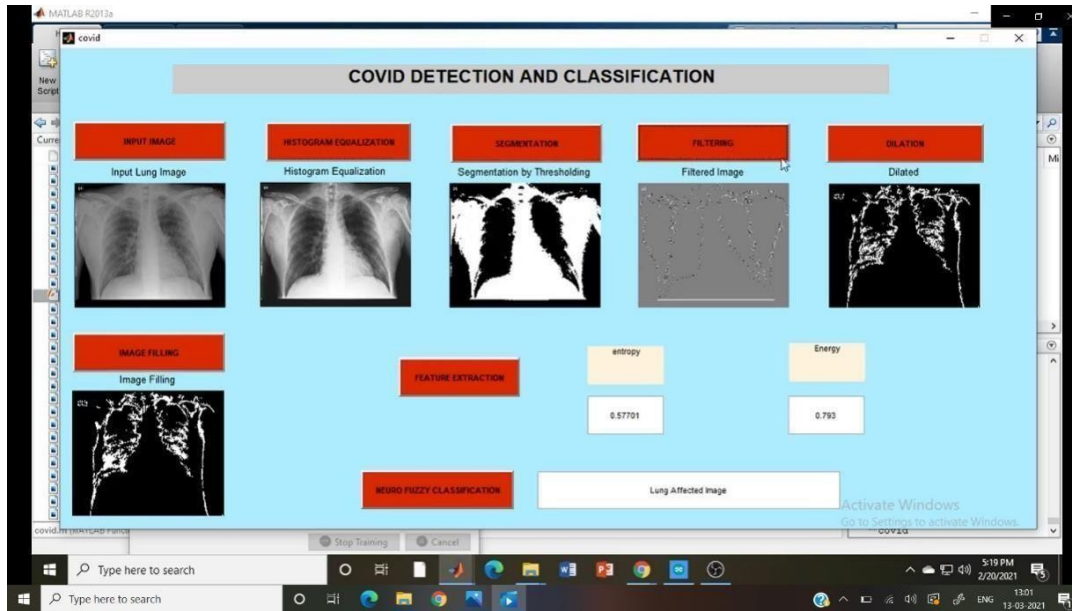


FIG 5.5 FEATURE EXTRACTION

It is a process that identifies important features or attributes of data. This technique is useful when you have a large dataset and to reduce number of resources without losing any important information. Neuro fuzzy classification is also used in image processing i.e it works with the techniques of neural network by the parameters of a system. **Feature extraction** is a part of the dimensionality reduction **process**, in which, an initial set of the raw data is divided and reduced to more manageable groups. The most important characteristic of these large data sets is that they have a large number of variables. Feature extraction methods encompass, structural and graph descriptors. The feature selection methods described in this chapter are the exhaustive search, branch and bound algorithm, max-min feature selection, sequential forward and backward selection, and also Fisher's linear discriminant. Advanced feature representation methods are becoming

necessary when it comes to dealing with the local image content or with [spatio-temporal characteristics](#) or with the statistical image content.

CHAPTER 6

CONCLUSION AND FUTURE

WORK

CONCLUSION

The prediction shows that the color features perform the best among all the three characteristics information and the KNN algorithm performs the best among all four algorithms. The model shows consistently good performance with different cameras and camera settings as well. Discharge produces radiation of UV, visible, and near infrared wavelengths. Past studies focused mostly on the UV spectrum, yet the measurement of the light spectrum demonstrates that radiation intensity of the visible spectrum can be high as well. Objects that can produce radiation of visible spectrum is affected by radiation across all spectra. Because of this, even though our RGB-GLH method uses information only from the visible spectrum, it is still able to encompass discharge status- related information across spectra, thus enabling us to build a more successful model. The RGB color information characteristics method can also be applied to other discharge types other than corona.

FUTURE WORK

In future, this going to make the process of covid testing so easier than present testing process. It is going predict the result quickly in fraction of seconds. It does not require any blood sample of affected person this going to predict the result of covid tested person with help his/her lungs xray report. This technique process the report and analysis the covid on the basis of water content present in the lungs. This can also checked or will be available in online so that the people can check it by their own and get the related treatment quickly. So, people do not face any difficulties in the testing process of covid.

REFERENCES

- [1] Lai, C.C., Shih, T.P., Ko, W.C., Tang, H.J. and Hsueh, P.R., 2020. Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) and corona virus disease-2019 (COVID-19): the epidemic and the challenges. *International journal of antimicrobial agents*, p.105924.
- [2] Gorbalenya, A.E., 2020. Severe acute respiratory syndrome-related coronavirus- The species and its viruses, a statement of the Coronavirus Study Group. *BioRxiv*.
- [3] Huang, C., Wang, Y., Li, X., Ren, L., Zhao, J., Hu, Y., Zhang, L., Fan, G., Xu, J., Gu, X. and Cheng, Z., 2020. Clinical features of patients infected with 2019 novel coronavirus in Wuhan, China. *The Lancet*, 395(10223), pp.497-506.
- [4] Wang, C., Horby, P.W., Hayden, F.G. and Gao, G.F., 2020. A novel coronavirus outbreak of global health concern. *The Lancet*, 395(10223), pp.470-473.
- [5] Carlos, W.G., Dela Cruz, C.S., Cao, B., Pasnick, S. and Jamil, S., 2020. Novel Wuhan (2019-nCoV) Coronavirus. *American journal of respiratory and critical care medicine*, 201(4), pp.P7-P8.
- [6] Zhou, P., Yang, X.L., Wang, X.G., Hu, B., Zhang, L., Zhang, W., Si, H.R., Zhu, Y., Li, B., Huang, C.L. and Chen, H.D., 2020. A pneumonia outbreak associated with a new coronavirus of probable bat origin. *Nature*, pp.1-4.
- [7] Ruuskanen, O., Lahti, E., Jennings, L.C. and Murdoch, D.R., 2011. Viral pneumonia. *The Lancet*, 377(9773), pp.1264-1275.
- [8] Lee, J.Y., Yang, P.C., Chang, C., Lin, I.T., Ko, W.C. and Cia, C.T., 2019. Community-acquired adenoviral and pneumococcal pneumonia complicated by pulmonary aspergillosis in an immunocompetent adult. *Journal of microbiology, immunology, and infection= Wei mian yu gan ran za zhi*, 52(5), p.838.

[9]Su, I.C., Lee, K.L., Liu, H.Y., Chuang, H.C., Chen, L.Y. and Lee, Y.J., 2019. Severe community-acquired pneumonia due to *Pseudomonas aeruginosa* coinfection in an influenza A (H1N1) pdm09 patient. *Journal of Microbiology, Immunology and Infection*,

52(2), pp.365-366.

[10] Pan, S.J. and Yang, Q., 2009. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), pp.1345- 1359.

APPENDIX

SOURCE CODE

```
Function varargout =  
covid(varargin) gui_Singleton =  
1;  
gui_State = struct('gui_Name',    mfilename,  
    ... 'gui_Singleton', gui_Singleton, ...  
    'gui_OpeningFcn',  
    @covid_OpeningFcn, ...  
    'gui_OutputFcn', @covid_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 covid is made visible.
```

```
function covid_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 covid (see VARARGIN)

% Choose default command line output
for covid handles.output = hObject;

% Update handles
structure
guidata(hObject,
handles);

% UIWAIT makes covid wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command
line. function varargout = covid_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) global I
```

```

clc

[filename, pathname] = uigetfile('*.jpg', 'Pick an
    Image'); if isequal(filename,0) |
    isequal(pathname,0) warndlg('File is not
    selected');
        else
            I=imread(filena
me);
            axes(handles.axes1)
            imshow(I);
            title 'Input
            Image' end
title 'Input Lung Image'

% --- 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) global I

t=rgb2gray(I);

he=histeq(t);

```

```
axes(handles.axes  
2); imshow(he);  
title 'Histogram Equalization'  
% --- 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) global I

%% Histogram
Equalization
t=rgb2gray(I);
he=histeq(t);

%% Segmentation by
thresholding threshold =
graythresh(he);
bw = im2bw(he,threshold);

axes(handles.axes
3); imshow(bw)
title 'Segmentation by Thresholding'

% --- 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) global I

%% Histogram Equalization

```

t=rgb2gray
(l);
he=histeq(t
);
%% Segmentation by
thresholding threshold =
graythresh(he);
bw = im2bw(he,threshold);
%% Filter
hy =
fspecial('sobel');
hx = hy';
ly = imfilter(double(bw), hy,
'replicate'); axes(handles.axes4);
imshow(ly,[]),

title('Filtered Image')

% --- Executes on button press in pushbutton5.
global

%% Histogram
t=rgb2gray
he=histeq(

```

```
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)
```

```
global

%% Histogram

t=rgb2gray

he=histeq(
```

```

%% Segmentation by
thresholding threshold =
graythresh(he);
bw = im2bw(he,threshold);

%% Filter
hy =
fspecial('sobel');
hx = hy';
ly = imfilter(double(bw), hy,
'replicate'); se =
strel('line',11,90);
bw2 =
imdilate(ly,se);
axes(handles.axes
5); imshow(bw2),
title('Dilated')

% --- 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)

```

%% Segmentation by
thresholding threshold =
graythresh(he);
bw = im2bw(he,threshold);

%% Filter
hy =
fspecial('sobel');
hx = hy';
ly = imfilter(double(bw), hy,
'replicate'); se =
strel('line',11,90);
bw2 = imdilate(ly,se);

BW5 =
imfill(bw2,'holes');
axes(handles.axes6);
imshow(BW5)

title 'Image Filling'

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)

```

% hObject handle to pushbutton7 (see GCBO)

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

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

global I

```
% figure,imshow(I);

% title 'Input Lung Image'

%% Histogram

Equalization

t=rgb2gray(I);

he=histeq(t);

% figure,imshow(he);

% title 'Histogram Equalization'

%% Segmentation by

thresholding threshold =

graythresh(he);

bw = im2bw(he,threshold);

% figure,imshow(bw)

% title 'Segmentation by Thresholding'

%% Filter

hy =

fspecial('sobel');

hx = hy';

ly = imfilter(double(bw), hy,

'replicate'); se =

strel('line',11,90);
```

```
bw2 = imdilate(Iy,se);  
BW5 =  
imfill(bw2,'holes');  
%% Features Extraction  
%% Contrast  
disp ('Contrast  
Value'); cmap =  
contrast(BW5)
```

```
%% Entropy
```

```
disp ('Entropy
```

```
Value'); E =
```

```
entropy(BW5)
```

```
set(handles.edit2,'string',num2str(E));
```

```
%% GLCM Values
```

```
glcm = graycomatrix(BW5,'Offset',[2 0])
```

```
% --- Executes on button press in pushbutton8.
```

```
function pushbutton8_Callback(hObject, eventdata, handles)
```

```
% hObject handle to pushbutton8 (see GCBO)
```

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

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

```
GUIDATA) global I
```

```
% figure,imshow(I);
```

```
% title 'Input Lung Image'
```

```
%% Histogram
```

```
Equalization
```

```
t=rgb2gray(I);
```

```
he=histeq(t);
```

```
% figure,imshow(he);  
  
% title 'Histogram Equalization'  
  
%% Segmentation by thresholding
```

```

threshold =
graythresh(he); bw =
im2bw(he,threshold);
% figure,imshow(bw)
% title 'Segmentation by Thresholding'
%% Filter
hy =
fspecial('sobel');
hx = hy';
ly = imfilter(double(bw), hy, 'replicate');
% figure, imshow(ly,[]),
% title('Filtered
Image') se =
strel('line',11,90);
bw2 = imdilate(ly,se);
BW5 =
imfill(bw2,'holes');
C=BW5;
%% mean
c1=mean(
C);

```

```
%% variance
```

```
c2=var(double(
```

```
C));
```

```
%%
```

```
contrast
```

```
d=C;
```

```
%% energy, homogeneity, contrast
```

```
GLCM2 = graycomatrix(d,'Offset',[2  
0;0 2]);
```

```

c4 = graycoprops(GLCM2,
{'contrast','homogeneity','Energy'});
set(handles.edit4,'string',num2str(min(c4.Energy)));
c24= graycoprops(GLCM2,'contrast');
set(handles.edit3,'string',num2str(min(c24.Contrast
)));
%% correlation
c5=corr(double(
d)); c6=c5(1,:);
c7=c
1;
c8=c
2;

c9=[c6 c7 c8];
%%
network
net =
network
net.numInputs = 6

```

```
net.numLayers =  
1 P =  
size(double(c1));  
Cidx =  
strcmp('Cancer',c9); T  
= size(double(c2));  
net = newff(P,T,25);  
[net,tr] =  
train(net,P,T);  
testInputs =  
P(:,tr.testInd);  
testTargets = T(:,tr.testInd);  
out = round(sim(net,testInputs));
```

```

diff = [testTargets - 2*out];

detections =
length(find(diff== -1))

false_positives =
length(find(diff== 1))

true_positives =
length(find(diff== 0))

false_alarms =
length(find(diff== -2)) Nt =
size(testInputs,2);

fprintf('Total testing samples: %d\n', Nt);

cm = [detections false_positives; false_alarms
true_positives] cm_p = (cm ./ Nt) .* 100 ;

%%

sim_out =
round(sim(net,testInputs)); if
((max(c24.Contrast))>1)

    set(handles.edit1,'string','Lung Affected
Image'); else

    set(handles.edit1,'string','not
affected'); end

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
```

```
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','w
hite'); 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','w
hite'); 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','w  
hite'); end
```

% --- Executes on button press in pushbutton9.

function pushbutton9_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton9 (see GCBO)

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

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