SPRINGER BRIEFS IN
ELECTRICAL AND COMPUTER ENGINEERING

Laura Caponetti
Giovanna Castellano

Fuzzy Logic for Image Processing

A Gentle Introduction Using Java

Springer

SpringerBriefs in Electrical and Computer Engineering

Laura Caponetti · Giovanna Castellano

Fuzzy Logic for Image Processing

A Gentle Introduction Using Java

Springer

Laura Caponetti
Dipartimento di Informatica
Università degli Studi di Bari Aldo Moro
Bari
Italy

Giovanna Castellano
Dipartimento di Informatica
Università degli Studi di Bari Aldo Moro
Bari
Italy

ISSN 2191-8112 ISSN 2191-8120 (electronic)
SpringerBriefs in Electrical and Computer Engineering
ISBN 978-3-319-44128-3 ISBN 978-3-319-44130-6 (eBook)
DOI 10.1007/978-3-319-44130-6

Library of Congress Control Number: 2016947389

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

To Alfred, Philip and Vito and all my friends

Laura Caponetti

To David and Serena, improved images of me

Giovanna Castellano

Preface

The secret to getting ahead is getting started.

Mark Twain

This book is the result of many years of teaching image processing and fuzzy logic taught by the authors for undergraduate courses. Most of the material used is also the result of fertile interactions with the students whose case studies contributed a lot in the Java implementation of algorithms and methods. The volume has been conceived as a gentle introduction to fuzzy logic approaches useful in image processing tasks.

First we describe image processing algorithms based on fuzzy logic under a methodological point of view. Then, we provide some practical applications without passing over the important formal details. We tried to identify the most important works that researchers have done in the area of fuzzy image processing, and we described and illustrated them through Java examples that the interested readers can easily follow.

The book covers both theoretical and practical applications of fuzzy techniques in image processing. Accordingly, the chapters have been grouped into two parts: *Fundamentals of Fuzzy Image Processing* and *Applications to Image Processing*.

In the first part, we explain how image processing can take advantage of fuzzy logic, giving basic theoretical aspects of both fuzzy logic and image processing through five chapters. Chapter 1 is devoted to the basics of image representation using Java. Chapter 2 deals with low-level image processing. In Chap. 3 the reader will find the basic concepts of fuzzy logic, starting from fuzzy set theory up to fuzzy systems. Chapter 4 discusses the issue of vagueness in digital images, that is the motivation of using fuzzy techniques to process images. Finally, Chap. 5 introduces the Java language and its use for image processing.

In the second part, we present four chapters covering different image processing tasks, namely color contrast enhancement, image segmentation, morphological analysis, and image thresholding. For each task an example of practical application is described. Some examples are presented in the medical domain, using light microscope images provided by the *Dipartimento di Endocrinologia ed Oncologia Molecolare e Clinica* of the University "Federico II" of Naples, Italy. Lastly, the appendix provides some Java code examples that the user can easily run which will help create a concrete feeling of the potential fuzzy image processing.

We believe that this volume will provide a state-of-art coverage of various aspects related to fuzzy image processing and show the potential of fuzzy techniques in solving image processing problems. We hope this book will serve as a reference for scientists and students in this area, as well as a means to stimulate some new ideas for researchers.

We are grateful to a number of people from academic circles as well as from domestic environments who have contributed to the writing of this book in many different ways. In particular, we thank all the members of the CILab (Computational Intelligence Laboratory) of the Department of Informatics at the University of Bari "Aldo Moro" for giving answers to our questions at the right time. We thank Menina Di Gennaro for reading the first draft of some chapters and giving helpful suggestions at the early stages of the work. Our special thanks go to Mara Basile and Vito Corsini who gave their contribution to the research on morphology and segmentation applied to the medical domain. We also thank our Ph.D. student Przemyslaw Gorecky for his contribution on document analysis by fuzzy approaches. Finally, we thank our students Antonio Vergaro, Francesco Tangari, Gabriella Casalino, Marco Lucarelli, and Massimo Minervini for developing some Java examples cited in this book. The contribution of everyone is truly appreciated.

Bari, Italy   Laura Caponetti
June 2016   Giovanna Castellano

Contents

About the Authors

Laura Caponetti received her degree in Physics at the University of Bari, Italy, in 1972. She is Associate Professor (retired) at the Computer Science Department of the University of Bari "A. Moro", where she has been working from 1982 as Assistant Professor. Her research interests are in image processing and computer vision. Her research (https://www.researchgate.net/profile/Laura_Caponetti) spans a range of topics including image segmentation, 3D object recognition, 3D scene analysis, fuzzy and image processing. She has published over eighty papers and she is a referee of international journals and conferences. She has been a member of the scientific committee of the Master in Remote Sensing Techniques and a member of the Council of Ph.D. in Computer Science at the Bari University. She has been a lecturer of Information Processing Systems (Sistemi di Elaborazione della informazione) for the degree course in Computer Science and Computer System Foundations (Fondamenti di Informatica) for the degree course in Civil Engineering. Moreover, she has been a lecturer of Image Processing for the degree course in Computer Science and for the Master in Remote Sensing Techniques. Currently she is a referee of the "Ministero dell'Istruzione, dell'Università e della Ricerca" (http://www.istruzione.it/). Also, she is a member of many scientific associations such as IAPR (Italian Chapter), AICA, and AIIA.

Giovanna Castellano is Associate Professor at the Computer Science Department of the University of Bari "A. Moro", Italy. In 1993 she received her degree in Informatics from the University of Bari. From 1993 to 1995 she was a fellow researcher at the CNR Institute for Signal and Image Processing in Bari, Italy. In 2001 she got a Ph.D. in Informatics from the University of Bari, where she became Assistant Professor in 2002. Her research interests are mainly focused on computational intelligence, with special focus on fuzzy systems, neural networks, neuro-fuzzy modeling, fuzzy clustering, granular computing, and fuzzy image processing. Her current research activity concerns the application of fuzzy techniques in image processing and retrieval. Within these research areas, she has been co-author of more than 170 papers published on scientific journals, book collections, and international conference proceedings. Working on these topics she joined a number of research projects and she organized a number of special sessions and workshops in international conferences. She is Associate Editor of the journal Information Sciences (ISSN:0020-0255) and a member of the editorial board of several international journals.

Part I
Fundamentals of Fuzzy Image Processing

The first part of the book introduces the fundamental concepts of fuzzy image processing; namely, we provide basic concepts of low-level image processing and fundamental principles of fuzzy logic. These concepts are equipped with a basic introduction to the Java programming language.

Chapter 1
Image Representation Using Java

Intelligence is not to make no mistakes, but quickly to see how to make them good.

Bertolt Brecht

Abstract This chapter covers some basic concepts of gray-level and color image representation. Digital images are logically represented using a matrix of elements, each element having a single value in case of gray-level images and three/four values in case of color images. The chapter also introduces the most used color models and the representation of images provided by Java.

1.1 Introduction

A digital image is a two-dimensional representation of an object or a three-dimensional scene resulting from a digitalization process (see Fig. 1.1). An image can be considered as a two-dimensional digital signal acquired by means of two basic processes: sampling in the spatial domain and quantization in the value/level domain. In other words, a digital image is defined as a two-dimensional digital function $f(x, y)$ that is a mapping from a spatial coordinate domain D into an intensity value domain D':

$$f : D \rightarrow D'$$

where D is a finite domain consisting of pairs of discrete coordinates (x, y) and D' is the discrete domain of values called *gray levels*. Typically the domain D' is the range [0, 255]. A digital image consists of a finite set of elements having a location (x, y) and a value denoted by l. Each element $f(x, y)$ is called *pixel*—picture element.

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_1

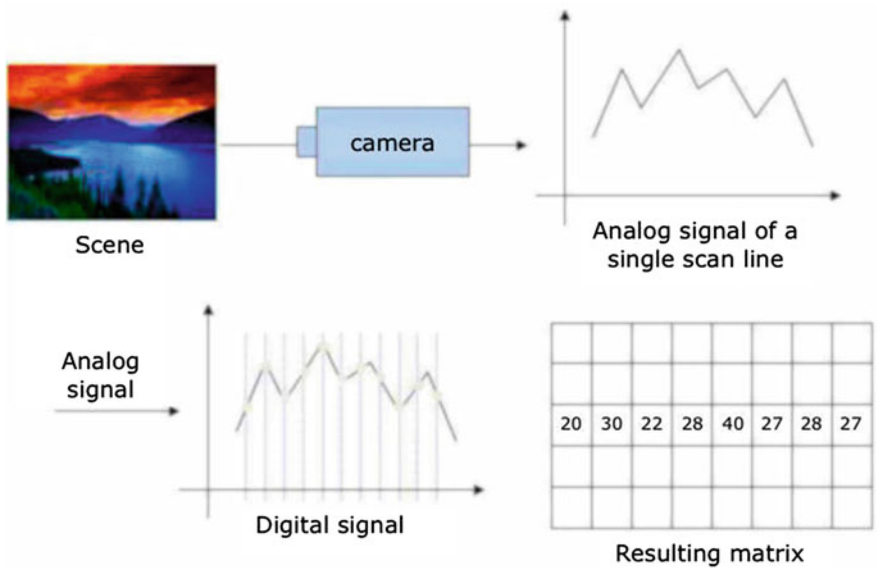


Fig. 1.1 Image acquisition and digitalization process

Digital image processing refers to all those tasks that involve processing a digital image by a computer. We can distinguish among low-level, medium-level, and high-level processing. In low-level processing tasks we have a digital image as input and a digital image as output—for example an image improved for the visualization. In high-level processing the outcome is a description of the content of the input image. In medium-level processing some features are obtained from the input image, for examples edges or regions. This book mainly focuses on low-level and medium-level image processing.

This chapter provides a brief introduction to digital image processing. The aim is to introduce the fundamentals of gray-level and color image processing, with special focus on the tasks considered as applications of fuzzy techniques in the second part of the volume. For further details about the basics of image processing, the reader is referred to specific bibliography [1, 3].

1.2 Gray-Level Images

A digital image f having a finite number of non negative values—typically in the range [0, 255]—is called gray-level image. It can be represented by a matrix of $N \times M$ elements, where N is the row number and M the column number. Each element $f(j, k)$ for $0 \leq j \leq N$ and $0 \leq k \leq M$ is a pixel having abscissa $x = k$ and ordinate $y = -j$. Then a digital image can be represented using the logical coordinate system (x, y) or the image coordinate system (j, k), where $x = k$ is the column and $y = -j$ is the row (see Fig. 1.2).

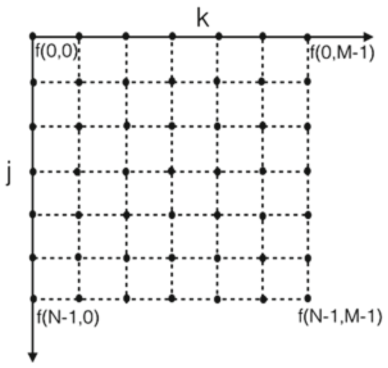


Fig. 1.2 Image coordinate system

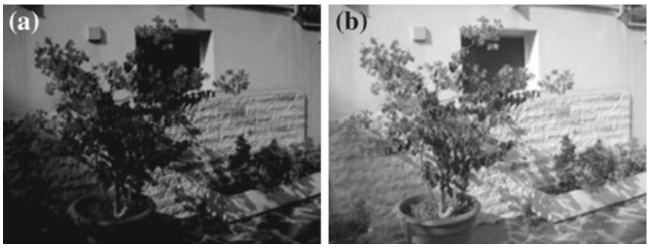


Fig. 1.3 A gray-level image with **a** high dynamic range and **b** low dynamic range

A digital image can be described by means of some features such as

- *dimension*: the number of pixels in the image.
- *size*: the number of rows and columns of the image or also the size (width and height) expressed in cm/inch.
- *spatial resolution*: the number of pixels for size unity expressed in dot per inch (dpi).
- *dynamic range* or resolution in the gray-level range: the number of different gray levels actually used in the image. For example if the range is [0, 255], an image using all 256 levels presents a high dynamic range. Conversely, an image using only few levels presents a very low dynamic range (Fig. 1.3).

1.3 Color Models

There are basically two ways to specify colors in a computer. The RGB (Red Green Blue) definition is the more natural approach in terms of the human visual system and it is also the approach used to drive computer monitors. In fact the RGB system matches with the fact that the human eye is strongly perceptive to red, green, and blues primaries.

The secondary colors CMY (Cyan Magenta Yellow) are used for printing and are basically complementary to RGB. However, the RGB and CMY color models are not well suited for describing color for human interpretation. Defining a specific color using a RGB set of numbers is very difficult, unless it is one of the end members. Indeed, one does not refer to the color of an object by giving the percentage of each primary component.

For these reasons other color representations are adopted, such as the HSB representation where the features used to distinguish one color from another are Hue, Saturation, and Brightness. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Hue represents dominant color as perceived by an observer. Saturation refers to the relative purity or the amount of white light mixed with the hue. The pure spectrum colors are fully saturated and the degree of saturation being inversely proportional to the amount of light added. Brightness embodies the chromatic notion of intensity. Hue and saturation together are called *chromaticity* so a color can be represented by its brightness and chromaticity.

A color model or color system is a subspace of a three-dimensional coordinate system in which each color is represented by a single point. Generally a color model is oriented either toward hardware (as for monitor and printers) or toward applications.

The most common hardware-oriented color models are

- the RGB model for color monitor and color video cameras;
- the CMY model for color printers;

By contrast, the HSV (Hue, Saturation, and Value) and HSI (Hue, Saturation, and Intensity) color models, referred to the HSB representation, are user/application oriented and correspond to the way humans perceive and describe color using the words tint, shade, and tone.

RGB Color Model

In the RGB system each color is represented by its primary components relative to Red, Green, and Blue. The model is based on a Cartesian coordinate system. The RGB space is represented by a cube (see Fig. 1.4), in which the primary colors are at three corners; cyan, magenta, and yellow are at the three opposite corners. Black is at the origin and white is at the corner farthest from the origin. The gray scale—points of equal RGB values—extends from the black point to white point along the line joining these two points. The different colors in this model are points on or inside the cube. They are defined by vectors extending from the origin of the Cartesian system.

HSV Color Model

The HSV color system, also called HSB (B for Brightness) is closer than RGB system to the way human describe color sensations. In artistic terminology hue,

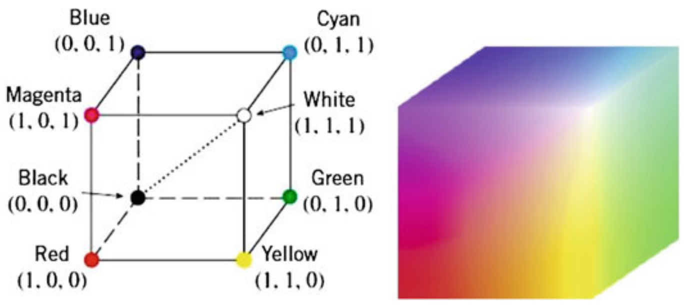
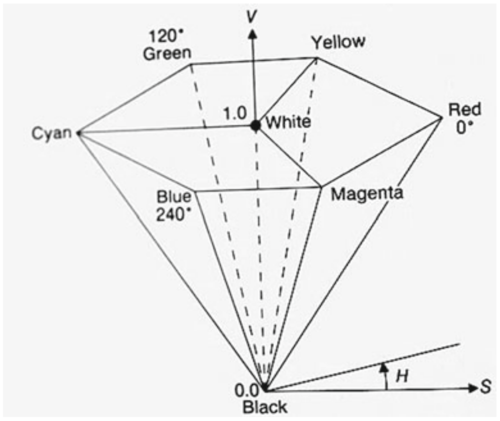


Fig. 1.4 RGB color model (taken from [3])

Fig. 1.5 HSV-HSB color model



saturation, and value refer to tint, shade, and tone. The HSV model is defined in a subspace represented by a hexacone or six-sided cone (Fig. 1.5). The HSV is based on cylindrical coordinate. A cylindrical coordinate system is a three-dimensional coordinate system (ρ, ϕ, z) that specifies point positions by the distance ρ from a chosen reference axis, the direction ϕ from the axis relative to a chosen reference direction, and the distance z from a chosen reference plane perpendicular to the axis. Hue, corresponding to ϕ, is expressed as an angle around a color hexagon using the red axis as the 0 axis. Value, corresponding to z, is measured along the axis of the cone lying in the center of the color hexagon. The end of the axis $V = 0$ represents black, while the end $V = 1$ is white. Thus the cone axis represents all shades of gray. Saturation, corresponding to ρ, is measured as the distance from the V axis.

The conversion from a RGB value to a HSV value is given in the following [2]. We define $\Delta = \max(R, G, B) - \min(R, G, B)$. As concerns the Hue component, there are two cases. If $R = G = B$ then $\Delta = 0$ and the Hue is undefined. If $\Delta > 0$ then we compute

$$R^* = (\max(R, G, B) - R)/\Delta$$
$$G^* = (\max(R, G, B) - G)/\Delta$$
$$B^* = (\max(R, G, B) - B)/\Delta$$

and then

$$H^* = \begin{cases} B^* - G^* & \text{if } R = \max(R, G, B) \\ R^* - B^* + 2 & \text{if } G = \max(R, G, B) \\ G^* - R^* + 4 & \text{if } B = \max(R, G, B) \end{cases}$$

Finally, by normalizing, we have

$$H = \begin{cases} \frac{1}{6}(H^* + 6) & \text{for } H^* < 0 \\ H^* & \text{otherwise} \end{cases} \tag{1.1}$$

The S and V components are defined as follows:

$$S = \begin{cases} \max(R, G, B)/\Delta & \text{if } \Delta > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

$$V = \max(R, G, B)/Cmax \tag{1.3}$$

where $Cmax$ is the maximum value in the RGB scale (generally $Cmax = 255$).

HSI Color Model

The HSI color model is based on triangular and circular planes, as depicted in Fig. 1.6. The triangles and circles are perpendicular to the vertical intensity axis. The Hue (H) is measured by the angle around the vertical axis and has a range of values between 0° and 360° beginning with red at 0°. It gives a measure of the spectral composition of a color. The saturation (S) is a ratio that ranges from 0 (i.e. on the I axis) extending radially outwards to a maximum value of 1 on the surface of the cone. This component refers to the proportion of pure light of the dominant wavelength and indicates how far a color is from a gray of equal brightness. The intensity (I) also ranges between 0 and 1 and measures the relative brightness. At the top and bottom of the cone, where I = 0 and I = 1, respectively, H and S are undefined and meaningless. At any point along the I axis the saturation component is zero and the Hue is undefined. This singularity occurs whenever R = G = B.

$$H = \cos^{-1}\left\{\frac{\frac{1}{2}(R - G) + (R - B)}{\sqrt{(R - G)^2 + (R - B)(R - G)}}\right\} \tag{1.4}$$

$$S = 1 - \frac{3}{R + G + B}[\min(R, G, B)] \tag{1.5}$$

$$I = \frac{1}{3}(R + G + B) \tag{1.6}$$

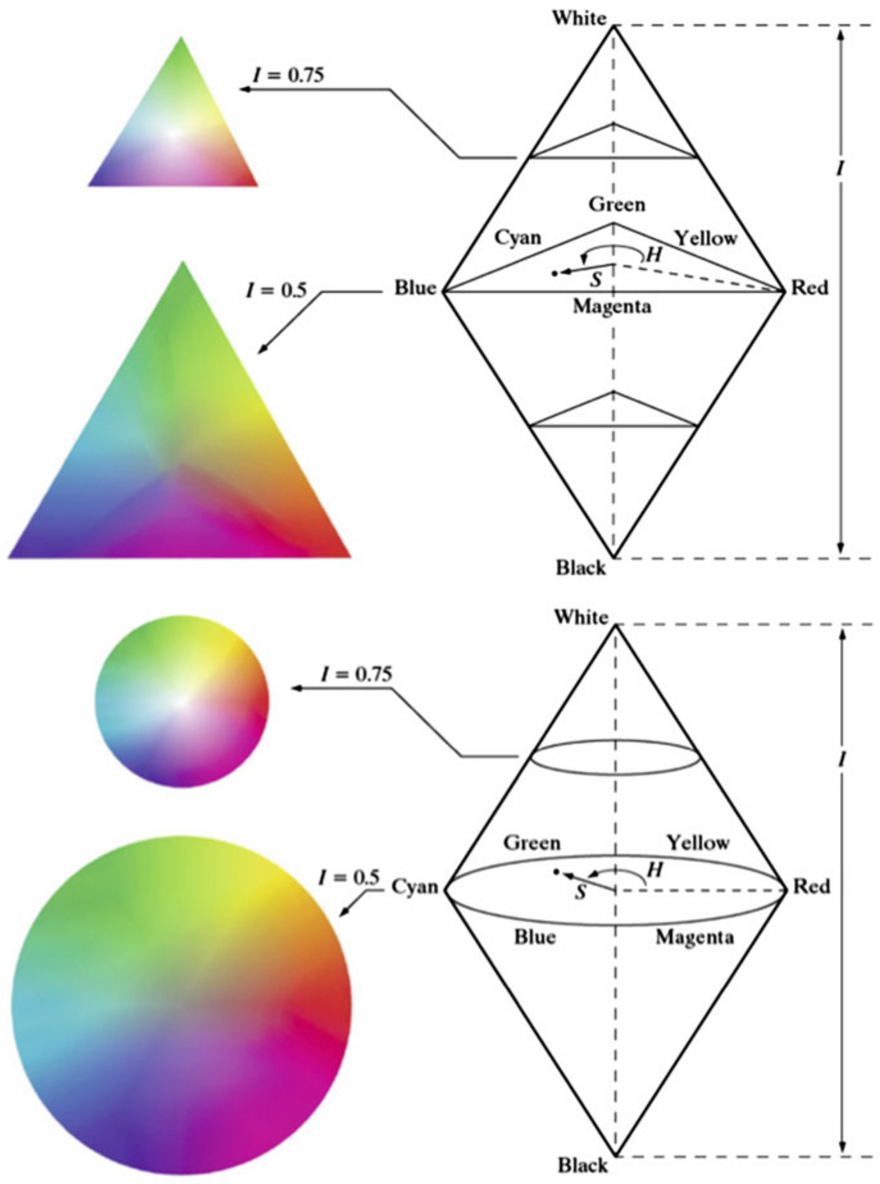


Fig. 1.6 HSI color model (taken from [3])

The HSV/HSI systems provide a more natural way to define a color: the value of hue sets the color according to the colors of the rainbow red, orange, yellow, green, blue, violet, and back to red. Decreasing the value of brightness moves the color toward black and decreasing the saturation moves the color toward white. The reason is that HSV/HSI systems allow movements in color space which correspond more closely to what we mean by tint and shade. An instruction like add white is easy in HSI but not so obvious in RGB. The HSV color system is somewhat similar to HSI system, but its aim is to present colors that are meaningful when interpreted in terms of a color artist palette.

1.4 Color Image Representation Using Java

To represent a color image it is necessary to define the color model. Then a color pixel can be represented directly by means of three components (e.g., Red, Green, Blue). If a byte is used to represent each component, it is possible to represent $256 \times 256 \times 256$ different colors (about 16 millions). The BitMaP format uses the direct representation of the color, also known as true color representation. Often to limit the memory size a number of four or eight bits are used for each pixel. In this case it is necessary to use also a palette or colormap, that is a look-up table in which every element contains a tuple of three values RGB. In this case each pixel value is an entry in the look-up table. Then this representation is an indirect or indexed representation of the color value. The GIF, TIF, and PNG formats use an indexed representation of the color and use a look-up table of 4 or 8 bit, hence each pixel is an index into a palette of 16 or 256 colors.

Java permits to memorize and process a color image by means of the package `image`. In Chap. 5 an introduction to Java and ImageJ plugin for image processing is presented.

Java represents a RGB color image by an array of pixels. Each color pixel is represented in a packed mode using a 32 bit integer value, where the high order byte represents the alpha component followed by the Red, Green, and Blue components. The alpha value represents the level of transparency of the pixel varying from 0 (transparent pixel, i.e., invisible) to 255 (opaque pixel). In a gray-level image the three components (R, G, B) have the same value. To transform a color pixel into a gray-level one we can use the following formulas:

$$I = (R + G + B) \tag{1.7}$$

Since the eye is more sensible to Green and Red than to Blue color, usually a weighted sum that takes in account the different perception of the human eye for the three fundamental colors, is used

$$I = 0.299R + 0.587G + 0.114B \tag{1.8}$$

Java supports several image formats for RGB true color images, such as TIFF, BMP, JPEG, PNG, and RAW. Moreover it supports formats for RGB indexed color images such as GIF, PNG, BMP, and TIFF. In particular the Java class `ColorProcessor` provides a support to process easily color images in RGB and HSB spaces by offering the following functions:

1. creation of a new image
2. conversion from RGB to HSB and vice versa
3. splitting or merging the color components
4. managing a stack of components or slides by adding a slice or deleting, merging slides into an image.

The Listing 1.1 shows a Java plugin to convert a color image from RGB to a HSB (HSV) stack using the Eqs. (1.1), (1.2) and (1.3).

Listing 1.1 A Java program to convert RGB to HSI.

```java
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
/* Splits an RGB image into three 8-bit grayscale components
 (hue, saturation and brightness) */
public class RGB_Splitter_into_HSI_components
                  implements PlugInFilter {
    ImagePlus imp;
  public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_RGB+NO_UNDO;
    }
    public void run(ImageProcessor ip) {
         int w = imp.getWidth();
         int h = imp.getHeight();
     ImageStack hsbStack = imp.getStack();
         ImageStack hueStack = new ImageStack(w,h);
         ImageStack satStack = new ImageStack(w,h);
         ImageStack brightStack = new ImageStack(w,h);
      byte[] hue,s,b;
         ColorProcessor cp;
         int n = hsbStack.getSize();
         for (int i=1; i<=n; i++) {
             IJ.showStatus(i+"/"+n);
             hue = new byte[w*h];
             s = new byte[w*h];
             b = new byte[w*h];
             cp = (ColorProcessor) hsbStack.getProcessor(1);
             cp.getHSB(hue,s,b);
             hsbStack.deleteSlice(1);
            //System.gc();
             hueStack.addSlice(null,hue);
             satStack.addSlice(null,s);
             brightStack.addSlice(null,b);
             IJ.showProgress((double)i/n);
        }
        String title = imp.getTitle();
        imp.hide();
        new ImagePlus("(hue)" + title,hueStack).show();
        new ImagePlus("(saturation)" + title,satStack).show();
        new ImagePlus("(brightness)" + title,brightStack).show();
    }
}
```

Listing 1.2 Color inverter Java plugin.

```java
import ij.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
 /* ColorInverter
  * Inverts the pixels in the ROI of a RGB image.
  * This is an example from the ImageJ plugin writing tutorial:
  * http://www.fh-hagenberg.at/mtd/depot/imaging/imagej
  */
  public class ColorInverter_ implements PlugInFilter {
  public int setup(String arg, ImagePlus imp) {
    if (arg.equals("about"))
      {showAbout(); return DONE;}
                return DOES_RGB+NO_CHANGES;
  }
//
  public void run(ImageProcessor ip) {
    // get width, height and the region of interest
    int w = ip.getWidth();
    int h = ip.getHeight();
    Rectangle roi = ip.getRoi();
    // create a new image with the same size
    // and copy the pixels of the original image
    ImagePlus inverted = NewImage.createRGBImage(
              "Inverted␣image", w, h, 1, NewImage.FILL_BLACK);
    ImageProcessor inv_ip = inverted.getProcessor();
    inv_ip.copyBits(ip,0,0,Blitter.COPY);
    int[] pixels = (int[]) inv_ip.getPixels();
    // invert the pixels in the ROI
    for (int i=roi.y; i<roi.y+roi.height; i++) {
      int offset =i*w;
      for (int j=roi.x; j<roi.x+roi.width; j++) {
        int pos = offset+j;
        int c = pixels[pos];
        int r = (c&0xff0000)>>16; // extract red component
        int g = (c&0x00ff00)>>8;  // extract green component
        int b = (c&0x0000ff);     // extract blue component
        r=255-r;
        g=255-g;
        b=255-b;
        pixels[pos]=((r & 0xff)<<16)+((g & 0xff)<< 8)+(b & 0xff);
      }
    }
    inverted.show();
    inverted.updateAndDraw();
  }
  void showAbout() {
    IJ.showMessage("ColorInverter","inverts␣ROI␣of␣a␣RGB␣image");
  }
}
```

The Listing 1.2 shows an example of Java plugin to convert a RGB value, packed in an integer `c`, into the three fundamental components `[r g b]` and then to invert each component and produce a new image with the color scale inverted. To isolate each color component, for each pixel `[u,v]` a bit-wise AND operation is applied to an appropriate bit mask expressed in hexadecimal notation. After the extracted bits are shifted right 16 bit positions for the `r` component, and right 8 positions for the `g` component, as shown in the Listing 1.2. The details of the code can be better understood by reading Chap. 5 about Java introduction for image processing.

References

1. Burger, W., Burge, M.J.: Digital Image Processing: An Algorithmic Introduction Using Java. Springer Science & Business Media, Heidelberg (2009)
2. Foley, J.D., Van Dam, A.: Fundamentals of Interactive Computer Graphics, vol. 2. Addison-Wesley, Reading (1982)
3. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice Hall, Upper Saddle River (2008)

Chapter 2
Low-Level Image Processing

As regards obstacles, the shortest distance between two points can be a curve.

Bertolt Brecht

Abstract This chapter covers some basic concepts of low-level image processing. It introduces fundamental methods for two primary image processing tasks, namely contrast enhancement, image smoothing, and edge detection. The chapter also introduces methods of function optimization for searching the optimal configuration of edge points.

2.1 Introduction

In the previous chapter, we have introduced the concept of low-level, medium-level, and high-level digital image processing. In low-level processing tasks a digital image is used as input and another digital image is obtained as output (e.g., an image improved for the visualization). In high-level processing the outcome is a description of the content of the input image. In the medium-level processing some features are obtained from the input image, such as edges or regions.

Different operators are adopted for low-level processing. Usually, we distinguish among the following operators:

1. *Point operators* that produce a single output pixel by processing each pixel independently of the other pixels.
2. *Local operators* that produce a single output pixel by processing a neighborhood of that pixel.
3. *Global operators* that produce a single output pixel by processing the entire image.

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_2

Generally the aim of low-level operators is to improve the visual quality of the input image by means of enhancement or noise removal processes. In this chapter we introduce two fundamental image processing tasks that are contrast enhancement and edge detection. To this aim, some notations are given in the following.

Let $f(x, y)$ a gray-level image of $M \times N$ pixels with L gray levels. We introduce the following definitions:

- the *level scale* or *dynamic range* of image $f(x, y)$ is the range $[a, b]$ of values such that $a \leq f(x, y) \leq b$ for each (x, y);
- the *histogram* $h(i), i = 0, \ldots, L-1$ of image $f(x, y)$ denotes the occurrence frequency of each level.

In Listing 2.1, we provide a simple applet to compute the histogram of a gray-level image. The applet can be executed using the HTML code listed in 2.2. To run the applet it is necessary to create a Java project with the files HistogramApplet.java and HistogramApplet.html, then include in the same directory an image named `im.png` to be processed.

In the RGB color space, individual histograms of each component can be computed. In [5] we find a plugin to compute the histogram of each R, G, B component given an input color image.

Listing 2.1 HistogramApplet.java: a Java applet for histogram visualization.

```
import java.awt.*;
import java.awt.image.*;
import java.applet.Applet;
public class HistogramApplet extends Applet {
  private Image image;
  private ImageCanvas imageCanvas;
  private Panel panel;
  private TextArea text;

  public void init() {
    String image_file = getParameter("IMAGEFILE");
    image = getImage(getDocumentBase(), image_file);
    while(image.getWidth(this)<0);
    Dimension imageSize = new Dimension(
        image.getWidth(this), image.getHeight(this));
    imageCanvas = new ImageCanvas(image, imageSize);
    int[] pixels = ImageCanvas.grabImage(image, imageSize);
    panel = new Panel(new GridLayout(1,2,10,10));
    text = new TextArea(20,5);
    panel.add(imageCanvas);
    panel.add(text);
    add(panel);
    text.setText((new Histogram(pixels)).toString());
  }
}
class Histogram {
private int histo[] = new int[256];
  public String toString() {
    String text = "";
    for(int i=0; i<256; i++) {
      text += i+"␣"+histo[i]+'\n';
    }
    return text;
  }
  public Histogram(int[] rgb) {
    for(int i=0; i<rgb.length; i++) {
      int tmp = (int) (
        (((rgb[i] & 0xff0000)>>16) * 0.299) +
        (((rgb[i] & 0x00ff00)>>8 ) * 0.587) +
        (((rgb[i] & 0x0000ff)    ) * 0.114) );
      histo[tmp]++;
    }
  }
  public int getValueAt(int index) {
    return histo[index];
  }
}
 class ImageCanvas extends Canvas {
 static final int MIN_WIDTH = 64;
 static final int MIN_HEIGHT = 64;
 private Image image;
 private Dimension size;

  public ImageCanvas(Image img, Dimension dim) {
    super();
    image = img;
    size = dim;
  }
  public Dimension getMinimumSize() {
    return new Dimension(MIN_WIDTH, MIN_HEIGHT);
  }
  public Dimension getPreferredSize() {
    return new Dimension(size);
  }

  public void paint(Graphics g) {
    g.drawImage(image, 0, 0, getBackground(), this);
  }
static public int[] grabImage(Image image, Dimension size) {
    int[] data = new int[size.width * size.height];
    PixelGrabber pg = new PixelGrabber(
      image, 0, 0, size.width, size.height, data, 0, size.width);

    try {
      pg.grabPixels();
    }
    catch (InterruptedException e) {
      System.err.println(
        "ImageSampler:␣interrupted␣while␣grabbing␣pixels");
      return null;
    }
    if ((pg.status() & ImageObserver.ABORT) != 0) {
      System.err.println(
        "ImageSampler:␣pixel␣grab␣aborted␣or␣errored");
      return null;
    }
    return data;
   }
   }
```

Listing 2.2 HistogramApplet.html: Html code to run `HistogramApplet.java`.

```
<html>
<head><title>Histogram</title></head>
<body>
<H1>Histogram</H1>
<applet
   name="HistogramApplet"
   code="HistogramApplet.class"
   width="800"
   height="500"
   alt="If␣you␣have␣a␣Java-enabled␣browser,
 ␣␣␣you␣would␣see␣an␣applet␣here.">
<param name="IMAGEFILE" value="im.png">
</applet>
</body>
</html>
```

Generally, to define each low-level operator, a mapping T that transforms an input image $f(x, y)$ into an output image $g(x, y)$ has to be defined over some neighborhood of each pixel. Namely

$$g(x, y) = T(f(x, y)) \tag{2.1}$$

where T is a linear or nonlinear function defined on the dynamic range $[a, b]$.

2.2 Contrast Enhancement

The contrast of an image refers to the range of gray levels used in the image—the dynamic range. It refers to the intensity variation of the pixels, defined by the minimum and maximum intensity value. Contrast resolution is the ability to distinguish between differences in intensity. For example, low contrast image values may be concentrated near a few values of the gray scale (e.g., mostly dark, or mostly bright, or mostly medium values). One definition of image contrast is the following:

$$C = \frac{S_A - S_B}{S_A + S_B}$$

where S_A and S_B are intensity average values computed on pixels of two different regions A and B (for example background and object).

Low contrast images can result from poor illumination, lack of dynamic range in the imaging sensor, or even wrong set up during image acquisition. A fundamental low-level task is to improve the contrast in an image, by means of *contrast enhancement operators*.

To improve the contrast it is necessary to transform the levels of the image into the range of all the levels available for visualization (typically the range $[0, 255]$). Specifically, a contrast stretching, that means highlighting a specific range of gray levels in an image, is performed. The idea behind contrast stretching is to increase the dynamic range in the image being processed. Moreover, to enlarge the dynamic range it is necessary to interpolate between successive levels. Figure 2.1 shows an example.

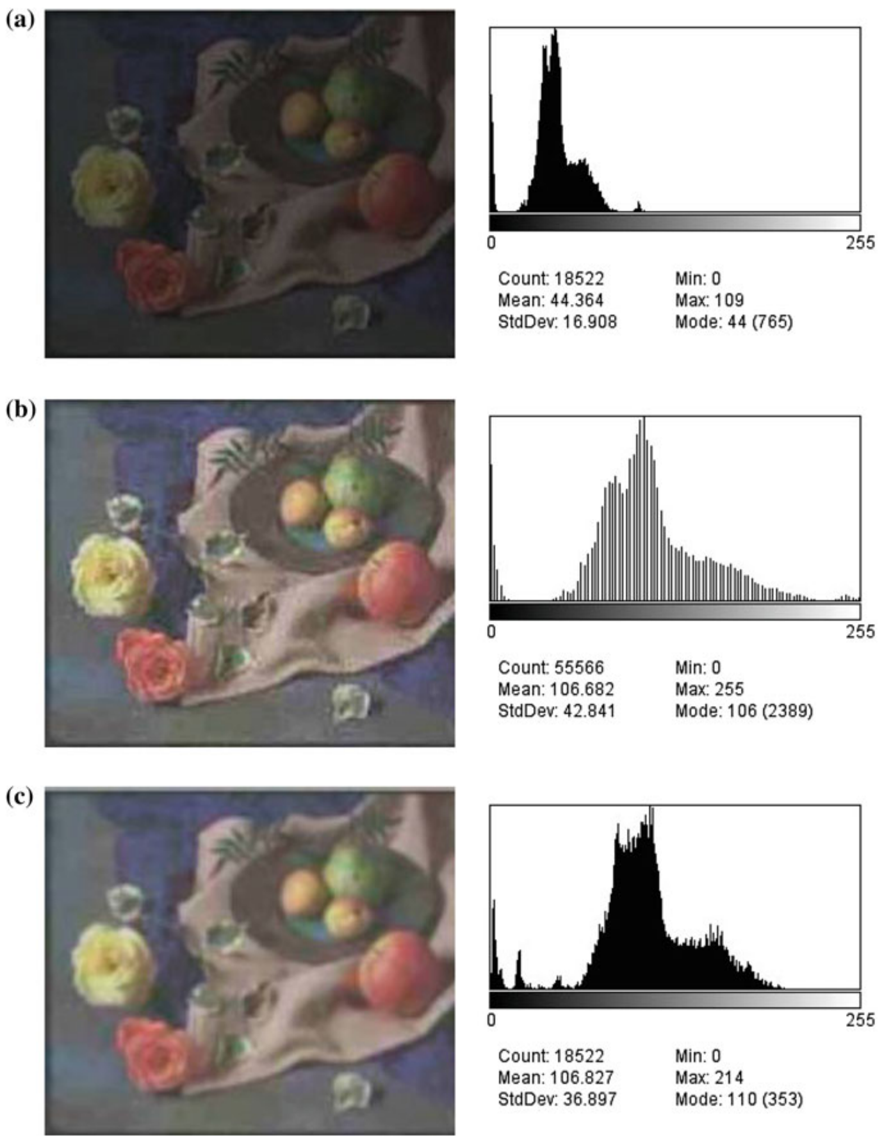


Fig. 2.1 **a** A RGB image and its brightness histogram. **b** The enhanced image and its histogram. **c** The enhanced and interpolated image and its histogram

2.2.1 Gray-Level Transformation

Some methods for contrast enhancement are based on gray-level transformation and histogram modification. These are point operators that are applied to a neighborhood reduced to (1×1) pixel. Hence Eq. (2.1) can be expressed in the form $l' = T(l)$ where l and l' denote the input pixel value and the output pixel value, respectively. Since the mapping $T(\cdot)$ denotes a point operator, it is independent on the pixel coordinates and it is the same for all the image pixels. Hence, each output pixel depends only on the input pixel having the same coordinates. These operators may be expressed by means of lookup tables.

Gray-level transformation operators can be divided into two main classes: linear operators and nonlinear operators. In the following we give some examples of both classes.

Linear Contrast Stretching
This transformation enhances the dynamic range by linearly stretching the original gray-level range $[a, b] \subset [0, 255]$ to the range $[0, 255]$. The transformation is defined as

$$l' = T(l) = 255\frac{(l-a)}{(b-a)} \tag{2.2}$$

where $a \leq l \leq b$.
Generally the linear transformation from the range $[a, b]$ to the range $[a', b']$ is

$$l' = T(l) = (l-a)\frac{(b'-a')}{(b-a)} + a' \tag{2.3}$$

where $a \leq l \leq b$.

Linear Contrast Stretching with Clipping
This transformation is used when $[a, b] \supset [0, 255]$. If the number of levels outside the range $[0, 255]$ is small, these levels are clipped in the following manner: levels $l \leq 0$ are set to 0, levels $l \geq 255$ are set to 255. For all the other levels, the Eq. (2.2) is applied.

Logarithmic Transformation
This transformation is defined as

$$l' = c \log(1 + |l|) \tag{2.4}$$

with $c > 0$. This transformation is used to compress the dynamic range so as to enhance details related to low levels. For example, it is used to visualize the Fourier spectrum of an image [11].

Power-Law Transformation
This transformation is defined as

$$l' = l(c \cdot \exp^{\gamma}) \tag{2.5}$$

where c and γ are positive constant values. By varying the value of γ different functions can be obtained to compress or to expand the dynamic range of gray levels. Conventionally, the exponent in the power-law function is referred to as *gamma*. The power-law transformations are useful to perform gamma correction in the visualization of images on a monitor or generally they are useful for general-purpose contrast manipulation [6].

Sigmoid Transformation
This transformation is defined as:

$$S(x) = 1/(1 + \exp^{-\gamma(x-c)}) \tag{2.6}$$

where the value of c indicates the abscissa of the inflection point of the function and the parameter γ controls the contrast (values greater than 5 results in an enhancement of the contrast). Figure 2.2 shows a plot of the S-function with $c = 0.2$ and $\gamma = 15$. By applying the S-function with different values of γ and c we obtain different contrast enhancement results.

In Fig. 2.3 we show some examples of contrast modification using the plugin given in Listing 2.3 that provides different S-functions.

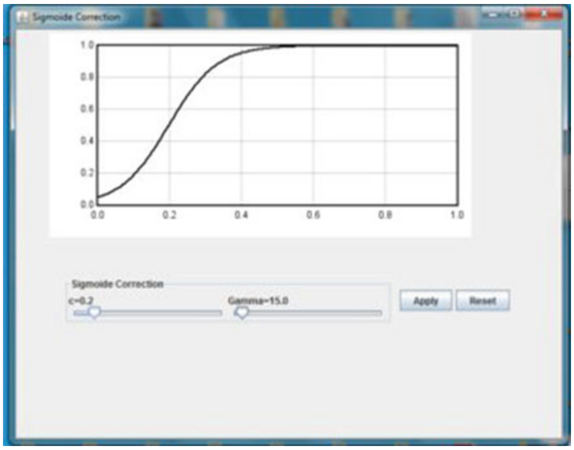
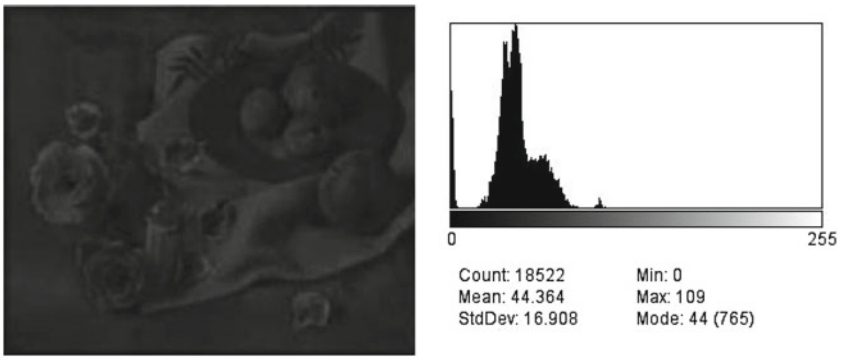
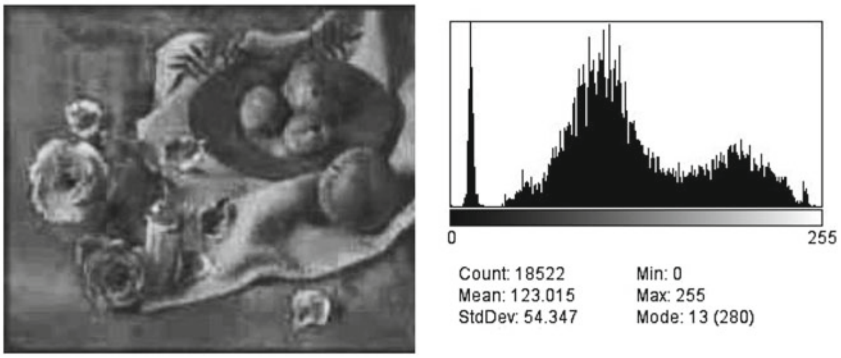


Fig. 2.2 A plot of the S function with $c = 0.2$ and $\gamma = 15$

(a)


(b)


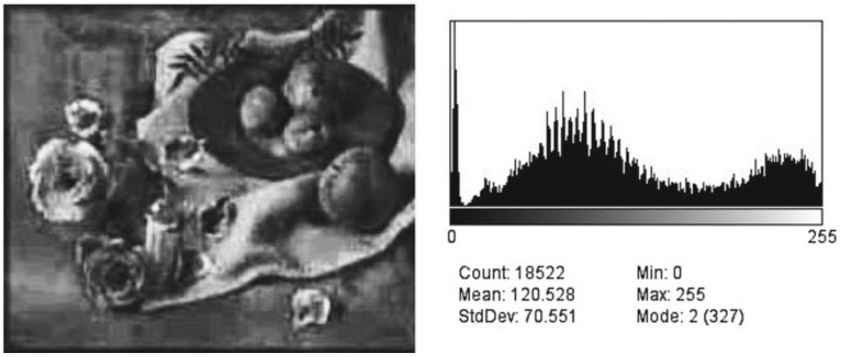
(c)


Fig. 2.3 Some examples of contrast modification using different S-functions. **a** Original image (brightness component of Fig. 2.1 and its histogram). **b** Contrasted image obtained using the S-function with $c = 0.2$ and $\gamma = 15$. **c** Contrasted image obtained by applying the S-function with $c = 0.2$ and $\gamma = 24$

Listing 2.3 S-function.java: Java plugin cor contrast enhancement using the sigmoid function.

```
/**
 * Contrast enhancement by the following sigmoid function:
 * bb = 1/(1+Math.exp(GAMMA*(c-aa))).
 *
 * Different values for the parameters c and GAMMA
 * can be chosen
 *
 * Author: Ignazio Altomare
 * Date: 4/11/2010
 */
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;
import ij.gui.GenericDialog;
import ij.*;
import ij.gui.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import ij.text.*;

public class Sigmoid_Correction extends WindowAdapter
   implements PlugInFilter, ChangeListener, ActionListener {

  private int w;
  private int h;
  private ImagePlus im_sig;
  private ImageProcessor ip_orig,ip_sig;
  private byte[] im;
  private ImageWindow w_sig;

  // variables
  private int K = 256;
  private int aMax = K - 1;
  private float GAMMA_ini=15f;
  private float c_ini=0.5f;
  // window for visualizing the sigmoid function
  private JFrame windowSig;
  private PlotPanel graphicSig;
  // button for applaySig e resetSig
  private JButton applySig;
  private JButton resetSig;
  //labels for C and Gamma
  private JLabel C_label;
  private JLabel Gamma_label;
  //sliderfor the values C and Gamma
  private JSlider C_slider;
  private JSlider Gamma_slider;

  public int setup(String arg, ImagePlus img) {
    return DOES_8G;
  }
  public void run(ImageProcessor ip) {
    w = ip.getWidth();
    h = ip.getHeight();
    ip_orig=ip;

    //create a copy of the image
    im_sig = NewImage.createByteImage("Sigmoid Correction",w,h,1,
        NewImage.FILL_BLACK);
    ip_sig = (im_sig.getProcessor()).convertToByte(true);
    ip_sig.copyBits(ip,0,0,Blitter.COPY);

    //get pixel values
    im = (byte[]) ip_sig.getPixels();

    //process
    this.process();

    //show the sigmoid window
    this.showSig();

    w_sig = new ImageWindow(im_sig);
    w_sig.addWindowListener(this);
    im_sig.updateAndDraw();
  }


  private void process(){
      // create a lookup table for the mapping function
      int[] Fgc = new int[K];
      for (int a = 0; a < K; a++) {
         double aa = (double) a / (double)aMax;   // scale to [0,1]
         double bb = 1/(1+Math.exp(GAMMA_ini*(c_ini-aa)));

          // scale back to [0,255]
          int b = (int) Math.round(bb * aMax);
          Fgc[a] = b;
      }

      ip_sig.applyTable(Fgc);  // modify the image
  }


  private ImagePlus plotSig(){
    float[] x = new float[256];
    float[] y = new float[256];

    for(int i=0; i<256; i++){
      x[i]=(float)i/(float)aMax;
      y[i]=(float)(1/(1+Math.exp(GAMMA_ini*(float)(c_ini-x[i]))));
    }

    Plot p = new Plot("Sigmoid Correction","","",x,y);
    p.setLimits(0.0,1.0,0.0,1.0);
    p.setLineWidth(2);

    return p.getImagePlus();
  }
  private void showSig(){
    //create buttons
    applySig=new JButton("Apply");
    applySig.addActionListener(this);
    resetSig=new JButton("Reset");
    resetSig.addActionListener(this);

    //create panels
    JPanel panelSigmoid=new JPanel(new GridLayout(2,2));
    JPanel panelApply_Reset=new JPanel();
    graphicSig = new PlotPanel(this.plotSig().getImage());


    //set borders of panel
    panelSigmoid.setBorder(BorderFactory.createTitledBorder
        ("Sigmoid Correction"));

    //create labels
    C_label=new JLabel();
    Gamma_label=new JLabel();

    this.setLabelSig();

    //add labels to panel
    panelSigmoid.add(C_label);
    panelSigmoid.add(Gamma_label);

    //create sliders
    C_slider=new JSlider(JSlider.HORIZONTAL);
    Gamma_slider=new JSlider(JSlider.HORIZONTAL);

    this.setSliderSig();

    C_slider.addChangeListener(this);
    Gamma_slider.addChangeListener(this);

    //add slider to panel
    panelSigmoid.add(C_slider);
    panelSigmoid.add(Gamma_slider);

    //add Apply and Reset buttons
    panelApply_Reset.add(applySig);
    panelApply_Reset.add(resetSig);

    //create window for the Sigmoid function
    windowSig = new JFrame("Sigmoid Correction");
    windowSig.setSize(700,550);
    windowSig.setLocation(300,200);
    windowSig.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    windowSig.setLayout(new FlowLayout());
    Container contentPane=windowSig.getContentPane();
    contentPane.add(graphicSig);
    contentPane.add(panelSigmoid);
    contentPane.add(panelApply_Reset);
    windowSig.setVisible(true);
  }

  private void setLabelSig(){
    C_label.setText("c="+c_ini);
    Gamma_label.setText("Gamma="+GAMMA_ini);
  }

  private void setSliderSig(){
    C_slider.setMinimum(1);
    C_slider.setMaximum(10);
    C_slider.setValue((int)(c_ini*10));
    Gamma_slider.setMinimum(1);
    Gamma_slider.setMaximum(255);
    Gamma_slider.setValue((int)GAMMA_ini);
  }

  private void resetSig(){
    c_ini=0.5f;
    GAMMA_ini=15;
    C_slider.setMinimum(1);
    C_slider.setMaximum(10);
    C_slider.setValue((int)(c_ini*10));
    Gamma_slider.setMinimum(1);
    Gamma_slider.setMaximum(255);
    Gamma_slider.setValue((int)GAMMA_ini);
  }

  public void actionPerformed(ActionEvent e){

    Object source=e.getSource();

    if(source==applySig){
      ip_sig.copyBits(ip_orig,0,0,Blitter.COPY);
      im = (byte[]) ip_sig.getPixels();
      this.process();
      im_sig.updateAndDraw();
    }

    if(source==resetSig){
      this.resetSig();
      setSliderSig();
      setLabelSig();
      graphicSig.updateImage(plotSig().getImage());
    }
  }

  public void stateChanged(ChangeEvent e){
    Object source=e.getSource();

    if(source==C_slider){

      c_ini = (float)C_slider.getValue()/(float)10;

      setSliderSig();
      setLabelSig();
      graphicSig.updateImage(plotSig().getImage());
    }


    if(source==Gamma_slider){

      GAMMA_ini = Gamma_slider.getValue();

      setSliderSig();
      setLabelSig();
      graphicSig.updateImage(plotSig().getImage());
    }


  }
  public void windowClosing(WindowEvent e){
   windowSig.setVisible(false);
  }
}
```

2.2.2 Thresholding

Another way to perform contrast stretching is thresholding. A threshold t is defined so that levels $l \leq t$ are set to 0, while levels $l > t$ are set to 255. In this way a binary image is obtained having values {0, 255} or {0, 1} (Fig. 2.4).

2.2.3 Histogram Transformation

Histograms are the basis for numerous spatial domain processing techniques. Histograms are simple to calculate and also lend themselves to economic hardware implementations, thus making them a popular tool for real-time image processing.

The histogram represents a global information for an image: all the pixels having a particular value i contribute to populate the i-th bin of the histogram. That is $h(i) = n_i$ being n_i the number of pixels having value i. The presence of peaks in an histogram may represent bright or dark regions, or regions having low or high contrast. The modification of the histogram produces a different distribution of gray levels. Hence histogram manipulation can be used effectively for image enhancement. The modification of an histogram is defined by means of a point transformation $T : j \rightarrow k$ such that if the level j has frequency $h(j)$ then the transformed level k has frequency $g(k)$ where $h(\cdot)$ and $g(\cdot)$ are the initial histogram and the transformed one, respectively.

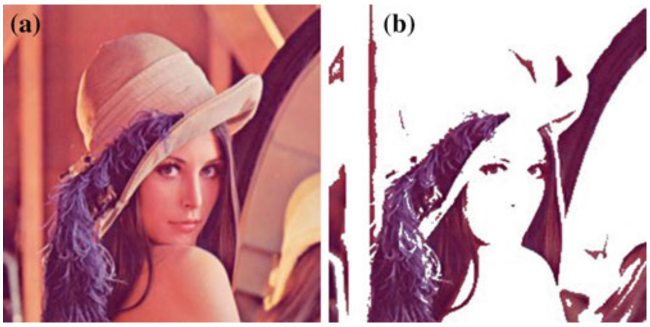


Fig. 2.4 **a** LENA image. **b** Image obtained by thresholding only the brightness channel

Equalization
Equalization or normalization is the transformation of the level distribution into a uniform distribution in which the frequency of each transformed level is approximately constant, i.e., $g(i) \simeq \textit{constant}$ for $i = 0, \ldots, L_{max}$ where L_{max} is the maximum gray level. Since equalization provides an histogram that is almost uniform, it improves the image by augmenting the contrast and removing regions that are too bright or too dark.

Let $f(x, y)$ be a gray-level image of $n = M \times N$ pixels with $L_{max} + 1$ gray levels and let $h(i)$ be its histogram. We define the cumulative histogram of f as

$$h_c(i) = \sum_{j=0}^{i} h(j) = \sum_{j=0}^{i} n_j$$

for $i = 0, \ldots, L_{max}$. If the histogram h is uniform then its cumulative function h_c is a line. Hence the equalization of the histogram h can be computed by imposing the cumulative histogram h_c to be linear. To this aim we consider the following equation:

$$\frac{i}{h_c(i)} = \frac{L_{max}}{n} \qquad i = 0, ..., L_{max}$$

from which we obtain

$$i = L_{max}\frac{h_c(i)}{n} = L_{max}\frac{\sum_{j=0}^{i} h(j)}{n} = L_{max}\frac{\sum_{j=0}^{i} n_j}{n}$$

where n is the number of pixels and n_j is the occurrence number of the level j.

Appendix A provides reference to a Java plugin [5] useful to evaluate the histogram of a color image.

2.3 Image Smoothing

Smoothing, also called blurring, is a simple and frequently image processing operation, used to 'blur' images and remove detail and noise. A blurring process can attenuate the abrupt transitions of the gray levels between a pixel and its neighbor (random noise) or the irrelevant details associated to a small number of pixels.

Generally to perform a smoothing operation we apply a filter to the image, by means of a local operator. A local operator produces a value for each pixel (x, y) of the output image g computed in a neighbor or window w (Fig. 2.5) centered in the pixel (x, y) of the input image f by the following equation:

$$g = T(f, w)$$

In Fig. 2.5 a filter is visualized as a window of coefficients sliding across the image, that is the image is explored in a fixed sequence (for example, from left to

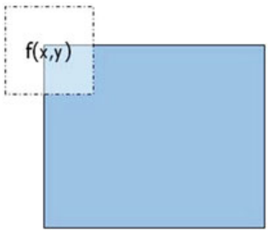


Fig. 2.5 Window or mask centered in the pixel $f(x, y)$

right and from top to down). The function T can be linear or not linear. The most common types of filter are linear: the output value $g(x, y)$ is determined as a weighted sum of input pixel values $f(x + i, y + j)$. Local operators based on the convolution operation can be formally described by means of the theory of linear systems, namely the Fourier transform and the convolution product theorem.

Linear image smoothing is a local operator based on a convolution of the image with a matrix h of proper dimension $L \times L$, called *mask* or *kernel*, where L is an odd value. More formally given an image $f(x, y)$ of $M \times N$ pixels and $h(x, y)$ a $L \times L$ spatial mask, we define $l = \lfloor L/2 \rfloor$ and the following equation describes the convolution product in the spatial domain between the image f and the mask h, with origin in the center of the mask.

$$g = h \otimes f$$

$$g(x, y) = \sum_{i=-l}^{l} \sum_{j=-l}^{l} h(i, j) f(x + i, y + j) \text{ for } x = 0, ..., M - 1, y = 0, ..., N - 1$$

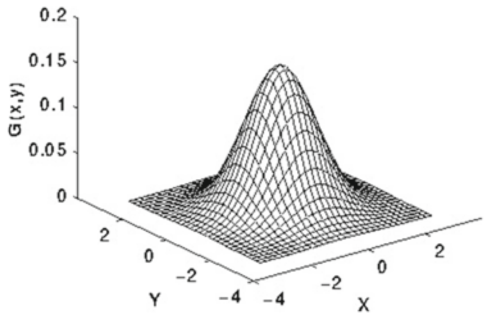
There are many kind of filters depending on the mask used in the convolution equation. In the following we will mention the most used.

Mean Filter

A simple process for image smoothing consists in locally computing the mean value for each pixel. This can be obtained by means of the convolution of the input image with the mask in Fig. 2.6 (lowpass spatial filtering). The mask is a square matrix of coefficients used to compute a new value starting from the neighbor of the examined pixel. It is also called filter in the spatial domain. The multiplying factor is needed to normalize the weights to 1. In this way the range of the output results the same as the input. The effect of the convolution operator is to compute each output pixel as mean value of the pixels in its $L \times L$ neighborhood.

Fig. 2.6 3×3 mean filter

$$h = \frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$



$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Fig. 2.7 A two-dimensional Gaussian function with $\sigma = 1$ and the corresponding 5×5 mask

Gaussian Filter

The Gaussian smoothing operator is a two-dimensional convolution operator that is used to blur images and remove details and noise. It uses kernels having the shape of a two-dimensional Gaussian function:

$$G(x, y, \sigma) = \left[\frac{1}{2\pi\sigma^2}\right] \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

where σ is the standard deviation and $r = x^2 + y^2$ is the ray from the center. The degree of smoothing is determined by the value of σ. Larger values of σ require larger convolution kernels in order to be accurately represented. Figure 2.7 shows a two-dimensional Gaussian function and the corresponding mask.

About 99.7 of values drawn from a Gaussian function are within three standard deviations (σ away from the mean). This fact is known as the three-sigma rule. For this reason, the Gaussian smoothing eliminates the influence of those points that are away from 3σ with respect to the current pixel in the mono-dimensional case and $6\sqrt{2}\sigma$ in the bi-dimensional case (the central lobe of the two-dimensional Gaussian function has the value $2\sqrt{2}\sigma$). A Java plugin for the two-dimensional Gaussian filter is available at [7]. As an example, Fig. 2.8 shows the effect of the Gaussian smoothing on the Lena image.

2.4 Edge Detection

Edges represent abrupt changes or discontinuities in an amplitude attribute of an image such as luminance, surface orientation, color and so on. Edges characterize object boundaries and are usually defined as curves separating two regions having different average values of their characteristics. The causes of the region dissimilarity may be due to some factors such as the geometry of the scene, radiometric characteristics of the surface, illumination, and so on. If the regions are sufficiently

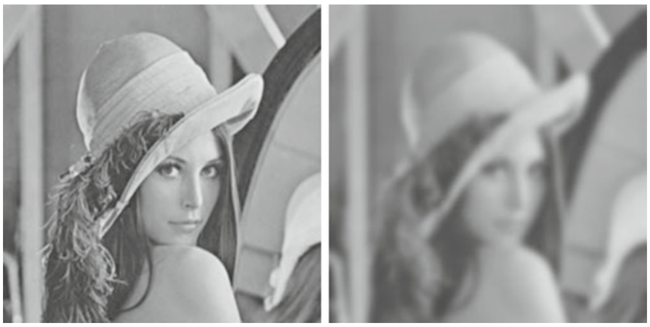


Fig. 2.8 The Lena image and the result of Gaussian smoothing with $\sigma = 3$

homogeneous, the transition between two adjacent regions may be detected by analyzing the discontinuities along gray levels.

Edge detection is a fundamental problem in image analysis and computer vision. It is the process to locate and identify sharp discontinuities in an image giving boundaries between different regions. This boundary detection is the first step in many computer vision edge-based tasks such as face recognition, obstacle detection, target recognition, image compression, and so on. Edge detection is a local operator based on a convolution of the image with a matrix h of dimension $L \times L$, called *mask* or *kernel*, where L is an odd value.

An edge is characterized by the following features:

- *Edge normal*: the unit vector in the direction of maximum intensity change.
- *Edge direction*: the unit vector along the edge (perpendicular to the edge normal).
- *Edge position or center*: the image position at which the edge is located.
- *Edge strength or magnitude*: local image contrast along the normal. Generally a pixel is an edge pixel if its strength overcomes a predefined threshold value.

An edge detection method detects the pixels candidate to be points of the boundary of an object or a region. To derive a boundary of an object all the edge pixels of that boundary should be grouped. This can be done by *border following* algorithms or *grouping* algorithms.

Since edges may not be represented by perfect discontinuities, the quality of detected edges is highly dependent on noise, lighting conditions, objects of same intensities, and density of edges in the scene. Regarding this problem it should be noted that even though noise is not visible in the original image, noise is highlighted in derivatives, especially in second derivatives. Hence edge detection, being based on derivatives, is highly affected by noise. Some noise effect can be reduced by thresholding. For example, we could define a point in an image as an edge point if its first derivative is greater than a specified threshold. By doing so, we automatically assess which significant gray-level transitions can be considered as edge. Another problem arises when the edge is located on a soft discontinuity. A solution to this problem is proposed in Chap. 8.

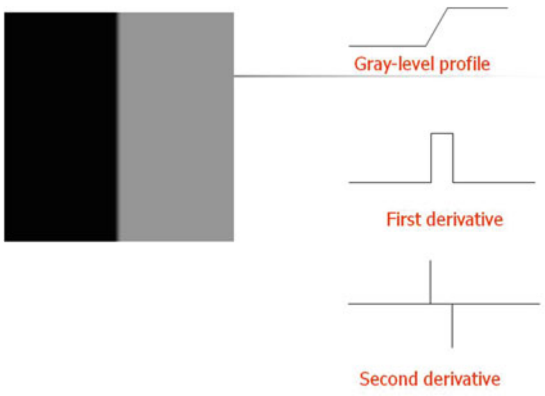


Fig. 2.9 Edge profile and derivatives

The most common edge detection methods are the Gradient operator based on first derivatives, the Laplacian and the LoG (Laplacian of a Gaussian) operators based on second derivatives. Figure 2.9 shows how the magnitude of the first derivative can be used to detect the presence of an edge in an image. The sign of the second derivative can be used to determine whether an edge pixel lies on the dark or the light side of an edge. The zero crossings of the second derivative provide a powerful way of locating edges in an image.

Generally an edge detection method involves three steps

1. Smoothing to reduce the noise;
2. Applying edge enhancement, that is a local operation that extracts all image pixels candidate to be edge points;
3. Applying edge localization (thresholding) to select the edge points among all the candidate points.

Steps 1. and 2. can be implemented by convolving the input image with a proper mask so as to obtain the gradient image. In the third step the edge points are detected, for example by looking for maximum and minimum magnitude values for the first derivative operators. These operators analyze the distribution of the gradient values in the neighborhood of a given pixel and determine if the pixel has to be classified as an edge point on the basis of threshold values. The results of these edge detectors are very sensitive to the threshold value. These operators require high computational time and hence cannot be used in real-time applications.

Gradient Operator

Given an image $f(x, y)$, its gradient is defined by

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

The magnitude of the gradient is given by

$$m(x, y) = |\nabla f(x, y)| = \left(\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2\right)^{\frac{1}{2}}$$

The direction of the gradient is given by

$$\alpha(x, y) = \tan^{-1}\left(\frac{\partial f(x, y)}{\partial x} \Big/ \frac{\partial f(x, y)}{\partial y}\right)$$

The Gradient operator may be implemented as a convolution with the masks shown in Figs. 2.10 and 2.11.

Laplacian Operator

In many applications it is of particular interest to construct derivative operators, which are isotropic, i.e., rotation invariant. This means that rotating the image f and applying the operator gives the same result as applying the operator on f and then rotating the result. In other words, if the operator is isotropic then equally sharpened edges are enhanced in any direction. One of these isotropic operators is the Laplacian operator, defined as

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial^2 x} + \frac{\partial^2 f(x, y)}{\partial^2 y}$$

This can be implemented using the mask of Fig. 2.12. If $f(x, y)$ is not constant or it does not vary linearly then the Laplacian of f has a zero crossing, i.e., a sign change crossing the x axis.

Laplacian of a Gaussian

Using second-order derivatives, the edge localization step is based on the extraction of zero-crossing points which indicate a sign change crossing the x-axis. Since the second-order derivative is very sensitive to noise, a filtering function is required. In [8] a Gaussian function is used to smooth the image hence deriving the operator called Laplacian of a Gaussian (LoG). The Gaussian smoothing operator is a

Fig. 2.10 Sobel masks to detect vertical edges

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

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

Fig. 2.11 Sobel masks to detect horizontal edges

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

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

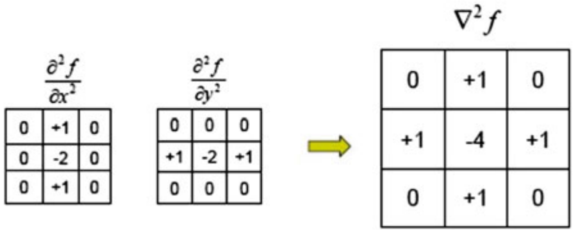


Fig. 2.12 Laplacian mask

two-dimensional convolution operator that is used to blur images and remove details and noise. It uses kernels that represent the shape of a two-dimensional Gaussian function

$$G(x, y, \sigma) = \left[\frac{1}{2\pi\sigma^2}\right] \exp\left(\frac{-x^2 + y^2}{2\sigma^2}\right)$$

The LoG operator based on this Gaussian function is defined as

$$LoG(x, y, \sigma) = c\left[\frac{(x^2 + y^2)}{\sigma^2} - 1\right] \exp\left(\frac{-x^2 + y^2}{2\sigma^2}\right)$$

where c is a factor that normalizes to 1 and the value of σ determines for each pixel (x, y) the number of points that influence the evaluation of the Laplacian in (x, y). A significant problem of the LoG operator is that the localization of edges with an asymmetric profile by zero-crossing points introduces a bias which increases with the smoothing effect of filtering [1].

2.4.1 *Canny Operator*

An interesting solution to avoid the dependence of detected edges on noise was proposed by J. Canny in [2], who defined an optimal operator for edge detection including three criteria: good detection, good localization, identification of single edge point (a given edge in the image should be marked only once).

Let $f(x, y)$ a gray-level image of $M \times N$ pixels and $G(x, y)$ a Gaussian filter. The Canny operator performs the following steps:

1. First the noise is filtered out from the image. A suitable Gaussian filter is used for this task. Gaussian smoothing can be performed using a convolution product $f_s(x, y) = g(x, y) \otimes f(x, y)$. Some parameters have to be fixed for this operator, such as the standard deviation σ of the Gaussian filter. The width of the mask must be chosen carefully since it is directly proportional to the localization error. Since the Gaussian smoothing eliminates the influence of the points far more than $3(2\sqrt{2}\sigma)$ with respect to the current pixel, the mask size must be equal to $6\sqrt{2}\sigma$ for a fixed value of σ.

2. The second step consists of computing the gradient of f by means of the Sobel masks along x (columns) and y (rows) directions. Edge strength is found out by taking the gradient of the image.
3. The third step finds the edge direction using the gradient in x and y directions. For each pixel (x, y) we evaluate the gradient strength $m(x, y)$ and the gradient direction $\alpha(x, y)$, where m and α are matrices having the same size of the image $f(x, y)$. A non-maxima suppression algorithm is applied that follows the edge direction and suppresses any pixel value that is not to be considered as edge point. That is, for each pixel (x, y) we consider the gradient direction $\alpha(x, y)$ and check if $m(x, y)$ has a local maximum in that direction. Usually a small number of directions is considered. For example, the four directions $(0°, 90°, 45°, -45°)$ of a (3×3) window centered in the pixel (x, y) may be considered to produce an initial edge map $S(x, y)$.
4. The last step uses double thresholding to eliminate false edges. Two thresholds $t_1 < t_2$ are selected, with a ratio of 2 or 3. Pixels of $edge(x, y)$ having a gradient magnitude $m(x, y)$ greater than t_2 are definitively labeled as edge pixels. If a point (x, y) has $m(x, y) < t_2$ and is also connected to points yet labeled as edge points then $m(x, y)$ is compared with t_1. If $m(x, y) > t_1$ then the point (x, y) is definitively labeled as an edge point. All the other points are not labeled as edge points.

An *Imagej* plugin that implements the Canny algorithm can be found at [4]. An application example is shown in Fig. 2.13. We can observe how an appropriate choice of the parameters of the Canny operator may produce very thin edges.

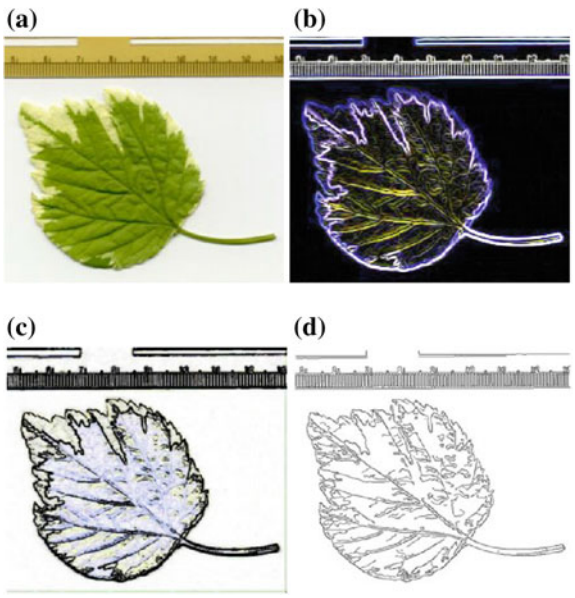


Fig. 2.13 **a** Original image. **b** Image obtained by applying the gradient in (x, y) directions. **c** Image after thresholding of the brightness of the image **b**. **d** Image obtained from **a** after applying Canny operator with $\sigma = 1$, $t_1 = 2.5$, $t_2 = 7.5$

2.4.2 Optimization-Based Operators

In the previous sections, we have seen that detection of edges usually involves two stages. The first one is an edge enhancement process that requires the evaluation of derivatives of the image making use of gradient or Laplacian operators. Methods such as thresholding or zero crossing produce an edge map that contains pixels candidates to be labeled as edge points. In the second stage, pixels of the edge map are selected and combined in contours using processes such as boundary detection, edge linking, and grouping of local edges [11, 13].

This last phase can be viewed as a search of the optimal configuration of those pixels that better approximate the edges. More precisely let us consider an image $F = \{f(x, y); 0 \leq x \leq M - 1, 0 \leq y \leq N - 1\}$ and an edge configuration $S = \{s(x, y); 0 \leq x \leq M - 1, 0 \leq y \leq N - 1\}$ where $s(x, y) = 1$ if (x, y) is an edge pixel, $s(x, y) = 0$ otherwise. Therefore an edge could be considered as one of the possible paths in the universe of the pixels of the image F. If we define a function $T(S)$ evaluating the edge S, the searching of the best edge configuration can be accomplished by means of an optimization method that minimizes/maximizes the function $T(S)$.

In other words, the edge detection problem can be formulated as one of optimization where the evaluation function depends on the local edge structure. Since the search space for the optimal solution is extremely large due to the number of possible configurations of the $M \times N$ pixels of the image, a 'blind' search would be fully inefficient. Then optimization methods are necessary which take into account the geometric and topological constraints of the problem. In this sense some methods have been introduced such as graph searching, relaxation, and simulated annealing [9, 10, 12].

In [3] optimization techniques known as Genetic Algorithms are proposed for the search of the optimal edge. The peculiarities of these algorithms are the robustness in the application to different classes of problems and the natural parallel implementation. When using a genetic algorithm for optimization, a solution is encoded as a string of genes to form a chromosome representing an individual. In [3] an individual is an edge configuration S represented by a string of $M \times N$ bits. Each bit encodes the presence (or not) of an edge pixel in the image F. The approach consists essentially of two phases: evaluation of the likelihood of a pixel to be an edge pixel and boundary detection by means of genetic algorithms. An objective function T is supplied which assigns a fitness value to each edge configuration S. This function evaluates the cost of S as the sum of the costs of each pixel (x, y) in S. The assumptions are that the edges should tend to be continuous, thin and of sufficient length; moreover the edges should be perpendicular to the gradient at each pixel. The cost function T evaluates at each point the deviation from the previous assumptions by computing a linear combination of five weighted factors: fragmentation, thickness, local length, region similarity, and curvature. These factors capture the local nature of the edges and are evaluated in a $(w \times w)$ window centered on each pixel (x, y) using the values of the configuration S and a likelihood map L based on the gradient (amplitude and direction). The pixels in this window constitute the neighbor of the central pixel. The genetic algorithm, starting from an initial population (i.e., a collection of possible solutions) iteratively produces new generations of individuals (i.e., potential solutions) using the operators of reproduction, crossover, and mutation. Since the problem is the minimization of the objective function $T(S)$, each individual S of the population must reproduce itself in proportion to the inverse of its function $T(S)$. The iterative optimization process ends when the mean value of the objective function T does not change, within a tolerance value, between two consecutive generations.

References

1. Bhardwaja, S., Mittalb, A.: A survey on various edge detector techniques. Procedia Technol. **4**, 220–226 (2012)
2. Canny, J.F.: A computational approach to edge detection. IEEE Trans. PAMI **8**(6), 679–698 (1986)
3. Caponetti, L., Abbattista, N., Carapella, G.: A genetic approach to edge detection. In: Proceedings of the IEEE International Conference on Image Processing, 1994. ICIP-94, pp. 318–322. IEEE, New York (1994)
4. Gibara, T.: Canny edge detector. ImageJ plugin available at: http://rsbweb.nih.gov/ij/plugins/canny/index.html
5. Gibara, T.: Color histogram. ImageJ plugin available at: http://rsb.info.nih.gov/ij/plugins/color-histogram.html
6. Huang, S.C., Cheng, F.C., Chiu, Y.S.: Efficient contrast enhancement using adaptive gamma correction with weighting distribution. IEEE Trans. Image Process. **22**(3), 1032–1041 (2013)
7. Lieng, E.: 2D Gaussian filter. Java plugin available at: https://imagej.nih.gov/ij/plugins/gaussian-filter.html
8. Marr, Y.D., Hildreth, E.: Theory of edge detection. Proc. R. Soc. Lond. B: Biol. Sci. **207**(1167), 187–217 (1980)
9. Martelli, A.: An application of heuristic search methods to edge and contour detection. Commun. ACM **19**(2), 73–83 (1976)
10. Mumford, D., Shah, J.: Boundary detection by minimizing functionals. In: Proceedings of the IEEE Computer Vision Pattern Recognition (San Francisco), pp. 22–26 (1985)
11. Rosenfeld, A., Kak, A.C.: Digital Picture Processing. Academic Press, Cambridge (1982)
12. Tan, H.L., Gelfand, S.B., Delp, E.J.: A cost minimization approach to edge detection using simulated annealing. IEEE Trans. PAMI **14**(1), 3–18 (1991)
13. Torre, V., Poggio, T.A.: On edge delection. IEEE Trans. PAMI **8**(2), 147–163 (1986)

Chapter 3
Basics of Fuzzy Logic

Of all things that are certain, the most certain is doubt.
Bertolt Brecht

Abstract In this chapter, foundations of fuzzy logic are presented to introduce the necessary notations used throughout the following chapters. The chapter provides basic notions of fuzzy set theory and fuzzy systems, such as fuzzification, fuzzy rule base and inference engine, defuzzification, and fuzzy models.

3.1 Introduction

Fuzzy logic [31] offers a problem-solving tool that is between the precision of classical logic and the inherent imprecision of the real world. In the last years, several approaches based on fuzzy logic have been introduced to process image data having characteristics of vagueness and ambiguity due to the acquisition phase and also to imprecise and ill-defined knowledge about the image contents. The imprecision in an image contained in the pixels can be handled using fuzzy sets that are the primary elements of fuzzy logic [32]. Vague concepts like 'good contrast' or 'sharp boundaries,' 'light red,' 'high saturation,' etc., can be perceived qualitatively by the human reasoning and expressed in a formal way by means of fuzzy logic, that enables a machine to simulate human reasoning.

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_3

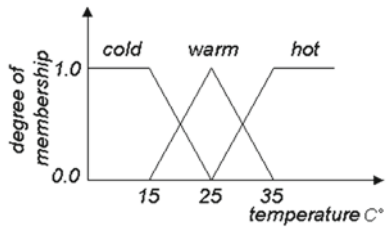


Fig. 3.1 Example of fuzzy sets defined for the variable *temperature*

3.2 Fuzzy Set Theory

Fuzzy set theory, introduced by Zadeh [31], is a mathematical tool for translating abstract concepts found in natural language into computable entities. Such entities are called fuzzy sets. Fuzzy sets represent vague descriptions or properties of objects, i.e., *tall*, *small*, *cold*, *bright*, etc. The fuzzy set theory offers a way to deal with imprecise and vague information, so as to simulate human reasoning and perception. Fuzzy logic concerns the concept of partial truth; the truth values vary from 0.0—fully false to 1.0—fully true. Then a fuzzy set A is defined as a set whose membership function μ_A has values in the range [0.0, 1.0]. Values $\mu_A(x) = 0.0$ and $\mu_A(x) = 1.0$ stand, respectively, for null and full membership of x to A, whereas all values $\mu_A(x)$ between 0.0 and 1.0 indicate partial membership of x to A. Mathematically, a fuzzy set A is represented by a membership function defined on a domain X, called universe of discourse,[1] given by

$$\mu_A : X \to [0, 1]$$

where A is the fuzzy label or linguistic (value) term describing the variable x. As an extension to boolean logic, $\mu_A(x)$ represents the grade of membership of x belonging to the fuzzy set A. Fuzzy sets can be used to define the values of *fuzzy variables*.

Consider the fuzzy variable *temperature*, which can be described by many different adjectives (linguistic terms) each with its own fuzzy set. A typical partition of the universe of discourse, $0 - 40\,°\mathrm{C}$, is shown in Fig. 3.1, where the fuzzy sets cold, warm, and hot are defined. Here, the crisp temperature 15 °C has a grade of membership of 0.5 for both the cold and the warm fuzzy sets, i.e., $cold(15\,°\mathrm{C}) = warm(15\,°\mathrm{C}) = 0.5$. The power of fuzzy variables is that they facilitate gradual transitions between attributes and, consequently, possess a natural capability to express and deal with uncertainties.

It is clear that the definition of fuzzy sets is nonunique for the nature of language, but it is very context-dependent and user specific (e.g., this definition may seem inappropriate to an Eskimo!). On specifying a membership function $\mu_A(x)$ in its present context the vague fuzzy label A is precisely defined. Hence fuzzy sets can be thought as measuring the inherent vagueness of language precisely. The properties

[1]Fuzzy sets can be defined in either discrete or continuous universes. Universes found in real-world applications are typically continuous, but in the area of digital image processing discrete universes are more often considered.

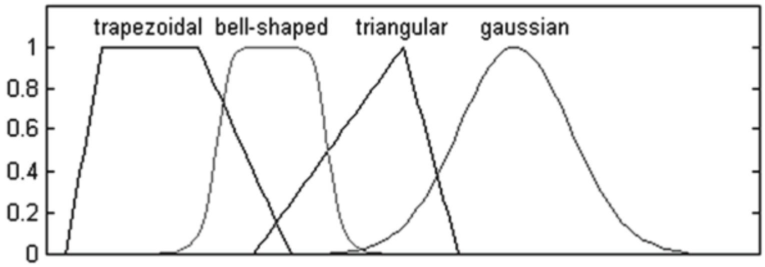


Fig. 3.2 Principal types of membership functions

of these fuzzy sets play an important role in the modeling capabilities of the fuzzy system, and for a model to be truly transparent these sets should sensibly represent terms that describe the input and output variables. It is up to the system designer to determine the shape of the fuzzy sets. In most cases, however, the semantics captured by fuzzy sets is not too sensitive to variations in the shape; hence it is convenient to use simple membership functions. The most common membership functions are the triangular, the trapezoidal, and the Gaussian function, depicted in Fig. 3.2. As such, fuzzy sets constitute a powerful approach not only to deal with incomplete, noisy, or imprecise data but also to develop models of the data that provide smarter and smoother performance than traditional modeling techniques.

The popularity and practicality of fuzzy systems derives from their ability to express complex relations in terms of linguistic rules. Thus, fuzzy systems have advantages of excellent capabilities to describe a given input–output mapping. The second important property to be considered concerns function approximation: fuzzy systems have been proved to be universal approximators [6, 7, 30], i.e., they are able to uniformly approximate continuous functions to any degree of accuracy on closed and bounded (compact) sets, a property they share with feedforward neural networks. Moreover, in contrast with other universal approximators (e.g., neural networks) fuzzy systems are uniquely suited to incorporate linguistic information in a natural and systematic way. Since Zadeh's first pioneering paper [31] fuzzy systems have been successfully used to build models that can explain complex processes, such as in biology, chemistry, material science, or in economy, growing into various industry applications.

3.3 Fuzzy Rule-Based Systems

The working scheme of a fuzzy system is based on a particular inference process where the involved variables are modeled by means of fuzzy sets.

Using fuzzy sets, a fuzzy system can represent the imprecision characterizing real-world problems using IF-THEN rules expressed in natural language. A collection of such rules called the *knowledge base* is used to describe the I/O mapping being

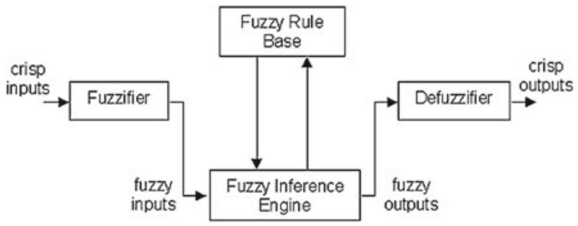


Fig. 3.3 General scheme of a fuzzy inference system

approximated, and a proper inference process is applied. Hence a fuzzy system, also called fuzzy rule-based system, approximates an unknown I/O mapping by inference from a set of fuzzy rules that are humanly understandable statements such as

R1: IF *temperature* is *cold* THEN set output of the heater *high*
R2: IF *temperature* is *warm* THEN set output of the heater *zero*

describing a typical relationship between room temperature and the desired output of a heater. Such rules use linguistic terms such as 'cold' and 'warm,' that express vague concepts defined over the input variable 'temperature.' Such vague concepts are modeled as fuzzy sets.

The basic structure of a fuzzy system, as described by Mamdani and Assillan [19], is shown in Fig. 3.3. A fuzzy system processes crisp data at the input and produces crisp data at the output through inference from a fuzzy rule base, which represents the knowledge base of the system. Therefore a fuzzier is used at the front of the system to convert crisp data to fuzzy sets, and a defuzzifier is used at the output of the system to convert fuzzy sets into crisp values.

The fuzzy inference engine combines the rules in the rule base according to approximate reasoning theory to produce a mapping from fuzzy sets in the input space to fuzzy sets in the output space. Hence a fuzzy system provides a computational scheme describing how rules must be evaluated and combined to compute a crisp output value (vector) for any input crisp value. One can therefore think of a fuzzy system simply as a parameterized function that maps real vectors to real vectors.

According to the form of fuzzy rules, different models can be employed to define a fuzzy rule-based system. In a Mamdani-type fuzzy system, the consequence of each rule is a fuzzy set defined for a linguistic variable. The inference of the rules follows the *modus ponens* model, extended to fuzzy sets. Namely, given an input value, the inference of a rule consists in scaling down the consequence fuzzy set by the firing strength of the rule defined by the membership degree of the input value to the premise. The definition of the THEN operator, also called implication operator, gives the output fuzzy set for that rule.

Due to the use of fuzzy sets in the premise of rules, several fuzzy rules can be fired at the same time. For example, with the fuzzy sets defined in Fig. 3.1, both the two rules R1 and R2 are fired by input temperature 13 °C, hence the output command for heating is calculated by combining the fuzzy sets resulting from inference of both rules R1 and R2. Usually, to perform rule aggregation, i.e., to combine partial results provided by individual rules, the operators MAX or SUM are used.

However, for most control applications crisp values are needed as output, hence qualitative information in form of fuzzy sets is not suitable for the output variable. A defuzzification phase is therefore needed to obtain an output crisp value from a fuzzy set. There are several defuzzification methods, some are based on the centroid of the outcomes and others on the maximum values given by the membership functions.

3.3.1 Fuzzification

Fuzzification is the process that converts crisp inputs to fuzzy sets defined on the input space. Generally the component of the system performing this process is called *fuzzifier*. In this step, a fuzzification function is introduced for each input variable to express the associated measurement uncertainty. The purpose of the fuzzification function is to interpret measurements of input variables, each expressed by a real number, as more realistic fuzzy approximations of the respective real numbers. The fuzzier models uncertainties found on the input and hence it smoothes out the system response, making it less sensitive to specific input values and hence to uncertainty on the input, such as noise.

In many cases input variables are not fuzzified. That is, the singleton fuzzification is applied, which assumes no noise on the crisp inputs, hence measurements of input variables are employed in the inference process directly. The singleton fuzzification maps a crisp input value x_0 to a fuzzy singleton, i.e., to a fuzzy set such that its support is reduced to x_0. This type of fuzzification requires low computational cost since the calculation of the system's output is simplified. However singleton fuzzification may not always be adequate, especially in cases where input data are corrupted by noise. To account for uncertainty in the data non-singleton fuzzification is necessary [20]. In such cases, the fuzzification function has the form

$$f_e : [-a, a] \rightarrow X$$

where X denotes the set of all fuzzy sets and $f_e(x_0)$ is a fuzzy approximation of the measurement x_0.

3.3.2 Fuzzy Rule Base and Inference Engine

For a fuzzy system with n inputs and one output, the rule base is composed of a set of K fuzzy rules formally defined as

$$R_k : \text{IF } (x_1 \text{ is } A_{1k}) \text{ AND } \cdots \text{ AND } (x_n \text{ is } A_{nk}) \text{ THEN } (y \text{ is } B_k) \tag{3.1}$$

for $k = 1, \ldots, K$, where $A_{ik}, i = 1, \ldots n$ are fuzzy sets defined on the input variables and B_k is a fuzzy set defined on the output variables. The antecedent of a rule can be seen as multidimensional fuzzy set $\mathbf{A}_k$ obtained as *intersection* of univariate fuzzy sets A_{ik}, $i = 1, \ldots, n$. Hence the basic form of a fuzzy rule in (3.1) can be also written as

$$R_k : \text{IF } (\mathbf{x} \text{ is } \mathbf{A}_k) \text{ THEN } (y \text{ is } B_k) \tag{3.2}$$

Using the fuzzy implication operator each rule maps the antecedent fuzzy set $\mathbf{A}_k$ into the consequent fuzzy set B_k. Each rule R_k can then be viewed as a fuzzy implication $\mathbf{A}_k \to B_k$ characterized by a continuous multivariate membership function

$$\mu_{R_k}(\mathbf{x}, y) = \mu_{\mathbf{A}_k \to B_k}(\mathbf{x}, y) = \tau(\mu_{\mathbf{A}_k}(\mathbf{x}), \mu_{B_k}(y))$$

where τ is a T-norm operator.

Given a fuzzified input $\mathbf{A}_0 = (A_{10}, \ldots, A_{n0})$, the inference engine uses the fuzzy rule base to infer a fuzzy output B_0 by means of the compositional rule of inference. The compositional rule can be applied locally to each rule R_k and the resulting fuzzy sets are aggregated to provide the inferred fuzzy set. Specifically, the inference engine first composes $\mathbf{A}_0$ (the fuzzified input) with each rule R_k producing as intermediate result the fuzzy set $B_{k0} = \mathbf{A}_0 \circ R_k$ with membership function

$$\mu_{B_{k0}}(\mathbf{y}) = \sup_{\mathbf{x}}[\tau(\mu_{\mathbf{A}_0}(\mathbf{x}), \mu_{R_k}(\mathbf{x}, y))]$$

Then, a fuzzy output is inferred as the *union* of all individual fuzzy outputs, that is $B_0 = \bigcup_{k=1}^{K} B_{k0}$, being K the number of fuzzy rules in the rule base.

3.3.3 Defuzzification

Frequently, the output of a fuzzy rule-based system is required to be a crisp value, an essential requirement in many engineering problems, for example fuzzy control applications. In these cases, a defuzzification stage is needed to obtain a crisp output from the fuzzy output resulting from the inference of rules.

This stage is performed by the defuzzifier which maps the output fuzzy set to a single crisp point in the output space. There are many different defuzzification techniques, most of which are described in [4, 14, 15, 29]. The most successful method is the *center of gravity* or *center of area* where the crisp value for a generic output variable y is found as

$$y_0 = \frac{\int_y \mu_{B_0}(y) y dy}{\int_y \mu_{B_0}(y) dy}$$

Typically, to reduce computational costs, a discrete representation of the above formula is used

$$y_0 = \frac{\sum_{q=1}^{N_q} \mu_{B_0}(y_q) y_q}{\sum_{q=1}^{N_q} \mu_{B_0}(y_q)}$$

where N_q is the number of quantization steps by which the universe of discourse Y is discretized. There are also a number of computationally simplified defuzzification methods that combine rule aggregation and defuzzification into a single phase. One of them is the weighted average method, which computes the defuzzified output as

$$y_0 = \frac{\sum_{k=1}^{K} \mu_k(\mathbf{x}_0) b_k}{\sum_{k=1}^{K} \mu_k(\mathbf{x}_0)} \tag{3.3}$$

where K is the number of rules, $\mu_k(\mathbf{x}_0)$ is the degree of activation of the k-th rule, and b_k is a numerical value associated with the consequent B_k of the k-th rule.

3.4 Fuzzy Models

Depending on the particular form of the consequent proposition in fuzzy rules, two categories of fuzzy models can be distinguished, which recall the contrast between two different goals of modeling: readability and performance [1, 2].

Mamdani Fuzzy Models.
The first vision of fuzzy models, and by far the most innovating one, assumes to represent an input/output mapping by means of a collections of IF-THEN rules whose antecedents and consequences use fuzzy values. This is the basic form of fuzzy rule given in (3.2). The use of fuzzy sets in consequent part makes these models very intuitive and understandable. This class of fuzzy models uses fuzzy reasoning [24] and forms the basis for qualitative modeling, which describes an input–output mapping by using a natural language [27]. The Mamdani model [18] falls into this category. When adopting this perspective, which pursuits the ultimate goal of fuzzy logic, i.e., ‘computing with words,’ the emphasis is put essentially on the readability of the model, rather than on computational cost and accuracy of the model (i.e., fine quality of approximation, classification or control). However, fuzzy models of this class tend to become complex, requiring too many parameters, hence they can become heavy to run, maintain, and tune.

Takagi–Sugeno Fuzzy Models
The second category of fuzzy models is based on the Takagi–Sugeno–Kang (TSK) method [28]. These models use fuzzy rules with fuzzy antecedents and functional consequent parts, thereby qualifying them as mixed fuzzy or non-fuzzy models. Such models can represent a general class of static or dynamic nonlinear mappings via a combination of several linear models. The whole input space is decomposed into several partial fuzzy spaces and each output space is represented with a linear

equation. The resulting models are referred to as TS models and are represented by a series of fuzzy rules of the form

$$R_k : \text{IF } (\mathbf{x} \text{ is } \mathbf{A}_k) \text{ THEN } (y = h_k(\mathbf{x})) \tag{3.4}$$

where $h_k(\mathbf{x})$ is a polynomial function of the inputs and represents a local model used to approximate the response of the system in the region of the input space represented by the antecedent $\mathbf{A}_k$. This type of knowledge representation does not allow the output variable to be described in linguistic terms, which is one of the drawbacks of this approach. Hence, this class of fuzzy models should be used when only accuracy (not interpretability) is the ultimate goal of predictive modeling.

Singleton Fuzzy Models
Each of these fuzzy models has inherent drawbacks. For Mamdani fuzzy models, the defuzzification process may be time consuming, and systematic fine-tuning of the parameters is not easy. For TS fuzzy models it is hard to assign appropriate linguistic terms to the rule consequence part, which does not use fuzzy values. Readability and performance thus appear as antagonist objectives in fuzzy rule-based systems. Some form of compromise can be found by using simplified fuzzy rules of the form

$$R_k : \text{IF } (\mathbf{x} \text{ is } \mathbf{A}_k) \text{ THEN } (y \text{ is } b_k) \tag{3.5}$$

where b_k is a fuzzy singleton, i.e., a fuzzy set reduced to a single element. Fuzzy models relying on such rules are referred to as *singleton fuzzy models*. This class of fuzzy models can employ all the other types of fuzzy reasoning mechanisms because they represent a special case of each of the above-described fuzzy models. More specifically, the consequent part of this simplified fuzzy rule can be seen either as a singleton fuzzy set in the Mamdani model, or as a constant output function in TS models. Thus the two fuzzy models are unified under this simplified fuzzy model.

3.4.1 Design of Fuzzy Rule-Based Systems

The main advantage of using a fuzzy rule-based system is the possibility to express the human knowledge in form of linguistically interpretable IF-THEN rules. Despite no standard technique is available to transform the human knowledge into a set of fuzzy rules and membership functions, there are some fundamental steps that are usually performed. The first step is to identify and label the input and output variables. Then their value ranges should be specified and a fuzzy partition of each input and output should be defined. The final step is the construction of the rule base and the specification of the membership functions characterizing the fuzzy sets. In Sect. 5.5, we give an example of how to define a fuzzy rule-based system using Java.

In the last years, many tools have been developed to enable easy design of a fuzzy rule-based system by reducing specific expertise. Some tools are currently available as open source software, mainly based on Java. Some of existing packages and libraries, such as *DotFuzzy* [3], *FRBS* [25], *Funzy* [9], *JFuzzinator* [23], *libFuzzyEngine* [16], *nxtfuzzylogic* [22] enable the design of a fuzzy system for specific purposes. Such specific tools, even if they are simple and easy to use, usually have limited functionality. For example, they include only one membership function (typically trapezoid) and/or one defuzzification method. Among general purpose tools, both *JFuzzyLogic* [8] (developed in Java) and *FisPRO* [10, 11] (developed in C++ and Java) provide an interactive environment for designing and optimizing fuzzy inference systems.

In general, tools with graphical interface (such as *FisPRO*) are designed to include the following main components:

1. *User work section*: it allows the user to access the primary functions of the tool to introduce configuration information as language and working directory.
2. *Fuzzy set editor*: it allows the user to define and set the parameters of the fuzzy sets. This includes input and output definition and fuzzy partitioning specification. First, the user can define input and output variables and for each variable the domain name and the range of possible values. Then, for each domain, it is possible to define a number of fuzzy sets by associating a membership function. Different membership functions can be defined, namely triangular, Gaussian, trapezoidal, sigmoid, and singleton. This component allows the user to delete any of the created fuzzy sets. Moreover the user can also select a defuzzification method.
3. *Fuzzy rule editor*: it enables a fuzzy rule definition by selecting the input fuzzy sets to be included in the antecedent of each rule, together with the fuzzy sets to be included in the consequent of each rule. Also, the user can choose the fuzzy operator to be used for implementing the AND connective in the antecedent of rules (e.g., the MIN or the product operator). Also, other fuzzy operators (OR, NOT) adopted in the compositional rule of inference can be defined. Using this component the user can easily define all the necessary rules, so as to create the knowledge base of the fuzzy system.
4. *Rule Inference Engine*: given a fuzzified input, this component applies the compositional rule of inference to derive a fuzzy output. This component is the core of the fuzzy rule-based system, and usually it is not visible to the user. It performs also the defuzzification step, by applying the defuzzification operator selected by the user among different available defuzzification methods.
5. *Fuzzy rule testing*: this component allows the user to test any single rule created in the rule base on a sample input vector.

All these steps are often difficult to be performed manually, especially when defining a fuzzy rule-based system for a task of image processing. Indeed, the main advantage of fuzzy systems is that they can provide simple intuitive for interpretation and prediction in the form of fuzzy rules. However, due to vagueness and subjectivity of natural language statements, fuzzy rules based on qualitative knowledge alone can adequately model only very simple processes. Besides, for complex ill-defined processes it usually takes a lot of time to define and tune the parameters which quantitatively define linguistic labels. Finally, expert knowledge in the form of linguistic statements is unavailable or limited in real-world contexts. This restricts the application of fuzzy systems to fields where expert knowledge is available and the complexity of the process to be modeled is limited.

To overcome these inherent limitations of fuzzy systems due to the lack of enough expert knowledge, fuzzy rules should be learned from examples. Learning in fuzzy systems is most often implemented by learning techniques derived from neural networks [12], leading to the development of neuro-fuzzy modeling techniques, as explained in the following.

3.4.2 Neuro-Fuzzy Models

Neuro-fuzzy modeling substantially reduces development time and cost of creating a fuzzy rule base while improving the accuracy of the resulting fuzzy model. Being able to utilize a neural learning algorithm implies that a fuzzy system with linguistic information in its rule base can be created or adapted using numerical information. This gains an even greater advantage over a neural network that cannot make use of linguistic information and behaves as a black box model. Hence, the behavior of a neuro-fuzzy system can either be represented by a set of humanly understandable fuzzy rules or by a combination of localized basis functions associated with activation functions of the artificial neurons.

Integration of fuzzy logic and neural networks has led to the development of several neuro-fuzzy models with consolidated presence in scientific literature [2, 4, 14, 17, 21].

Here we introduce a general scheme of neuro-fuzzy network, deeply analyzed in [5]. Let us consider a MIMO (Multi-Input Multi-Output) fuzzy model with n inputs and m outputs. The rule base is composed of a set of fuzzy rules formally defined as

$$R_k : \text{IF } \underbrace{(x_1 \text{is} A_{1k})\text{AND} \cdots \text{AND}(x_n \text{is} A_{nk})}_{\text{antecedent}} \text{ THEN } \underbrace{(y_1 \text{is} B_{1k})\text{AND} \cdots \text{AND}(y_m \text{is} B_{mk})}_{\text{consequent}}$$

where A_{ik}, $i = 1, \ldots n$ and B_{jk}, $j = 1, \ldots, m$ are fuzzy sets defined on the input and output variables, respectively. Once we put the components of the fuzzy system in a parametric form, the fuzzy inference system becomes a parametric model which can be tuned by a learning procedure. This idea is the basis of neuro-fuzzy modeling. When algebraic operators are used to implement the fuzzy logical functions, crisp inputs are fuzzified using singleton fuzzification and center of gravity defuzzification is employed, the input–output mapping of a Mamdani fuzzy system can be represented by a linear combination of the (normalized) input fuzzy membership functions

$$y = \sum_{k=1}^{K} \left\{ \frac{\mu_{A_k}(\mathbf{x})}{\sum_{h=1}^{K} \mu_{A_h}(\mathbf{x})} \right\} b_k \tag{3.6}$$

where K is the number of rules and b_k are the centers of fuzzy sets B_k. Also the output of a Takagi–Sugeno fuzzy system, under some mild conditions, can be regarded as a linear combination of the normalized multivariate input membership functions. In fact, if the TS fuzzy system uses the weighted mean criterion (3.3) to combine the local representations, the output for a generic input $\mathbf{x}$ is computed as the normalized sum

$$y = \frac{\sum_{k=1}^{K} \mu_k(\mathbf{x}) h_k(\mathbf{x})}{\sum_{k=1}^{K} \mu_k(\mathbf{x})} \tag{3.7}$$

The basic assumption of the neuro-fuzzy integration relies on the fact that, at the computational level, a fuzzy inference system can be seen as a layered architecture, similar to a multilayer feedforward neural network.

To show the correspondence between a fuzzy system and a neural network, here we consider a singleton fuzzy system based on rules of the form (3.5), extended to m outputs

$$\text{IF } (x_1 \text{ is } A_{1k}) \text{ AND } \cdots \ (x_n \text{ is } A_{nk}) \text{ THEN } (y_1 \text{ is } b_{1k}) \text{ AND } \cdots \ (y_m \text{ is } b_{mk})$$

for $k = 1 \dots K$, where K is the number of fuzzy rules, $A_{ik} (i = 1 \dots n)$ are fuzzy sets defined over the input variables x_i, and $b_{jk} (j = 1 \dots m)$ are fuzzy singletons defined over the output variables y_j. Membership functions of fuzzy sets A_{ik} should be chosen to be everywhere differentiable, a useful property for gradient descent learning techniques. For example the Gaussian functions can be employed

$$\mu_{ik}(x_i) = \exp\left(- \frac{(x_i - c_{ik})^2}{2\sigma_{ik}^2} \right) \tag{3.8}$$

where c_{ik} and σ_{ik} are the center and the width of the Gaussian function, respectively. The output values of the singleton fuzzy systems are obtained by rule inference as

$$y = \frac{\sum_{k=1}^{K} \mu_k(\mathbf{x}) b_k}{\sum_{k=1}^{K} \mu_k(\mathbf{x})} \tag{3.9}$$

This fuzzy system can be visualized as being composed of: rules, fuzzy sets used in these rules, and fuzzy inference operations (intersection, aggregation, defuzzification, etc.). Since the specifications of the fuzzy system are fixed in terms of fuzzy inference operations, the only part that can be defined through learning are the premise and consequent parameters associated with rules. The problem can be stated in a precise way as that of finding a proper configuration of membership functions and generating a set of K fuzzy rules from a data set $\mathcal{D}_N$ of N input–output pairs such that the fuzzy system closely approximates the unknown function underlying the data. To do this, the fuzzy system is implemented using a four-layer neural network with four layers having following characteristics.

1. Layer 1 provides the crisp input values $(x_1, \dots, x_n)$ to the network. Nodes in this layer do not perform any calculation and simply take the input values and pass them to the second layer.
2. Layer 2 realizes a fuzzification of the input variables. Units in this layer are organized into K distinctive groups, being K the number of rules. Each group is associated with one of the fuzzy rules, and it is composed of n units, corresponding to the fuzzy sets in the fuzzy rule. The i-th unit in the k-th group, connected with the i-th node in layer 1, evaluates the Gaussian membership degree of the input x_i to the fuzzy set A_{ik} according to (3.8).
3. Layer 3 is composed of K units. The k-th unit performs the precondition matching of rule R_k and computes its fulfillment degree by means of the product operator

$$\mu_{A_k}(\mathbf{x}) = \prod_{i=1}^{n} \mu_{A_{ik}}(x_i)$$

4. Layer 4 supplies the final output vector $\mathbf{y}$ and is composed of m units. The j-th unit in this layer computes the output value y_j, according to (3.9). In particular, the fulfillment degrees of the rules are weighted by the fuzzy singletons b_k, which are encoded as the values of the connections weights between layers 3 and 4.

Figure 3.4 depicts the structure of the above-described neuro-fuzzy network.

Once a fuzzy rule-based system is modeled as a neural network, learning algorithms can be used to learn the parameters c_{ik}, σ_{ik} and b_k of the fuzzy rules. Many neuro-fuzzy systems use direct nonlinear optimization to identify all the parameters of a fuzzy system. The most widely used is an extension of the well-known backpropagation algorithm [26] implemented by gradient descent. A very large number of neuro-fuzzy systems are based on backpropagation-like algorithms, starting from the most famous ANFIS network [13].

The advantage of neuro-fuzzy modeling is twofold. On one side, model identification can be performed using both empirical data and qualitative knowledge. On the other side the resulting models are transparent, significantly aiding model validation and knowledge discovery.

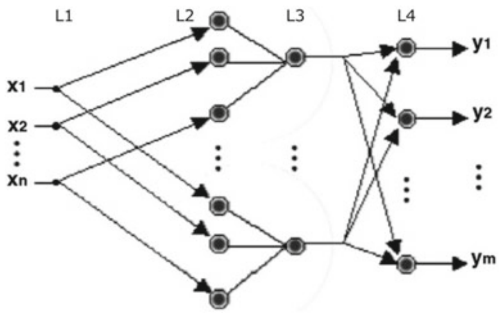


Fig. 3.4 General scheme of a neuro-fuzzy network

References

1. Babuska, R.: Fuzzy modeling and Identication. Ph.D. thesis, Technische Universiteit Delft (1996)
2. Bersini, H., Bontempi, G.: Now comes the time to defuzzify neurofuzzy models. Fuzzy Sets Syst. **90**, 161–169 (1997)
3. Bertoli, M.: DotFuzzy. https://github.com/MicheleBertoli/DotFuzzy
4. Brown, M., Harris, C.J.: Neurofuzzy Adaptive Modelling and Control. Prentice Hall, Hemel Hempstead (1994)
5. Castellano, G.: A neurofuzzy methodology for predictive modeling. Ph.D. thesis, University of Bari (2000)
6. Castro, J.: Fuzzy Logic Controllers are Universal Approximators. IEEE Trans. Syst., Man Cybern. **25**(4), 629–635 (1995)
7. Castro, J., Delgado, M.: Fuzzy systems with defuzzication are universal approximators. IEEE Trans. Syst., Man Cybern. **26**, 149–152 (1996)
8. Cingolani, P., Alcal-Fdez, J.: jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. Int. J. Comput. Intell. Syst. **6**, 6175 (2013)
9. Funzy.: Having fun with fuzzy logic. https://code.google.com/p/funzy/
10. Guillaume, S., Charnomordic, B.: Fuzzy inference systems: an integrated modeling environment for collaboration between expert knowledge and data using FisPro. Expert Syst. Appl. **39**(10), 8744–8755 (2012)
11. Guillaume, S., Charnomordic, B., Labl, J-L.: FisPro (Fuzzy inference system professional). https://www7.inra.fr/mia/M/fispro/
12. Haykin, S.: Neural Networks: A Comprehensive Foundation. MacMillun College Publishing Company, New York (1994)
13. Jang, J-S.R.: ANFIS: Adaptive-network-based fuzzy inference system. IEEE Trans. Syst., Man Cybern. **23**(3), 665–685 (1995)
14. Jang, J.-S.R., Sun, C.-T.: Neuro-fuzzy modelling and control. Proc. IEEE **83**, 378–406 (1995)
15. Lee, C.C.: Fuzzy logic in control systems: Fuzzy logic controller - part I and II. IEEE Trans. Syst., Man Cybern. **20**(2), 404–435 (1990)
16. LibFuzzyEngine++. http://sourceforge.net/projects/libfuzzyengine/
17. Lin, C., Lee, C.: Neural Fuzzy Systems: A Neural Fuzzy Synergism to Intelligent Systems. Prentice-Hall, Englewood Cliffs (1996)
18. Mamdani, E.H.: Advances in the linguistic synthesis of fuzzy controllers. Int. J. Man-Mach. Stud. **8**, 669–678 (1976)
19. Mamdani, E.H., Assillan, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man-Mach. Stud. **7**(1), 1–13 (1975)
20. Mouzouris, G.C., Mendel, M.J.: Dynamic non-singleton fuzzy logic systems for nonlinear modeling. IEEE Trans. Fuzzy Syst. **5**(2), 199–208 (1997)
21. Nauck, D.: Neuro-fuzzy systems: review and prospects. In: Proceedings of the Fifth European Congress on Intelligent Techniques and Soft Computing (EUFIT97), pp. 10441053 (1997)
22. NXTfuzzylogic. www.openhub.net/p/nxtfuzzylogic
23. Omran, H.: JFuzzinator. http://sourceforge.net/projects/jfuzzinator/
24. Pedrycz, W.: Fuzzy Control and Fuzzy Systems. Wiley, New York (1989)
25. Riza, L.S., Bergmeir, C., Herrera, F., Benitez, J.M.: FRBS - Fuzzy rule-based systems. http://dicits.ugr.es/software/FRBS/
26. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
27. Sugeno, M., Yasukawa, T.: A fuzzy-logic-based approach to qualitative modeling. IEEE Trans. Fuzzy Syst. **1**, 7–31 (1993)
28. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its application to modeling and control. IEEE Trans. Syst., Man Cybern. **15**, 116–132 (1985)

29. Wang, L.: Adaptive Fuzzy Systems and Control. Prentice Hall, Englewood Clis (1994)
30. Wang, L., Mendel, J.M.: Fuzzy basis functions, universal approximation, and orthogonal least squares. IEEE Trans. Neural Netw. **3**(5), 807–814 (1992)
31. Zadeh, L.A.: Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans. Syst. Man Cybern. **SMC-3** 28–44 (1973)
32. Zimmermann, H.J.: Fuzzy Set Theory and its Applications. Kluwer, Norwell (1992)

Chapter 4
Fuzzy Image Processing

Imagination is more important than knowledge
Albert Einstein

Abstract The use of fuzzy logic for image processing has led to the development of a wide range of techniques casting in the area of Fuzzy image processing. Fuzzy image processing consists of all those approaches that understand, represent, and process an image, its segments and/or its features as fuzzy sets. This chapter covers some basic concepts of fuzzy image processing, namely image fuzzification, image defuzzification and fuzziness measures. The chapter shows also that an image can be considered as an array of fuzzy sets having a membership function that denotes the degree of some image properties satisfied by the image pixels.

4.1 Introduction

Difficulties in the field of image processing are often connected with the typical uncertainty embedded in the data, with many forms of ambiguity and vagueness [8]. Typically vagueness is due to grayness ambiguity in low level processing and it could appear in an image in the form of imprecise boundaries or poor color contrast. Other forms of vagueness appear in an image such as geometrical fuzziness in case of medium level processing tasks (e.g., segmentation) and imprecise and ill-defined knowledge in case of high level processing tasks (e.g., scene analysis and image understanding). Some examples of problems that could be naturally addressed by fuzzy methods are as follows:

- The question whether a pixel is more or less bright in a problem of contrast enhancement.

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_4

- The question whether a pixel is on the edge among two regions in a problem of image segmentation.
- The question of what is a face in a problem of scene analysis and image understanding.

The imprecision in an image can be handled by modeling the image as a fuzzy set. Then vague concepts like "dark", "high contrast", or "sharp boundaries" can be perceived qualitatively by the human reasoning and expressed in a formal way by means of fuzzy logic. Also many difficulties in some tasks of image processing depend on the fact that data/results are uncertain. This uncertainty does not only depend on the data randomness, but also on their ambiguity and vagueness. Indeed randomness could be a matter of the probability theory, while ambiguity and vagueness are a matter of the fuzzy set theory.

A fuzzy image processing task [4] consists of three fundamental steps

1. image fuzzification Φ
2. modification of membership values Γ
3. image defuzzification Ψ

The result Y for an input X is given by

$$Y = \Psi(\Gamma(\Phi(X)))$$

The fuzzification and the defuzzification steps are motivated from the fact that we do not have any fuzzy image, namely any hardware device able to produce fuzzy images. Therefore the coding of image data and decoding of the results are necessary steps to use fuzzy techniques in image processing. The strength of fuzzy image processing is in the modification of membership values, depending on the particular application. As explained in Chap. 3 membership functions represent the basic concept of the fuzzy set theory. Membership values represent the degree of specific properties satisfied by objects. The fuzzy membership functions represent similarities among objects belonging to a set, according to properties defined in an imprecise manner. The membership values also indicate how much fuzzy a set is. The computation of membership values is fundamental in fuzzy image processing. Traditionally image data are processed in the gray-level plane, specifying the values of the gray brightness of each pixel. Instead the aim of the fuzzy image processing is to map the gray-level plane onto the membership plane by means of image fuzzification. In this way, it is possible to operate at an approximate level closer to human reasoning and perception. Different examples will be provided in the application part of this volume.

The most important theoretical contributions in the field of fuzzy image processing are

- Fuzzy inference systems, involving the three steps of image fuzzification, inference, image defuzzification.
- Measures of fuzziness, useful to evaluate how fuzzy an image is. These are useful in many applications such as thresholding.

- Fuzzy rule-based systems, often used in image contrast enhancement.
- Fuzzy clustering, widely used in image segmentation.
- Fuzzy mathematical morphology.

The rest of this chapter deals with more general contributions, namely fuzzy inference systems and measures of fuzziness. The other contributions such as fuzzy rule-based systems, fuzzy clustering, and fuzzy mathematical morphology will be addressed in the second part of this book.

4.2 Image Fuzzification

Typically we model an image by a set of fuzzy singletons with no modification of the pixel values. To illustrate how to introduce a fuzzy set related to an image, we present a simple example: we want to define the set Dark of gray levels that share the property to be dark. In a standard manner (also called *crisp*), we can define a threshold t—for example $t = 100$ and then we can use the following definition:

- Levels $0 \leq g \leq 100$ are elements of the set Dark
- All the other levels do not belong to the set Dark

An alternative approach consists of modeling the darkness property by introducing a fuzzy set defined by a suitable membership function. To this aim, we define two parameters for example 50 and 100 as in Fig. 4.1 and use the following definition:

- Levels $g \leq 50$ are fully elements of the set Dark
- Levels $g \geq 150$ do not belong to the set Dark
- Levels $50 < g < 150$ have a partial degree of belonging to the set

Another example is shown in Fig. 4.2 where three fuzzy sets—Low, Medium and High—are associated to the input image describing the property brightness useful in some contrast enhancement problems (see Chap. 6).

Let us consider an image G of $M \times N$ pixels and L gray levels. To interpret G in a fuzzy way, we assume that G is associated to an array F of fuzzy singleton having membership values $\mu(G(m, n))$. Namely $\mu_{mn} = \mu(G(m, n))$ denotes the membership value of each pixel (m, n) with respect to a predefined property (e.g.,

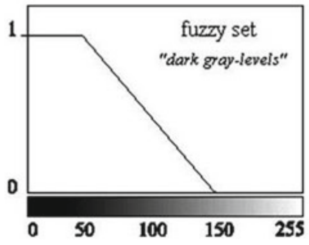


Fig. 4.1 A possible definition of the fuzzy set "Dark gray levels" (taken from [2])

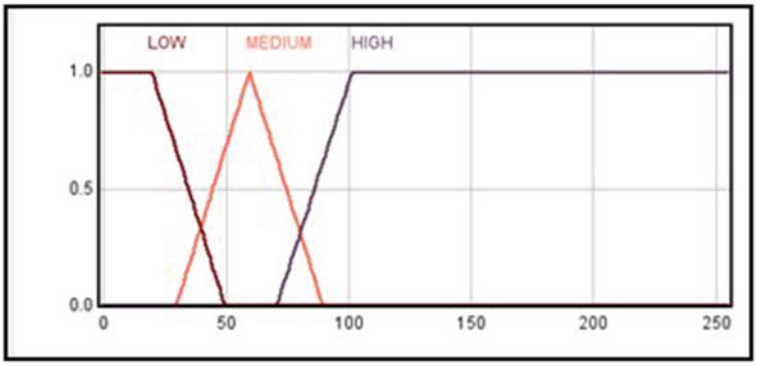


Fig. 4.2 Fuzzy set—Low Medium High Brightness

brightness, edginess or texture), i.e. the degree to which the pixel (m, n) satisfies a property. Hence the image G can be defined as

$$G = \bigcup_m \bigcup_n \mu_{mn}$$

with $\mu_{mn} \in [0, 1]$. The property associated to μ depends on the problem at hand and can be defined by means of global, local, or punctual information.

In other cases, such as image morphology, the gray levels of an image are properly scaled to range in the real interval [0.0, 1.0]. In this case, we transform the pixel values in order to obtain a real-valued image also called fuzzy image. In this way, we can regard the gray level of a pixel as a membership degree in the set of high-valued pixels representing a specific property; thus a gray-level image can be modeled as a fuzzy set.

4.2.1 *Fuzzy Image*

The difference between fuzzy image processing and other image processing approaches is that the input data—histograms, gray levels, features, etc.,—are being processed in the so-called membership plane. In this plane, methods of fuzzy logic can be used to modify the membership values in order to classify data or to come to a decision by means of the fuzzy inference process.

In order to transform the original image in a so-called fuzzy image a suitable membership function, also called fuzzifier, is used. Since the application of a membership function modifies the values of the pixels, it introduces some preprocessing on the image such as contrast modification. Different membership functions can be used such as the N-function, the sigmoid S-function and the bell-shaped function. If no contrast modification is required, it could be convenient to use the very simple

N-function that performs just a normalization, that is a scaling of the image intensity values in the interval [0, 1]. The N-function is a linear transformation with clipping. Starting from Eq. (2.3) it can be defined as a transformation from the range $[a, b]$ to the range [0, 1] as follows:

$$N(x) = \begin{cases} \frac{(x-a)}{(b-a)} = \frac{(x-b+w)}{w} = 1 - \frac{(b-x)}{w} & \text{if } (b-w) \leq x \leq b \\ 1 & \text{if } x > b\,; \\ 0 & \text{if } x < (b-w) \end{cases} \tag{4.1}$$

where b is the maximum intensity value in the image and $w = b - a$ defines the bandwidth of the value range.

Other fuzzifiers often used in image processing are the single-sigmoid S-function and the double-sigmoid symmetric S-function. Generally, the single-sigmoid S-function is used to select intensity bands while the S-function allows to enhance high-intensity bands; both of them correspond to contrast intensifiers. The S-function is defined as follows:

$$S(x) = 1/(1 + \exp^{-\gamma(x-c})) \tag{4.2}$$

where the value of parameter c indicates the abscissa of the inflection point of the function and the parameter γ controls the contrast (values greater than 5 results in an enhancement of the contrast). Figure 4.3 shows a plot of the S-function, obtained using the plugin listed in Listing 2.3. By applying the S-function with $c = 0.5$ and different values of γ, we obtain different contrast enhancement results.

In Fig. 4.4 we show some examples of contrast modification using different membership functions. Figure 4.4a shows the original image and its histogram. Figure 4.4b shows the contrasted image obtained by using the N-function and its histogram. Figure 4.4c shows the contrasted image obtained by applying the S-function depicted in Fig. 4.3 to each RGB channel. Figure 4.4d shows the contrasted image obtained by applying the S-function only to the brightness component in the HSB space.

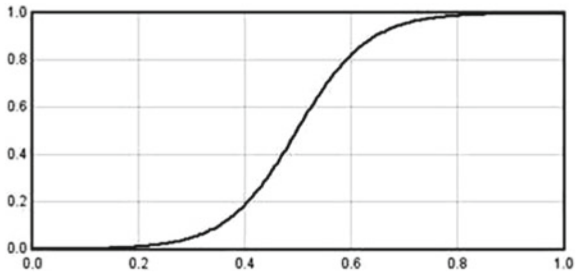


Fig. 4.3 A plot of the S-function

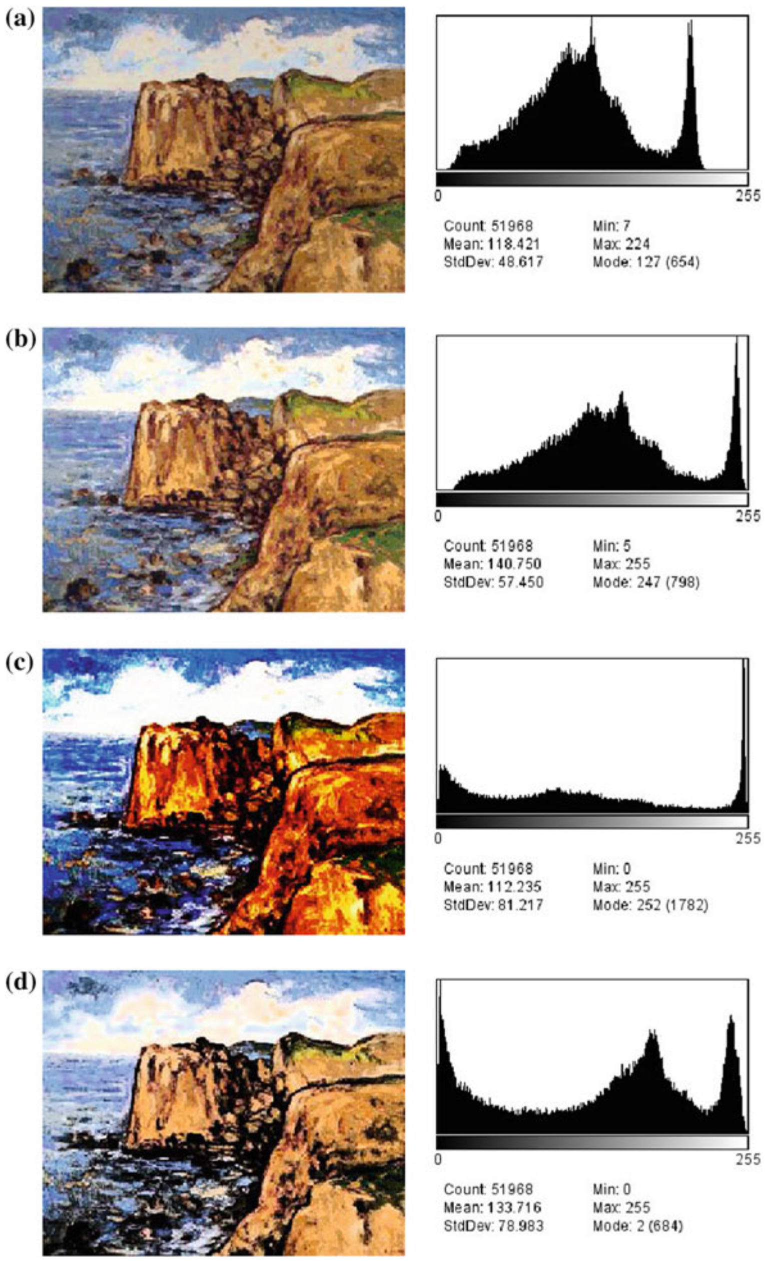


Fig. 4.4 Some examples of contrast modification using different membership functions

4.3 Image Defuzzification

Fuzziness is an intrinsic feature of images and a natural outcome of many image processing tasks. However, in many cases a crisp representation is needed. Typically a crisp representation is required to facilitate easier visualization and interpretation. Even though it contains less information, a crisp representation is usually easier to interpret and understand, especially if the spatial dimensionality of the image is higher than two. Moreover, many tools available for the analysis of binary images are still not developed for fuzzy images, which may force a further step in the fuzzy image processing so that it outputs a crisp representation of the image. The process of replacing a fuzzy representation by a crisp representation is referred to as *defuzzification*.

Defuzzification is a process that maps a fuzzy set to a crisp set. Namely, the goal of defuzzification is generating a good crisp representative of a fuzzy set or recovering a crisp original set. From an application point of view the following features are important for defuzzification:

- Continuity of the defuzzification result: this means that small changes in membership values of the output fuzzy set should not give large changes in the resulting crisp set.
- Computational efficiency: it depends mostly on the kind and the number of operations required for obtaining the result of defuzzification.
- Compatibility to the other operations used in a fuzzy system, like inference and composition may be important.

Most of the literature mentioning defuzzification considers defuzzification of a fuzzy set to a single (crisp) point (see Sect. 3.3.3). Main approaches [4, 6] for defuzzification to a point are

- *Maxima methods* give as a result an element from a fuzzy set core.[1] The main advantage of selecting an element from the core of a fuzzy set is the simplicity. The basic representative of that group is the First-of-Maxima (FOM) technique.
- *Distribution methods* consider the membership function of the output fuzzy set as a distribution, whose average value is evaluated. The output of these methods has continuous and smooth changes in correspondence of changes in the input values. The basic technique of this group is the Center-of Gravity (COG) technique. It converts the membership function into a probability distribution and computes the expected value. Main advantage is the continuity property.
- *Area methods* use area under the membership function to determine the defuzzified output value. The defuzzified value is the (*center of area*) that divides the area under the membership function in two (more or less) equal parts. The Center-of-Area (COA) method is the main technique in this group.

[1]A fuzzy set core (designated as core) consists of elements of a universe of discourse that belong to the set with the highest degree of membership.

Defuzzification of a fuzzy set to a crisp set, when applied to image processing, has not been well explored so far. The most commonly used defuzzification method in image processing consists in thresholding the membership function by selecting a specific α-cut. Of course the result of defuzzification by thresholding is very sensitive to the specific choice of the threshold α. Moreover, defuzzification by thresholding does not take into account spatial properties of the fuzzy set neither it preserves the geometry, shape, and topology of the original object. Since spatial properties are of high importance in image processing applications, generally it is necessary to consider information about global geometrical properties of the object to be defuzzified, in addition to the membership values of the set. Moreover spatial fuzzy sets may be useful as information preserving representations of objects in images, and defuzzification of a spatial fuzzy set can be seen as a crisp segmentation procedure.

In [7] a method of image defuzzification of fuzzy spatial sets based on feature distance minimization has been proposed as an alternative to crisp segmentation. This defuzzification method defines an optimal defuzzification as the one that best preserves (selected) features of the original fuzzy digital object. The correspondence between a fuzzy and a crisp set is established through a distance between their representations based on selected features, where the different resolutions of the images are taken into account. The distance measure is based on the Minkowski distance between feature representations of the sets. The distance minimization provides preservation of the selected quantitative features of the fuzzy set. This method utilizes the information contained in the fuzzy representation for defining a mapping from the collection of fuzzy sets to the collection of crisp sets. If the fuzzy set is a representation of an unknown crisp original set, such that the selected features of the original set are preserved in the fuzzy representation, then the defuzzified set may be seen as an approximate reconstruction of the crisp original set.

4.4 Fuzziness Measures

If we consider an image as a fuzzy set it is useful to introduce some measures to indicate how fuzzy the image is. Such measures are called fuzziness index. Several types of fuzziness index have been proposed. The most common ones are listed below.

Linear index. The fuzziness is evaluated as a difference between membership values and their complement, using the following equation [3]:

$$\gamma_l = \frac{2}{MN} \sum_m \sum_n \min(\mu_{mn}, 1 - \mu_{mn}) \tag{4.3}$$

or

$$\gamma_l = \frac{2}{MN} \sum_{g=0}^{L-1} h(g) \min(\mu_g, 1 - \mu_g) \tag{4.4}$$

where $h(g)$ is the image histogram and $\mu_g = \mu_{mn} = \mu(G(m,n))$ for $g = G(m,n)$. According to this linear index of fuzziness, an image is more or less fuzzy according to increasing or decreasing values of γ_l.

Quadratic index. It is defined as

$$\gamma_q = \frac{2}{\sqrt{MN}} \Big\{ \sum_m \sum_n [\min(\mu_{mn}, 1-\mu_{mn})]^2 \Big\}^{\frac{1}{2}} \tag{4.5}$$

or

$$\gamma_q = \frac{2}{\sqrt{MN}} \Big\{ \sum_{g=0}^{L-1} h(g)[\min(\mu_g, 1-\mu_g)]^2 \Big\}^{\frac{1}{2}} \tag{4.6}$$

The fuzziness value is zero if all membership values are 0.0 or 1.0 (ordinary set—binary image). The fuzziness value is maximum when all membership values are equal to 0.5.

Logarithmic Fuzzy entropy. In [1], a logarithmic function based on the definition of the entropy of Shannon was introduced. The Shannon function $S(\mu_{mn})$ increases monotonically in the interval [0.0, 0.5] and decreases in [0.5, 1.0]. When $\mu_{mn} = 0.5$ for each (m,n), the entropy achieves the maximum value of fuzziness

$$E(X) = \frac{1}{MN \ln 2} \sum_m \sum_n S(\mu_{mn})$$

where

$$S(\mu_{mn}) = -\mu_{mn} \ln(\mu_{mn}) - (1-\mu_{mn}) \ln(1-\mu_{mn})$$

for $m = 0, 1, ..., M-1$ and $n = 0, 1, ..., N-1$. The measure $E(X)$ satisfies the following properties:

1. $0.0 \leq E(X) \leq 1.0$
2. $E(X) = 0.0$ if $\mu_{mn} = 0.0$ or $\mu_{mn} = 1.0$ for each (m,n)
3. $E(X) = 1.0$ if $\mu_{mn} = 0.5$ for each (m,n)
4. $E(X) \leq E(X')$ if X is more crisp than X'
5. $E(X) = E(\bar{X})$ where $\bar{X}$ is the complement of X.

Fuzzy entropy of r-order introduced in [5] and defined as follows:

$$H^r(X)(-1/k) \sum_{i=1}^{k} \Big[\mu(s_i^r) \log(\mu(s_i^r)) + (1-\mu(s_i^r)) \log(1-\mu(s_i^r)) \Big] \tag{4.7}$$

where s_i^r denotes the *i-th* sequence (combination) of r pixels in X, k is the number of sequences and $\mu(s_i^r)$ is the degree with which the combination s_i^r as a whole satisfies a specific property μ.

Hybrid entropy introduced in [5] and defined as

$$H_{hy}(X) = -P_w \log E_w - P_b \log E_b$$

where μ_{mn} denotes the whiteness degree of pixel (m, n), P_w and P_b are the occurrence probability of white pixels $\mu_{mn} = 1.0$ and black pixels $\mu_{mn} = 0.0$, respectively, The quantities E_w and E_b denote the average likeliness of interpreting a pixel as white or black respectively, namely

$$E_w = \frac{1}{MN} \sum_m \sum_n [\mu_{mn} \exp(1 - \mu_{mn})]$$

$$E_b = \frac{1}{MN} \sum_m \sum_n [(1 - \mu_{mn}) \exp(\mu_{mn})]$$

Yager's measure. In [9] Yager argued that the measure of fuzziness should be dependent on the relationship between the fuzzy set X and its complement $\bar{x}$. Thus, he suggested that the measure of fuzziness should be defined as the measure of lack of distinction between X and its complement $\bar{X}$, defined as

$$D_p(X, \bar{X}) = \left[\sum_m \sum_n |\mu_{mn} - (1 - \mu_{mn})|^p \right]^{1/p} \text{for}\, p = 1, 2, 3, ...$$

For $p = 1$, D_1 is called the Hamming metric, and for $p = 2$, D_2 is called the Euclidean metric. Thus the Yager's measure of fuzziness is defined as

$$\eta_p(X) = 1 - \frac{D_p(X, \bar{X})}{|X|^{1/p}} = 1 - \frac{D_p(X, \bar{X})}{(MN)^{1/p}} \tag{4.8}$$

Note that the measure $\eta_p(X)$ also satisfies the five properties stated in the previous entropy measure $E(X)$.

The minimization or maximization of a fuzziness measure can be used in image enhancement and segmentation. In particular, fuzziness measures can be used for image thresholding, as we show in Chap. 9. Given an image X, the thresholding is made by selecting a proper threshold so as to minimize the fuzziness of X.

References

1. De Luca, A., Termini, S.: A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory. Inf. Control **20**(4), 301–312 (1972)
2. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice Hall, Upper Saddle River (2008)

3. Kaufmann, A.: Introduction to the Theory of Fuzzy Subsets - Fundamental Theoretical Elements. Academic, New York (1975)
4. Kerre, E.E., Nachtegael, M. (eds.): Fuzzy Techniques in Image Processing, vol. 52. Physica, New York (2013)
5. Pal, N.R., Pal, S.K.: Higher order fuzzy entropy and hybrid entropy of a set. Inf. Sci. **61**(3), 211–231 (1992)
6. Runkler, T.A.: Selection of appropriate defuzzification methods using application specific properties. IEEE Trans. Fuzzy Syst. **5**(1), 72–79 (1997)
7. Sladoje, N., Lindblad, J., Nystrom, I.: Defuzzification of spatial fuzzy sets by feature distance minimization. Image Vis. Comput. **29**(2), 127–141 (2011)
8. Tizhoosh, H.R.: Fuzzy Image Processing. Springer, Heidelberg (1997)
9. Yager, R.R.: On the measure of fuzziness and negation. Part I: membership in the unit interval. Int. J. General Syst. **5**, 221–229 (1979)

Chapter 5
Java for Image Processing

Insanity: doing the same thing over and over again and expecting different results.

Albert Einstein

Abstract This chapter covers some fundamental concepts of Object-Oriented programming in Java. Fundamental classes of the Java packages `java.awt` and `java.applet` for image processing are presented. Moreover, this chapter introduces the concept of plugins in ImageJ and its on-board tools for plugin development. It starts with the discussion of the code skeleton of a new plugin and the sample plugins that are part of the ImageJ distribution, and covers those parts of the ImageJ API, which are essential for writing plugins, with a special focus on the image representation.

5.1 Basic Concepts

Java is a high-level programming language widely used in Object-Oriented (OO) programming. It can support and handle digital image processing efficiently by providing various classes and methods. A Java program consists of a set of interdependent classes, where each class contains variables called *fields* and functions called *methods*. Main references for Java OO programming are [1, 2].

The three fundamental and general principles of the OO programming paradigm are the following:

- *Abstraction* is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency. Abstraction is related to both

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_5

encapsulation and information hiding. Creating abstract data types (classes) is a fundamental concept in OO programming. Abstract data types work almost exactly like built-in types: it is possible to create variables of a type (called objects) and manipulate those variables by sending messages or requests.
- *Inheritance* enables new classes called subclasses to receive - or inherit - properties and methods of existing classes. When a subclass is defined, we say that it extends the superclass or it is derived from the superclass.
- *Polymorphism* provides separation of interface from implementation to decouple *what* from *how*. Polymorphism allows improved code organization and readability as well as the creation of extensible programs than can be grown not only during the original creation of the program but also when new features are required.

Fundamentals concepts of the OO programming in Java are the following:

Class
The class is a fundamental unit in OO programming languages. A class describes a set of objects that have identical characteristics (data elements) and behaviors (functionality). In this way, an object is an instance of a class and a class can be seen as a type. A class is defined by attributes or properties that in Java are called *fields* and functionality called *methods*. Fields and methods of a class are referred to as class members. The properties of a class are defined by the declaration of the fields used by its instances or objects. Then a class declaration contains all the code useful for the objects created from the class: declarations for the fields that provide the state of the class and its objects, constructors for initializing new objects, and methods to implement the behavior of the class and its objects. In a class we can declare variables and objects. A variable is associated to a primitive type (boolean, char, byte, short, int, long float, double, void). Indeed an object is associated to a class.

Object
An object is a realization of a class description. The creation of a new object is also called creation of a new class instance. An object can have internal data which give its state and methods to produce its behavior. Each object can be uniquely distinguished from every other object.

Method
A method describes a functionality of a class. It consists of a set of instructions which is referred to by a name and can be invoked simply by using its interface, made of the name and the list of its arguments. In this way, the implementation of the method is hidden. The constructor is the method having the same name of the class and it contains the code to initialize the objects of the class. The `new` operator creates a new object, namely it allocates the memory for the object and invokes the class constructor.

Interface
An interface is an abstract class that contains only declaration of methods without implementation. No variables are allowed to be declared by the interface.

Encapsulation
It is the capability to wrap data and methods within a class hiding their implementation. This type of access control is often referred to as *implementation hiding*.

Inheritance
Inheritance allows to define new versions of a class, called subclasses or derived classes which can inherit properties and fields of the original class or base class. A subclass can also rewrite a method yet defined in its superclass by providing a different version. Essentially a program consists of a class hierarchy in which the derived classes or subclasses inherit members by the base class and can add functionalities.

Polymorphism
It is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you to define one interface and have multiple implementations. In Java, it is possible to define two or more methods having the same name in a class, provided that their argument lists are different, i.e., the arguments may differ in their type or number, or in both or they are sorted in different manner. This concept is known as *method overloading*. Overloaded methods should always be part of the same class (or in one of its sub-classes), with the same name but different parameters.

Dynamic Binding
The redefinition of a method uses the dynamic binding to associate at the run time the method name to the version to execute.

Package
The code is organized in compilation units with .java extension. Each unit contains a *public* class having the same name of the unit file. More class files can be grouped in a *package*. The word `package` must be inserted in each unit. A rule of coding is that the name of the class must be also the name of the source file (.java). The name of the package must be also the name of the directory containing all the classes of that package. Fundamental package are

`lang` - basic classes for the design of the Java programming language
`util` - utility classes
`io` - input/output
`text` - formatting of input/output
`awt` - graphics and user interfaces
`awt.event` - dealing with different types of events from keyboard, mouse, ...
`swing` - updates some awt functions
`applet` - creating programs to run on the net
`net` - network applications.

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application. In Listing 5.1, we provide a Java program that paints a colored rectangle into a window. It shows the use of packages, objects, classes, methods, and constructor.

Listing 5.1 ColoredRectangle.java

```
import javax.swing.*;  // provides a set of components to create
                       // graphical user interfaces (GUIs) for
                       // applications and applets
import java.awt.*;  // contains all the classes for creating user
                    // interfaces and painting graphics and images
import java.io.*;  // provides for system input/output

// *******  Declaration of the class ColoredRectangle ********
/* The body (the area between the braces) contains:
- declarations for the fields that provide the state of
  the class  and its objects
- methods to implement the behavior of the class and its objects
- constructors for initializing new objects. */
public class ColoredRectangle  {
//  declaration of variables
   int width;            int x;
   int height;           int y;
// declaration of objects
    JFrame window;      Color color, color2;
  // Jframe is an extended version of java.awt.Frame:
  // adds support for the JFC/Swing component architecture.
// ColoredRectangle():  constructor method
    public ColoredRectangle() {
    // set up the fields
    window = new JFrame("ColoredRectangle");
    window.setSize(400, 200);
    width = 40;     x = 80;
    height = 20;    y = 90;
    color = Color.blue;
    color2 = Color.red;
    window.setVisible(true);
  }
// paint(): show the rectangle in its window
  public void paint() {
    Graphics g = window.getGraphics();
    g.setColor(color);
    g.fillRect(x, y, width, height);
  }
// main method
  public static void main(String[] args)
          throws IOException  {
  ColoredRectangle r = new ColoredRectangle();
  int count = 10;
  while (count > 0)  r.paint();
  }
}
```

5.2 Java for Image Processing

The package `java.awt` contains all the classes for creating user interfaces and for painting graphics and images. This package provides interfaces such as `Paint` and `LayoutManager`. It provides also classes such as `Component`, `Canvas`, Container, Color, Frame, Graphics, Graphics2D, Image, Menu, MenuComponent, Rectangle, Toolkit.

The abstract class jawa.awt.Image is the superclass of all classes that represent graphical images by producing information referred to an image. Indeed the package java.awt.image provides classes and interfaces to get, visualize and process an image. Loading an image can require a lot of time depending of its dimensions. For this reason the loading and the visualization of an image are asynchronous processes. The image transmission is supervised by two interfaces: ImageObserver representing the receiver and ImageProducer representing the sender.

Some interfaces of java.awt.image are as follows:

BufferedImageOp
This interface describes single-input/single-output operations performed on objects of the BufferedImage class.

ImageConsumer
The ImageConsumer and ImageProducer interfaces provide the means for low level image creation. The ImageProducer provides the source of the pixel data that is used by the ImageConsumer to create an Image. It specifies the methods that must be implemented to receive data from an ImageProducer.

ImageObserver
This is an asynchronous update interface for producing notifications about an image while the image is loaded. An instance of this class is able to monitor an image while loaded. For example, it permits to load an image without interrupting other operations. This class only contains the method imageUpdate invoked when the information on a image requested in an asynchronous manner becomes available. This class is implemented by the class Applet and each component of awt are ImageObserver.

ImageProducer
This is an interface which can produce the image data for an Image instance. The ImageProducer interface defines the methods that ImageProducer objects must implement. The methods in the ImageProducer interface let objects of the ImageConsumer class register their interest in an image. In other words, when a consumer object is added to an image producer object, the producer delivers all of the data about the image using the method calls defined in this interface.

Mediatracker
This is a utility class to track the status of a number of media objects. Media objects could include audio clips as well as images, though currently only images are supported. To use a media tracker, one should create an instance of MediaTracker and call its addImage method for each image to be tracked. In addition, each image can be assigned a unique identifier. This identifier controls the priority order in which the images are fetched. It can also be used to identify unique subsets of the images that can be waited on independently. Images with a lower ID are loaded in preference to those with a higher ID number. An example is given in Listing 5.2.

`RasterOp`
It describes single-input/single-output operations performed on `Raster` objects.

`RenderedImage`
This is a common interface for objects which contain or can produce image data in a `Raster` format.

Some classes of the package `java.awt.image` are the following:

`AffineTransformOp`
This class uses an affine transform to perform a linear mapping from 2D coordinates in the source `Image` or `Raster` object to 2D coordinates in the destination `Image` or `Raster` object.

`AreaAveragingScaleFilter`
It is an `ImageFilter` subclass for scaling images using a simple area averaging algorithm.

`BufferedImage`
It is a `Image` subclass that describes an image with an accessible buffer of image data. The `getRGB()` method of the `BufferedImage` class gets the pixel values.

`BufferedImageFilter`
It subclasses the `ImageFilter` class to provide a simple means of using a single-source/single-destination image operator (`BufferedImageOp`) to filter a `BufferedImage` in the Image Producer/Consumer/Observer paradigm.

`ColorConvertedOp`
It performs a pixel-by-pixel color conversion of the data in the source image.

`ColorModel`
It is an abstract class that encapsulates the methods for translating a pixel value to color components (for example, red, green, and blue) and an alpha component.

`ComponentColorModel`
It is a `ColorModel` subclass that works with pixel values by representing color and alpha information as separate samples and that stores each sample in a separate data element.

`ConvolveOp`
It implements a convolution from the source image to the destination image.

`ImageFilter`
It implements a filter for the set of interface methods that are used to deliver data from an `ImageProducer` to an `ImageConsumer`.

`IndexColorModel`
It is a `ColorModel` subclass that works with pixel values, each representing an index to a fixed colormap in the RGB color space.

`LookupTable`
It is an abstract class that defines a lookup table object.

`Raster`
It is a class representing an image as a rectangular array of pixels.

`RGBImageFilter`
It provides an easy way to create an `ImageFilter` object which modifies the pixels of an image in the RGB color space.

Listing 5.1 shows a Java program that paints a colored rectangle into a window. It shows the use of package, objects, classes, methods, and constructor.

5.3 Applet

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

An applet is a program written in Java that can be included in an HTML code. Using a Java technology-enabled browser to view a page that contains an applet, the applet code is transferred to the user's system and executed by the Java Virtual Machine (JVM) of the browser. An applet does not require a `main()` method. An applet can be executed in a browser supporting Java or in an applet-viewer invoked in a HTML page. The interaction with the user takes place only through the graphic interface provided by the classes of the packages `awt` and `swing`. The hierarchy for the class `Applet` is shown in Fig. 5.1.

To implement an applet we define a subclass of the class `java.applet.Applet` containing the following methods:

- `init()` Called by the browser or the applet viewer to inform the applet that it has been loaded into the system. It is always called before the first time the `start()` method is called.

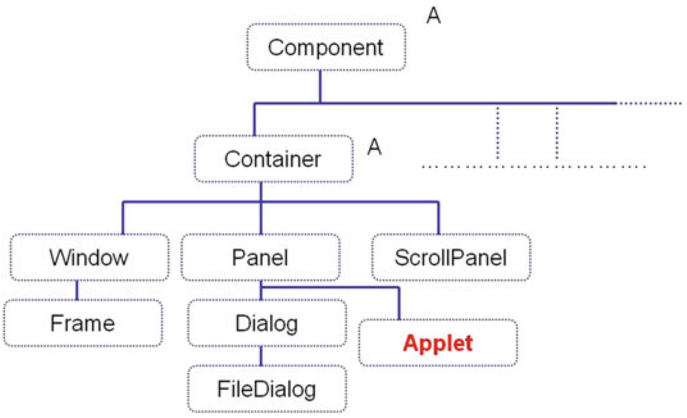


Fig. 5.1 Hierarchy for the class Applet - A indicates an Abstract class

- `start()` Called by the browser or the applet viewer to inform the applet that it should start its execution. It is called after the `init` method and each time the applet is revisited in a Web page.
- `stop()` Called by the browser or the applet viewer to inform the applet that it should stop its execution. It is called when the Web page that contains the applet has been replaced by another page, and also just before the applet is to be destroyed.
- `destroy()` Called by the browser or the applet viewer to inform the applet that it is being reclaimed and that it should destroy any resources that it allocated. The `stop` method is always called before `destroy()`.

Other methods used for applet implementation are: `getAppletContext()`, `getAppletInfo()`, `getAudioClip(URL url)`, `getImage(URL url, String imagineName)`. Moreover, the method `paint()` of the abstract class `Container` has to be written in the applet.

Applets are event-driven programs. An apple waits for an event to occur, then it executes the proper operation and it returns the control to the awt runtime system. The user interaction is realized using the package `java.awt.event` that handles events.

Listing 5.2 shows an example of applet. It visualizes an image ad its component R, G, B. Its execution starts by means of the HTML code given in Listing 5.3. To run the applet it is necessary to create a Java project with the files ScmpRGB.java and ScmpRGB.html, then include in the same directory the image named `sampleimage.jpg` to be processed.

Listing 5.2 ScmpRGB.java: a Java program to decompose an RGB image in the three components Red, Green and Blue.

```java
public class ScmpRGB extends Applet {
    Image img;       // input image
    Image redImg;    // red image
    Image blueImg;   // blue image
    Image greenImg; // green component
    MediaTracker tracker; /* verify the status of the image
    while loaded */
    public void init() {
        //set background color
        setBackground(Color.orange);
        tracker=new MediaTracker(this);
        //get image from the path
        img=getImage(getCodeBase(),"sampleimage.jpg");
        tracker.addImage(img,0);
        try {
            tracker.waitForID(0);
        }
        catch (InterruptedException e) {
          getAppletContext().showStatus
          ("Any␣problems␣with␣the␣first␣image?");
        }

    setSize(img.getWidth(this)*2+60,img.getHeight(this)*2+60)

    redImg=createImage(new FilteredImageSource(img.getSource(),
                new RedFilter())); // extracts red component
        try{
            tracker.addImage(redImg,1);
            tracker.waitForID(1);
            }
        catch (InterruptedException e) {
           getAppletContext().showStatus
           ("Any␣problems␣with␣the␣second␣image?");
            }

    blueImg=createImage(new FilteredImageSource(img.getSource(),
                new BlueFilter())); //extracts blue component
        try{
            tracker.addImage(blueImg,1);
            tracker.waitForID(1);
            }
        catch (InterruptedException e) {
           getAppletContext().showStatus
           ("Any␣problems␣with␣the␣third␣image?");
            }

    greenImg=createImage(new FilteredImageSource(img.getSource(),
                new GreenFilter())); //extracts green component
        try{
            tracker.addImage(greenImg,1);
            tracker.waitForID(1);
            }
        catch (InterruptedException e) {
            getAppletContext().showStatus
            ("Any␣problems␣with␣the␣fourth␣image?");
            }
        }

        public void paint(Graphics g) {
           g.drawImage(img,10,10,this);
           g.drawImage(redImg,img.getWidth(this)+20,10,this);
           g.drawImage(blueImg,10,redImg.getHeight(this)+20,this);
           g.drawImage(greenImg,blueImg.getWidth(this)+20,
                    redImg.getHeight(this)+20,this);
            }
        class RedFilter extends RGBImageFilter {
            public int filterRGB(int x, int y, int rgb) {
                return rgb & 0xFFFF0000;
            }
        }
        class BlueFilter extends RGBImageFilter {
            public int filterRGB(int x, int y, int rgb) {
                return rgb & 0xFF0000FF;
            }
        }
        class GreenFilter extends RGBImageFilter {
            public int filterRGB(int x, int y, int rgb) {
                return rgb & 0xFF00FF00;
            }
        }
    }
```

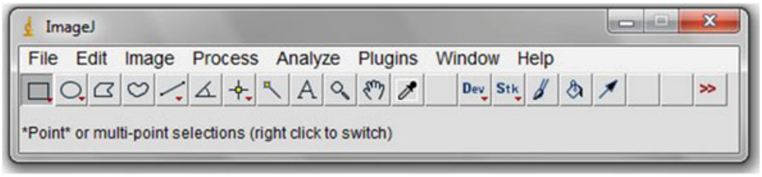

ImageJ
File Edit Image Process Analyze Plugins Window Help
Dev Stk
Point or multi-point selections (right click to switch)

Fig. 5.2 ImageJ user interface

Listing 5.3 **ScmpRGB.Html**: HTML code to run `ScmpRGB.java`.

```
<html>
<body>
<applet code= "ScmpRGB.class" width= "650" height= "700">
</applet>
</body>
</html>
```

5.4 ImageJ

ImageJ is a public-domain Java image processing program inspired by NIH Image for the Macintosh [3]. It can display, edit, analyze, process, save, and print 8-bit, 16-bit and 32-bit images. Figure 5.2 shows the graphical user interface of *ImageJ*.

ImageJ can read many image formats including TIFF, GIF, JPEG, BMP, DICOM, FITS and "raw". It supports "stacks", i.e., a series of images that share a single window. It is multithreaded, so time-consuming operations such as image file reading can be performed in parallel with other operations.

ImageJ is an open source software. Its plugins are free software: any user can redistribute and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation. Specifically, using ImageJ a user has the following essential freedoms:

- to run the program, for any purpose;
- to study how the program works, and change it to make it do what you wish;
- to redistribute copies so you can help your neighbor;
- to improve the program, and release improved versions to the public, so that the whole community can benefit of them.

The functions provided by *ImageJ* built-in commands can be extended by user-written code in the form of macros and plugins. These two options differ in their complexity and capabilities, as explained in the following.

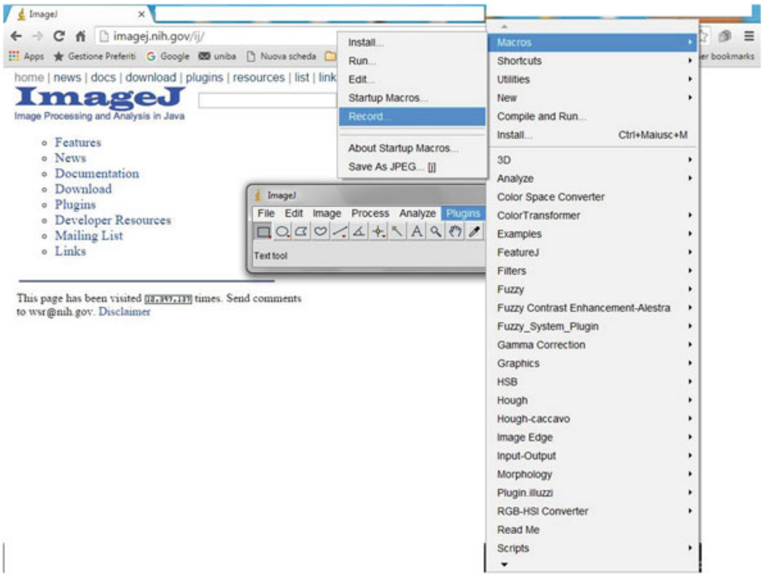


Fig. 5.3 Menu Plugin/Macros/Record in the *ImageJ* environment

5.4.1 *Macros*

Macros are an easy way to execute a series of *ImageJ* commands. The simplest way to create a macro is to use the Record command in the Plugins/Macros/ menu and execute the commands to be recorded. The code of the macro can be modified in the built-in editor (see Fig. 5.3). The *ImageJ* macro language contains a set of control structures, operators and built-in functions and can be used to call built-in commands and macros. Details of the macro language can be found in [4].

5.4.2 *Plugins*

Plugins are a more powerful concept than macros and most of *ImageJ* built-in menu commands are indeed implemented as plugins. Plugins are small Java programs that extend functionalities of *ImageJ*. This means that a plugin can use all features of the Java language, can access the full ImageJ API and use all standard Java APIs. A plugin can implement standard interface classes, as shown in the class hierarchy depicted in Fig. 5.4.

There are three types of plugin

1. *Plugin* that implements the `Plugin` interface. It does not require an image to be opened before execution.

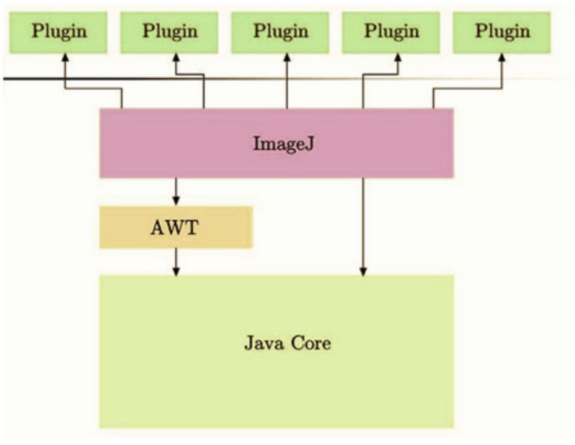


Fig. 5.4 Hierarchy class for Plugins

2. *PluginFilter* that implements the `PlugInFilter` interface. It requires an image to be opened before execution.
3. *PlugInFrame*, like *PluginFilter* but it runs in its own window.

Plugins are commonly used to analyze or process image (stacks) or to add support for new file format. But many other things can be done by plugins, such as rendering graphics or creating extensions of the *ImageJ* graphical user interface. Plugins located in *ImageJ* "plugins" folder are automatically installed in the Plugins menu. Plugins can be created or modified using Plugins/Edit. *ImageJ* provides an integrated editor for macros and plugins, which can be used not only to modify and edit code but also to compile and run plugins.

5.5 Fuzzy Systems in Java

A simple way to define fuzzy models in Java is to use *NRC FuzzyJ Toolkit* [5]. The NRC FuzzyJ Toolkit is a Java(tm) API for representing and manipulating fuzzy information. The toolkit consists of a set of classes (`nrc.fuzzy.*`) that allow a user to build fuzzy systems in Java. It includes a set of Java classes that enable the definition of a fuzzy rule-based system.

To show the use of the FuzzyJ toolkit, in the following we consider the definition of a fuzzy system for color classification [6]. The fuzzy rules are defined so as to perform a segmentation of the HSV color space using a model that follows a human intuition of color classification. The method is based on the HSV color space, which is more intuitive and closer to the human perception of color than the RGB space (humans do not refer to colors in terms of primaries Red, Green and Blue).

The fuzzy rules are defined so as to reflect the process that humans adopt to associate a label to a color according to perception. Thus, we consider the HSV rather than the RGB color model. As explained in Chap. 1, in the HSV model each color is represented in terms of perceptive concepts that are Hue, Saturation and Value. Hence, the fuzzy model has three antecedent variables (Hue, Saturation, and Value) and one consequent variable, which is a color class ID. Fuzzy rules of the following form are considered:

IF (*hue* is orange) AND (*saturation* is medium) AND (*value* is dark)
THEN *colorClass* is darkBrown

IF (*hue* is red) AND (*saturation* is clear) AND (*value* is bright)
THEN *colorClass* is pink

The domain of the antecedent variables Hue, Saturation, and Value is the interval (0,255). The domain of the consequent variable is discrete, and depends on the number of the predefined color classes.

The input variable Hue can be defined by as many fuzzy sets as the number of basic hues, namely: Red, Orange, Yellow, Green, Cyan, Blue, Purple, Magenta, and Pink. The membership functions of these fuzzy sets can be defined according to the spectrum of hues (Fig. 5.5). Saturation is defined using the fuzzy sets Gray, Medium, and Clear, as shown in Fig. 5.6. Value is defined using the fuzzy sets Dark, Medium, and Bright (Fig. 5.7). The fuzzy sets for the output variable colorClass are simply defined as fuzzy singletons (Fig. 5.8).

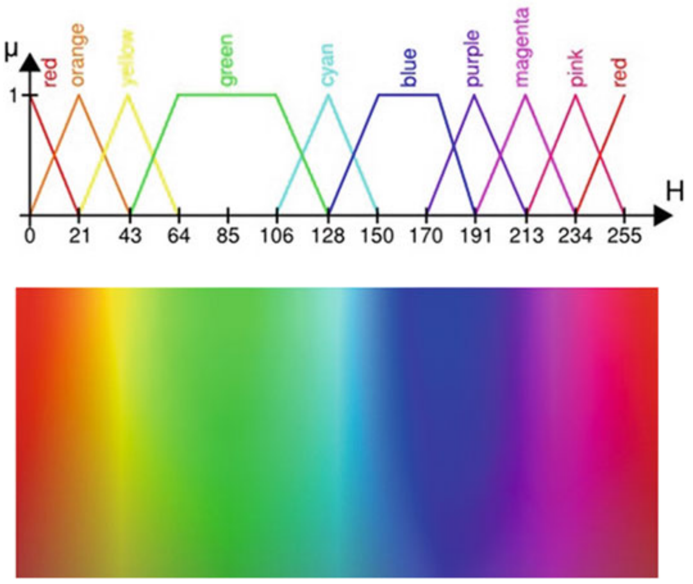


Fig. 5.5 Fuzzy sets for the input variable Hue defined according to the spectrum of all hues

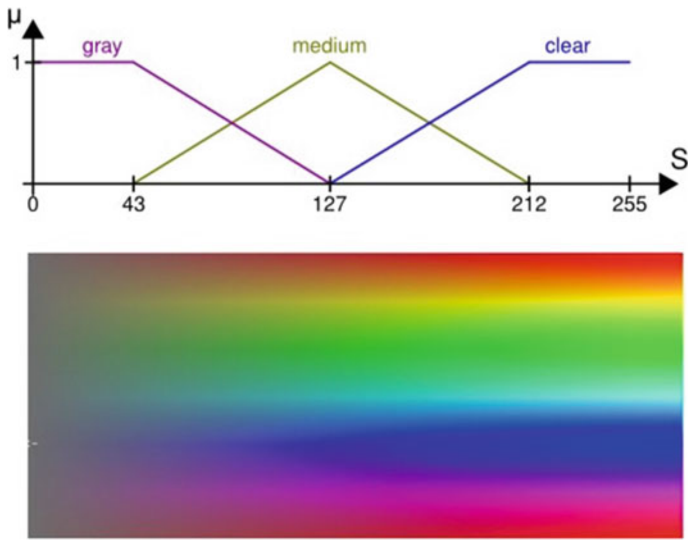


Fig. 5.6 Fuzzy sets for the input variable Saturation

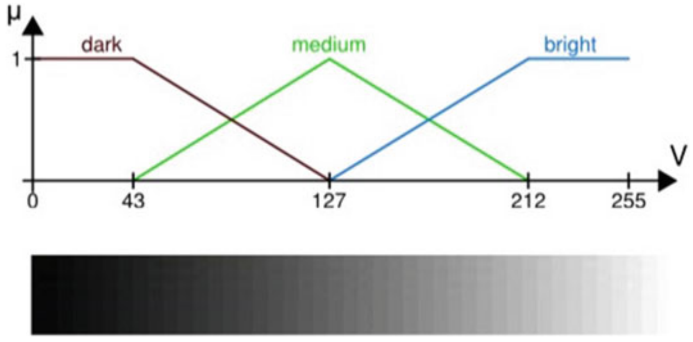


Fig. 5.7 Fuzzy sets for the input variable Value

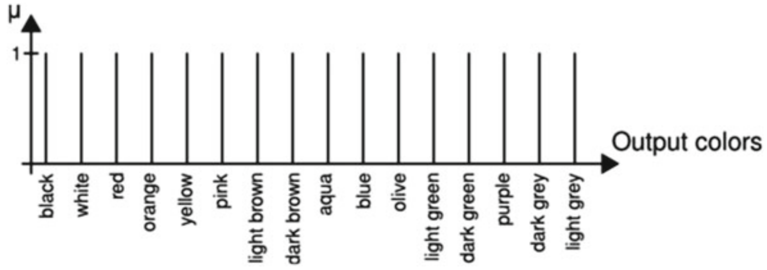


Fig. 5.8 Fuzzy singleton for the output variable *colorClass*

The toolkit *FuzzyJ* provides the class `FuzzyVariable` to define a fuzzy variable and the method `addTerm` to add fuzzy terms to it. To define fuzzy sets with triangular and trapezoidal membership functions we use the class `TriangleFuzzySet` and `TrapezoidFuzzySet` respectively. Listing 5.4 shows the definition of the fuzzy variable Hue and its fuzzy sets.

To define fuzzy IF-THEN rules, *FuzzyJ* provides the class `FuzzyRule` that enables definition of the antecedent and the consequent part using the methods `addAntecedent` and `addConclusion`. In Listing 5.4, we show the definition of the rule `deepRed`. To perform the inference of rules, *FuzzyJ* provides the method `execute()` that computes the activation level of a single rule. The last part of code in Listing 5.4 executes the inference of all the rules having pink as consequent. In the case of classification rules, all rules having the same consequent are executed and their activation levels are summed up to produce the final certainty degree of the corresponding output class. To store the certainty degrees of all the output classes, we can define a HashMap having as many keys as the number of color classes. For a given input, the output of the fuzzy system is given by the maximum value in the HashMap.

Listing 5.4 FuzzyInferenceSystem.java: An excerpt of Java code to define a fuzzy rule-based system using the *FuzzyJ* toolkit.

```java
import nrc.fuzzy.*;
import java.util.*;
/*
 * Fuzzy inference system to classify colors of an image
 *
 * @authors Gabriella Casalino, Marco Lucarelli, Massimo Minervini
 *
 * @version 2.0
 */

public class FuzzyInferenceSystem {

    private FuzzyVariable hue;
    private FuzzyVariable saturation;
    private FuzzyVariable value;
    private FuzzyVariable colorClass;

    private FuzzyRule whitePreprocessing = new FuzzyRule();
    private FuzzyRule blackPreprocessing = new FuzzyRule();
    private FuzzyRule deepRed = new FuzzyRule();
    private FuzzyRule darkRed = new FuzzyRule();
    // [...] other rules
    private FuzzyRule palePink = new FuzzyRule();

/************************************************
 * definition of input and output fuzzy variables
 ************************************************/
// definition of input variable "hue"
hue = new FuzzyVariable("hue",0.0,255.0);
hue.addTerm("red",(new TriangleFuzzySet(0.0,0.0,21.0)).fuzzyUnion
           (new TriangleFuzzySet(234.0,255.0,255.0)));
hue.addTerm("orange",new TriangleFuzzySet(0.0,21.0,43.0));
hue.addTerm("yellow",new TriangleFuzzySet(21.0,43.0,64.0));
hue.addTerm("green",new TrapezoidFuzzySet(43.0,64.0,106.0,128.0));
hue.addTerm("cyan",new TriangleFuzzySet(106.0,128.0,150.0));
hue.addTerm("blue",new TrapezoidFuzzySet(128.0,150.0,175.0,191.0));
hue.addTerm("purple",new TriangleFuzzySet(170.0,191.0,213.0));
hue.addTerm("magenta",new TriangleFuzzySet(191.0,213.0,234.0));
hue.addTerm("pink",new TriangleFuzzySet(213.0,234.0,255.0))
// [...] other input variables

// definition of output variable "color"
colorClass = new FuzzyVariable("color", 0.0, 16.0);

/**********************
 * definition of rules
 **********************/
deepRed.addAntecedent(new FuzzyValue(hue,"red"));
deepRed.addAntecedent(new FuzzyValue(saturation,"gray"));
deepRed.addAntecedent(new FuzzyValue(value,"dark"));
deepRed.addConclusion(new FuzzyValue(colorClass,
    new SingletonFuzzySet(OutputColors.DARK\_BROWN.classId())));

// [...] other rules

/*******************
 * inference of rules
 *******************/

// execute rules having PINK as consequent

map.put(OutputColors.DARK_BROWN,map.get(OutputColors.DARK_BROWN)
  + deepPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.DARK_BROWN,map.get(OutputColors.DARK_BROWN)
  + darkPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.RED, map.get(OutputColors.RED)
  + somberPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.PINK, map.get(OutputColors.PINK)
  + brightPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.PINK, map.get(OutputColors.PINK)
  + mediumPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.RED, map.get(OutputColors.RED)
  + grayPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.PINK, map.get(OutputColors.PINK)
  + luminousPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.PINK, map.get(OutputColors.PINK)
  + lightPink.execute().toFuzzyValueArray()[0].getMaxY());
map.put(OutputColors.PINK, map.get(OutputColors.PINK)
  + palePink.execute().toFuzzyValueArray()[0].getMaxY());

// [...] execute other rules

// the output is the color with highest certainty degree

for (OutputColors c : OutputColors.values())
  if (map.get(c) > map.get(pixelClass))
    pixelClass = c;
```

The complete Java code to define and use the above described fuzzy rule-based system is available as *ImageJ* plugin at [7]. This plugin is an example of how to construct a fuzzy image processing application by importing in *ImageJ* the classes of the `nrc.fuzzy.*` package. *FuzzyJ* offers a facility to easily implement plugins

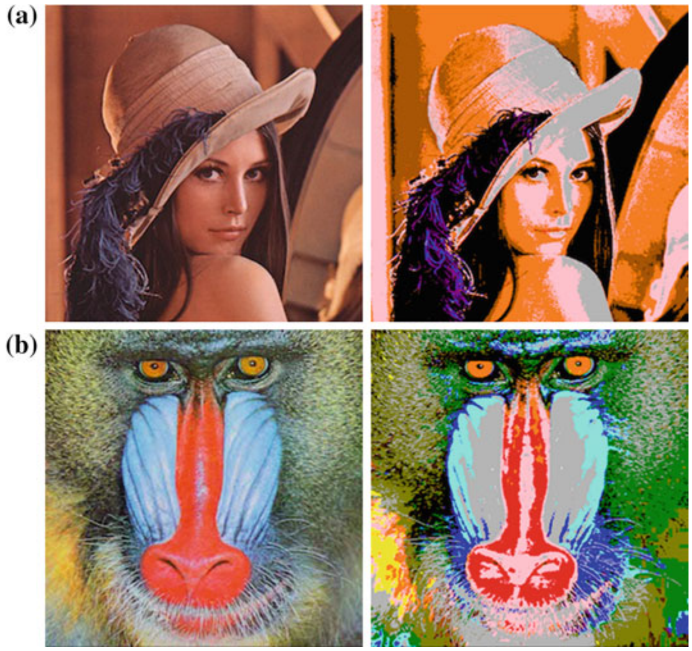


Fig. 5.9 Color classification obtained by fuzzy rules on **a** the Lena image, **b** the Baboon image

or applets. In Fig. 5.9, we show some results obtained by using the plugin to classify the pixel color of the Lena and the Baboon image.

References

1. Eckel, B.: Thinking in JAVA. Prentice Hall Professional, USA (2003)
2. Trail: Learning the Java Language. https://docs.oracle.com/javase/tutorial/java/
3. ImageJ. Image Processing and Analysis in Java. http://imagej.nih.gov/ij/index.html
4. ImageJ Macro Language. http://rsb.info.nih.gov/ij/developer/macro/macros.html
5. FuzzyJ Toolkit for the Java(tm) Platform. http://www.csie.ntu.edu.tw/~sylee/courses/FuzzyJ/Docs/
6. Shamir, L.: Human perception-based color segmentation using fuzzy logic. In: International Conference on Image Processing. Computer Vision and Pattern Recognition (IPCV 2006), Las Vegas, NV, vol. II, pp. 496–505 (2006)
7. Casalino, G., Lucarelli, M., Minervini M.: FuzzySegmentation, a Java plugin for color segmentation. https://sites.google.com/site/cilabuniba/research/fuzzysegmentation

Part II
Application to Image Processing

Fuzzy set theory finds a promising field of application in digital image analysis. Fuzzy sets are suitable to address situations where image components cannot easily be defined in a precise way, rather they can be better described in terms of their diffuse localization and extent. Fuzziness is an intrinsic quality of images and a natural outcome of many imaging techniques. Hence fuzzy set theory is advantageous when applied to images with little contrast or images immersed in noise, for which most standard contrast enhancement techniques fail to provide good results. Fuzzy concepts, incorporated in image segmentation techniques, are a useful tool for reducing the loss of data that is caused by hard decisions in the object definition. As well, using the concept of fuzzy sets in morphology we can represent both imprecision and uncertainty in images, from the signal level to the highest decision level. To show the potential of fuzzy techniques in image processing, this part of the book deals with fundamental tasks of image processing and shows how fuzzy techniques can be successfully applied to accomplish these tasks.

Chapter 6
Color Contrast Enhancement

One day machines will be able to solve problems, but none of them will be able to deliver us one

Albert Einstein

Abstract In this chapter, we present the basic concepts of color contrast enhancement and focus on the use of fuzzy logic as a valid tool to enhance color images. In particular, we show how to define a fuzzy rule-based system for color image enhancement. An application to real-world color images is presented.

6.1 Introduction

The use of digital processing techniques for image enhancement has received much interest especially in applications related to medical image research. Image enhancement consists of a collection of techniques that try to improve the visual appearance of an image or to convert the image to a form better suited for analysis by a human viewer or for machine processing.

Classical image enhancement methods can be classified into two groups, namely frequency domain and spatial domain methods. In the first case, image enhancement is performed by applying filtering on the frequency transform of an image. However, computing a two-dimensional transform for an entire image is a very time consuming task even with fast transformation techniques [7]. Thus, frequency domain methods are not suitable for real time processing. Spatial domain techniques directly operate on the pixels by using information such as histograms or moments (see Chap. 2). Contrast enhancement is one of the important image enhancement techniques in spatial domain. Enhancement methods for monochrome images such as contrast manipulation by means of gray level transformation or histogram modification could be

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_6

applied to color images by processing each color component individually. Typically, color images are processed in the RGB color space.

Indeed the nature of color images introduces some challenges in the image enhancement process, such as the choice of the color representation system and the selection of the more representative channels.

6.2 Multichannel Image Processing

Since a color image contains a multichannel information in a particular representation system (RGB, HSI, ...) generally methods developed for single channel data could not be directly applicable to multichannel data. For example, histogram equalization and its variants are quite useful for enhancing the details in gray level images, but could fail when applied to the three components (R, G, B) of a degraded color image, since they alter the original color composition by producing color artifacts.

Therefore, the application of color image enhancement on the RGB color model could produce results that are not adherent to the human visual system. Color models suitable for color image enhancement should enable splitting the achromatic and chromatic information, as well as maintain the color distribution of the original image. The three main attributes generally used to distinguish one color from another are hue, saturation and intensity thus the HSI model turns out to be more appropriate for color image enhancement.

Several adaptive filters for color image processing have been introduced in the last years [12, 14]. In [11] an adaptive methodology has been proposed that constitutes a unifying and powerful framework for multichannel signal processing. Using this methodology, color image filtering problems are treated from a global viewpoint that readily yields and unifies previous, seemingly unrelated, results. The new filters utilize Bayesian techniques and nonparametric methodologies to adapt to local data in the color image.

6.3 Fuzzy Techniques for Color Enhancement

Many image enhancement algorithms using fuzzy techniques have been proposed. Some examples are given in [2, 3, 6, 8, 10, 13]. In [3], the authors discuss a method for image enhancement based on manipulation of gray-level pixels. Their approach involves a transfer of the image into a fuzzy domain and modification using a contrast intensification function. This is followed by a defuzzification operation which converts the data from fuzzy domain to a spatial domain by using an inverse transformation function. Moreover, in [6] a new intensification operator has been proposed, it uses a parametric sigmoid function for the modification of the Gaussian membership function on the basis of optimization of the entropy. The approach presented in [5] extended in [6] for the enhancement of color images uses histogram as the basis for fuzzy modeling of color images. In [10], fuzzy entropy is used to derive a measure of image quality in the fuzzy domain, although the image quality remains subjective in nature. This subjectivity in the evaluation of the quality of a color image arises the necessity to have tools that simplify the definition of a fuzzy system to be used for the specific contrast enhancement application.

In the field of image enhancement and smoothing using fuzzy logic, interesting frameworks have been developed using fuzzy IF-THEN rules. In fact, this approach permits to extend and generalize enhancement methods based on histogram transformation. In [9] a fuzzy rule-based system is proposed for image enhancement. Here, a set of neighborhood pixels constitutes the antecedent and the consequent clauses of fuzzy rules that offer directives similar to human-like reasoning.

In the following section, we show how to define a fuzzy rule-based system for color contrast enhancement.

6.4 A Fuzzy Rule-Based System for Color Enhancement

As described in Chap. 3 a fuzzy rule based system processes crisp data at the input and produces crisp data at the output through inference from a fuzzy rule base. A fuzzifer is used at the front of the system to convert crisp data to fuzzy sets, and a defuzzifier is used at the output of the system to convert fuzzy sets into crisp values. The core of the fuzzy system is the rule base typically made of K rules of the form given in (3.1). Given a vector of input values three main steps are performed in order to derive an output value:

- At first, fuzzification of input values is performed by evaluating a degree of membership to each fuzzy set describing the input variable. Namely, for each input x_i the degrees of membership $\mu_{ik}(x_i)$ to the fuzzy sets A_{ik} are computed according to the type of membership functions. If the fuzzy system has to be defined for a color image processing task, typically the input values x_i are the intensity values of the pixels $I_c(n, m)$ of a $N \times M$ color image, for $c = 1, 2, 3$ channels and $0 \leq n \leq N - 1$, $0 \leq m \leq M - 1$. The fuzzification of the input image I consists of applying a membership functions to each color channel, thus producing the membership values $\mu_{n,m,c}$ for each pixel of each channel. These membership values represent the fuzzified input, namely the input fuzzy image.
- Then, a fuzzy output is obtained by means of a fuzzy inference process which combines the fuzzified input (membership values) with the fuzzy rules through a compositional rule of inference. The resulting fuzzy sets are aggregated via a fuzzy union operator to provide an output fuzzy set.
- Finally, a defuzzification stage is needed to obtain a crisp output from the fuzzy output resulting from the inference of rules. This stage is performed by the defuzzifier which maps the output fuzzy set to a single crisp output value.

In particular, to address the task of color contrast enhancement by means of a fuzzy rule-based system, the following main steps have to be carried out:

1. Select a 32-bit color image
2. Define a fuzzy rule-based system and set all the parameters of the fuzzy rules (input features, output features, membership functions,..)
3. Fuzzification of the color image for each of the color channels
4. Inference for each channel
5. Defuzzification of the inference results
6. Recompose the 32-bit color image

All these steps are often difficult to be performed manually. Here, we suggest the use of a Java plugin [1] that is intended to aid the user in the definition of a fuzzy rule-based system for gray level image enhancement. To obtain color image enhancement, we decompose the image in the color channels and apply the plugin to each color channel.

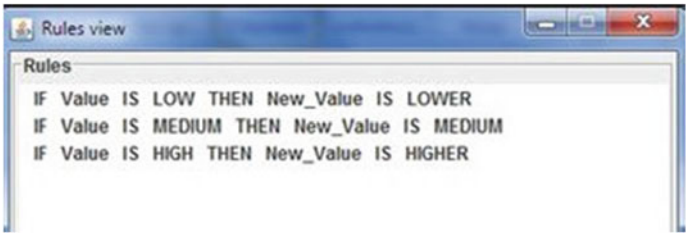
6.5 Example: Natural Image Enhancement

In this section, we present examples of natural image enhancement based on a simple fuzzy rule-based system. Some images that present a variety of colors and poor contrast between regions have been considered for testing. In particular, images have poor brightness, i.e., they are under-exposed and characterized by not discernable details and colors that are not well perceivable to the eye. All images were represented using the RGB model and then transformed in the HSB model. For each image, we derive the three channel components H, S, B and a fuzzy contrast enhancement is applied to pixel values of each component image. The contrast enhancement is carried out by applying three well-known fuzzy rules for gray-level contrast enhancement (see [4]) listed in Fig. 6.1. In our case, the input variable *Value* is the pixel level of the channel component in the input image and the output variable *New value* is the pixel value of the corresponding component in the enhanced image. The input variable is modeled by three fuzzy sets *Low, Medium, High* with the membership functions defined so as to fit the histogram of the image component. The output variable is modeled by fuzzy singletons *Lower, Medium, Higher*. Figures 6.2 and 6.3 show the input and output fuzzy sets, respectively.

To define the fuzzy rules and the input and output fuzzy sets, we refer to a Java application [1] able to build fuzzy rule-based systems for image processing.

Given any input value x_0, the inference of the three rules produces the following output value:

$$y_0 = \frac{\mu_{LOW}(x_0) \cdot b_L + \mu_{MEDIUM}(x_0) \cdot b_M + \mu_{HIGH}(x_0) \cdot b_H}{\mu_{LOW}(x_0) + \mu_{MEDIUM}(x_0) + \mu_{HIGH}(x_0)}$$



IF Value IS LOW THEN New_Value IS LOWER
IF Value IS MEDIUM THEN New_Value IS MEDIUM
IF Value IS HIGH THEN New_Value IS HIGHER

Fig. 6.1 Fuzzy rules for contrast enhancement

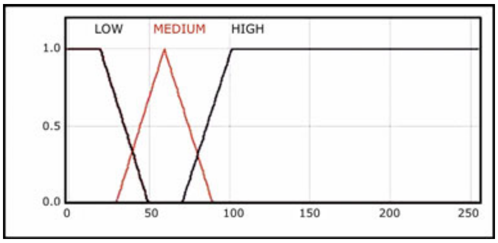


Fig. 6.2 Fuzzy sets defined on the input variable *Value*

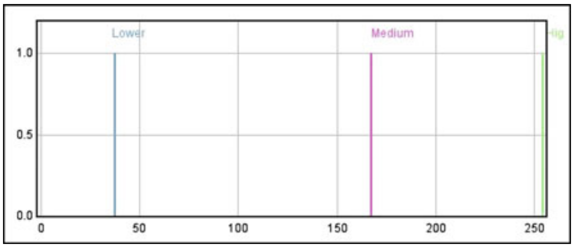


Fig. 6.3 Fuzzy singleton defined for the output variable

where b_L, b_M, v_H are the values of the output fuzzy singleton LOWER, MEDIUM, and HIGHER, respectively. The use of fuzzy singletons in the consequent part of a fuzzy rule reduces the computational burden of the system. Indeed, fuzzy image enhancement is computationally intensive because the entire process of fuzzification, rule inference and defuzzification is applied to every pixel in the input image.

To further reduce the computational complexity, one possibility is to apply the fuzzy inference rules only to all the 256 levels of an image component, and then construct a look-up table that provides correspondence between values of the input image and values of the enhanced image. In this way for each rule-based system we have a pre-compiled look-up table, thus the enhancement requires only the application of the look-up table. Operating with a look-up table is possible because the gray level transformations are point transformations and do not depend on the pixel position.

Given a color image f, the complete procedure for color contrast enhancement is the following:

```
Require: f = 32-bit color image;
Ensure: f' = 32-bit color enhanced image;
/*steps performed on f */

1. Select the color space (for example HSB)
2. Convert from RGB image to HSB stack;
3. Select one component - for example Brightness
4. Define the fuzzy rule-based system
   /* Steps performed on the Brightness component */
5. Fuzzification of each pixel value (memberships to the Low, Medium and High
fuzzy sets)
6. Inference of rules
7. Computation of the output value
8. Convert from HSB stack to RGB color using the obtained enhanced Brightness
```

The enhanced images produced after application of the fuzzy rule-based system only to the Brightness component are presented in Fig. 6.4. A qualitative comparison with enhanced images obtained by histogram equalization is made. It can be seen that in most cases a visually pleasing image is obtained with the fuzzy rule-based color enhancement. The improvement attained with the fuzzy approach is more pleasing in

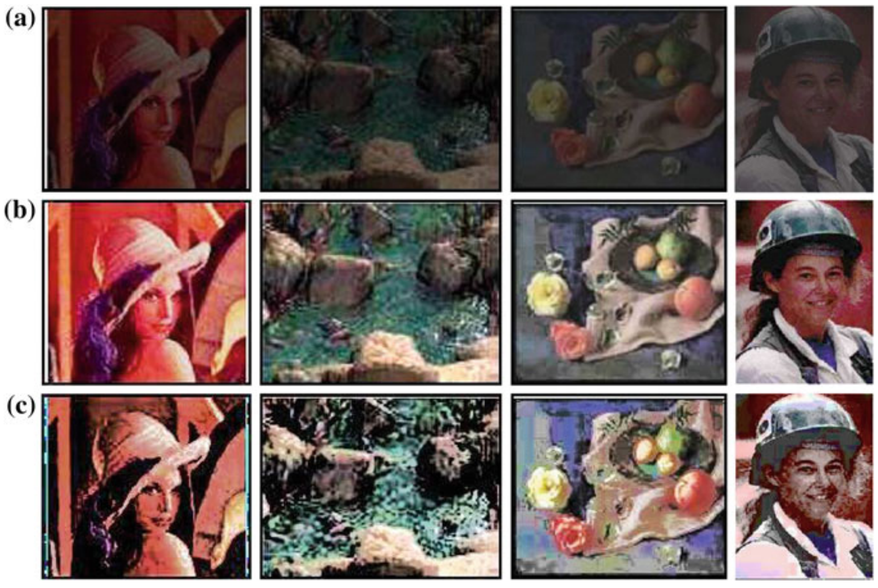


Fig. 6.4 **a** Original poorly contrasted images. **b** Images enhanced by histogram equalization. **c** Images enhanced by fuzzy rule-based system approach followed by histogram equalization

nature than that with the histogram equalization, which tends to over-enhance certain regions in some cases. Moreover, the presented approach permits to easily enhance the range of color values necessary to a specific visual application.

References

1. Alestra S.: ImageJ Plugin Fuzzy Contrast Enhancement. Availableat: http://svg.dmi.unict.it/iplab/imagej/index.htm
2. Bhattacharya, E.: An algebraic environment to process fuzzy images. Pattern Recognit. Lett. **8**, 29–33 (1988)
3. De, T.K., Chatterji, B.N.: An approach to a generalized technique for image contrast enhancement using the concept of fuzzy set. Fuzzy Sets Syst. **25**, 145–158 (1988)
4. Gonzales, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Prentice-Hall Inc, Upper Saddle River, NJ, USA (2006)
5. Hanmandlu, M., Jha, D.: An optimal fuzzy system for color image enhancement. IEEE Trans. Image Process. **15**(10), 2956–2966 (2006)
6. Hanmandlu, M., Tandon, S.N., Mir, A.H.: A new fuzzy logic based image enhancement. Biomed. Sci. Instrum. **34**, 590–595 (1997)
7. Lee, J.-S.: Digital image enhancement and noise filtering. IEEE Trans. Pattern Anal. Mach. Intell. **2**, 165–168 (1980)
8. Mahashwari, T., Amit, A.: Image enhancement using fuzzy technique. Int. J. Res. Eng. Sci. Technol. **2**(2), 1–4 (2013)
9. Pal, S.K., Rosenfeld, A.: Image enhancement and thresholding by optimization of fuzzy compactness. Pattern Recognit. Lett. **7**(2), 77–86 (1988)
10. Pal, S.K., King, R.A.: Image enhancement using smoothing with fuzzy sets. IEEE Trans. Syst. Man. Cybern. **SMC-11**(7), 494–501 (1981)
11. Plataniotis, K.N., Androutsos, D., Vinayagamoorthy, S., Venetsanopoulos, A.N.: Color image processing using adaptive multichannel filtering. IEEE Trans. Image Process. **6**(7), 933–941 (1997)
12. Raju, G., Nair, M.S.: A fast and efficient color image enhancement method based on fuzzy-logic and histogram. AEU-Int. J. Electron. Commun. **68**(3), 237–243 (2014)
13. Sun, S.: Image enhancement algorithm based on improved fuzzy filter. J. Multimed. **9**(1), 138–144 (2014)
14. Suneel, M., Kumar, K., Bhaskar, P.U.: Color image enhancement using fuzzy set theory. Digital Image Process. **4**(1), 10–12 (2012)

Chapter 7
Image Segmentation

On everything the party considers itself the nature of the whole
Galileo Galilei

Abstract This chapter deals with the methods of region-based image segmentation. It introduces some basic concepts such as definition of pixel neighbors, connectivity of a region, and the image segmentation problem. This chapter also describes clustering methods as powerful tools for image segmentation. Two application examples using clustering for color image segmentation and texture segmentation are provided.

7.1 Introduction

Image segmentation is the process of partitioning an image into nonoverlapped regions which are homogeneous with respect to some characteristics such as intensity, color or texture. Image segmentation is a fundamental step for high-level vision and image understanding, necessary in many applications such as object/pattern recognition and tracking, image retrieval, and so on. In other words the goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze [10, 22]. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

Although a large number of different segmentation methods have already been published in past years [6, 9–11, 16, 22] and other novel algorithms are continually appearing, the segmentation problem is still far from being satisfactorily solved for real-world images. The performance of each method depends highly on the type of visual scenes and on image parameters, such as resolution, illumination, and viewing

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_7

conditions. Fundamental segmentation methods detect discontinuity among pixels of different regions or similarity among pixels of the same region. The first category includes algorithms for detecting isolated points, lines, or edges. In the second category we find algorithms of region growing, region splitting, and region merging. In this chapter we introduce specific region-based approaches that are based on clustering. Clustering-based approaches received a great interest in several domain, such as medical imaging and content-based image retrieval.

In the following we present the segmentation problem and methods for handling it. Then crisp and fuzzy clustering are introduced. Finally a fuzzy clustering embedded with spatial information is presented. In the last section we show the use of clustering for two applications: color segmentation and texture segmentation of biological images.

7.2 The Segmentation Problem

Generally the segmentation problem involves separating the image pixels into a number of distinct partitions corresponding to homogeneous regions. The goal of segmentation is to accurately capture these regions. To formalize the segmentation problem we introduce the concept of neighbors of a pixel and connectivity of a region.

Pixel Neighbors
There are two different ways to define the neighbors of a pixel p located at (x, y)

1. The *4-neighbors* of pixel p, denoted by $N_4(p)$, are the four pixels located at $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, and $(x, y+1)$ that are located, respectively, north, south, west, and east of the pixel p.
2. The *8-neighbors* of pixel p, denoted by $N_8(p)$, include the *4-neighbors* and four pixels along the diagonal direction located at $(x-1, y-1)$ (northwest), $(x-1, y+1)$ (northeast), $(x+1, y-1)$ (southwest), and $(x+1, y+1)$ (southeast).

Region Connectivity
A region is connected if all pairs of its pixels are connected. Two neighboring pixels are connected if their values are close to each other, i.e., they both belong to the same subset of gray levels sharing the same property: $p \in V$ and $q \in V$, where V is a subset of all gray levels in the image that share a particular property. The connectivity can be defined as one of the following:

1. *4-connected.* Two pixels p and q are *4-connected* if they are *4-neighbors* and $p \in V$ and $q \in V$;
2. *8-connected.* Two pixels p and q are *8-connected* if they are *8-neighbors* and $p \in V$ and $q \in V$.

Let R be the spatial region occupied by the image. The segmentation process can be seen as the problem of partitioning R into a number of subregions R_1, R_2,..., R_n such that

1. $\bigcup_{i=1}^{n} R_i = R$ where R_i is a connected set, $i = 1 \ldots n$;
2. $R_i \bigcap R_j = \emptyset$ for $i \neq j$
3. $Q(R_i) = TRUE$ for $i = 1 \ldots n$
4. $Q(R_i \bigcup R_j) = FALSE$ for each pair of adjacent regions (R_i, R_j) where $Q(R_i)$ is a predicate defined on region R_i.

The first property means that the segmentation must be complete; that is, every pixel must belong to a region and points in a region must be connected in some predefined sense (e.g., 4-connected or 8-connected). The second property indicates that the regions must be disjoint. Property 3. require the pixels belonging to a region to satisfy a predicate Q. For example $Q(R_i) = TRUE$ if the average gray level of all pixels in R_i is less than a value m and its standard deviation is less than a value σ. Property 4. indicates that two regions R_i and R_j are different in the sense of a predicate Q.

7.3 Methods for Segmentation

Numerous methods are available for image segmentation. They can be selected based on the specific applications and imaging modality. Imaging artifacts such as noise, partial volume effects, and motion can also have significant influence on the performance of segmentation algorithms. The main methods for image segmentation may be grouped in the following manner:

- *Discontinuity-based methods*. Some segmentation methods such as thresholding look for the boundaries between regions based on discontinuities in gray scale, color, or other properties. Thresholding approaches segment an image by creating a binary partitioning of the image intensities. A thresholding procedure attempts to determine an intensity value, called threshold, which separates the desired classes. The segmentation is then achieved by grouping all pixels with intensity greater than the threshold into one class, and all other pixels into another class. See chapter 8 for more details.
- *Region-based methods.* Region-based segmentation is a technique for finding the region directly. One basic method among these is region growing that groups pixels or subregions into larger regions based on predefined criteria for growth. The basic approach involves the selection of a set of 'seed' points and from these grow regions iteratively by determining whether the pixel neighbors should be added to the region.

Another category of methods for image segmentation, and the one considered in this book, is that of clustering methods. Clustering is the process of grouping a set of points—feature vectors—into subsets (called clusters) so that points in the same cluster are similar in some sense [13]. The application of clustering to image segmentation requires taking into account also the spatial information. To this aim, a common strategy is to divide the image into a number of blocks and to extract a number of local features for each block. Successively, the clustering algorithm is applied to these features and a predefined number of clusters is obtained. Both crisp and fuzzy clustering schemes have been proposed for image segmentation. Fuzzy techniques revealed more robust than crisp algorithms especially in case of images characterized by some form of ambiguity, such as poor contrast, noise, and dishomogeneity in the intensity values.

Among fuzzy clustering algorithms the most used one is the well-known Fuzzy C-Means (FCM) algorithm. Despite its widespread use, FCM does not always provide good segmentation results due to the fact that it does not incorporate any information concerning the spatial context, which is fundamental because the obtained regions are likely to be disjoint, irregular, and noisy. In order to achieve more effective segmentation results, many works have been proposed aiming at incorporating the local spatial information into clustering schemes based on the conventional FCM algorithm [1, 5, 19, 21]. In the following the main algorithms for crisp, fuzzy, and spatial clustering are introduced. In [20] an ImageJ plugin implementing all these clustering algorithms is available.

7.3.1 *Crisp Clustering*

The conventional (crisp) clustering methods assume that each point of the data set belongs to exactly one cluster. The major example of partitional crisp clustering is the *K-means* algorithm [14]. Even though *K-means* was first proposed over 50 years ago, it is still by far the most used clustering algorithm for its simplicity of implementation and its effectiveness.

The *K-means* clustering aims to partition N point into K partitions (clusters) in which each point belongs to the cluster having the nearest mean. Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be the set of data points and $V = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$ be the set of cluster means (centers). A partition of X into K clusters can be represented by mutually disjoint sets $C_1, \ldots, C_K$ such that $C_1 \cup \cdots \cup C_K = X$. To represent the partition of X into K clusters, a binary membership matrix $U = [u_{ik}]$ is used, where $u_{ik} = 1$ if $\mathbf{x}_i \in C_k$, $u_{ik} = 0$ otherwise, for $i = 1 \ldots N$ and $k = 1 \ldots K$.

The objective of the *K-means* algorithm is to minimize the distance among points inside the same cluster and to maximize the distance between clusters. This is obtained by minimizing the following objective function:

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} u_{ik} d(\mathbf{x}_i, \mathbf{c}_k)^2 \tag{7.1}$$

where $d(\cdot, \cdot)$ is the Euclidean distance.

The main steps of the *K-means* algorithm for image segmentation are

1. Fix the parameter K (number of clusters) and initialize the cluster centers $\mathbf{c}_k$ ($k = 1 \ldots K$), either randomly or based on some heuristic;

2. Assign each pixel in the image to the cluster that minimizes the distance between the pixel and the cluster center;
3. Recompute the cluster centers by averaging all of the pixels in the cluster, namely

$$c_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \text{for } k = 1 \ldots K \tag{7.2}$$

4. Repeat steps 2 and 3 until convergence is attained (i.e., the assignment of pixels to clusters does not change).

One main issue of the K-means algorithm is that the clustering result depends strongly on the initialization of the cluster centers and on the number of clusters. Besides, *K-means* usually converges to a local minimum does not take data distribution in consideration.

7.3.2 Fuzzy Clustering

Fuzzy clustering methods assume that each data point belongs to more than one cluster with different membership degrees and vague or fuzzy borders between different cluster. The main fuzzy clustering algorithm is the fuzzy version of the K-means called *Fuzzy C-Means* (*FCM*) [3]. *FCM* is a partition clustering method based on the minimization of the following objective function:

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} (u_{ik})^m \, d\,(\mathbf{x}_i, \mathbf{c}_k)^2, \quad 1 < m < \infty, \tag{7.3}$$

where $d(\mathbf{x}_i, \mathbf{c}_k)$ is the distance between the point $\mathbf{x}_i$ and the cluster centroid $\mathbf{c}_k$, m is the fuzziness parameter, K is the number of clusters, N is the number of data points $\mathbf{x}_i$, $u_{ik} \in [0, 1]$ is the membership degree of $\mathbf{x}_i$ belonging to the cluster k, calculated as follows:

$$u_{ik} = \frac{1}{\sum_{l=1}^{K} \left(\frac{d(\mathbf{x}_i, \mathbf{c}_k)}{d(\mathbf{x}_i, \mathbf{c}_l)} \right)^{\frac{2}{m-1}}}. \tag{7.4}$$

for $i = 1 \ldots N$, $k = 1 \ldots K$. Using the fuzzy membership matrix $U = [u_{ik}]$ a new position of the k-th centroid is calculated as

$$\mathbf{c}_k = \frac{\sum_{i=1}^{N} (u_{ik})^m \mathbf{x}_i}{\sum_{i=1}^{N} (u_{ik})^m} \tag{7.5}$$

with the constraint $\sum_i u_{ik} = 1$.

Given the initial parameters (number of clusters K and fuzziness parameter m), FCM iteratively computes the matrix U according to Eq. (7.4), and updates the centroid positions as in Eq. (7.5). The algorithm terminates after a fixed number of iterations, or if the improvement expressed by J is substantially small. The parameter m determines the fuzziness degree of the clustering process. If this parameter has value 1 the fuzzy c-means approximates the crisp K-means algorithm, being the membership values equal only 0 or 1. The most common choice for m is 2.

From Eq. (7.3) it can be observed that FCM does not incorporate any spatial dependencies between observations. This may degrade the overall clustering result in case of image segmentation, because neighboring regions may be highly correlated and thus they should belong to the same cluster.

7.3.3 *Spatial Fuzzy Clustering*

When applied to image segmentation, clustering should take into account the spatial information of pixels. To this aim, several spatial variants of the FCM have been proposed. Among these, the *Spatial FCM* (*SFCM*) proposed in [7] uses a spatial function which is defined as:

$$h_{ij} = \sum_{k \in NB(\mathbf{x}_i)} u_{ik} \tag{7.6}$$

where $NB(\mathbf{x}_i)$ represents a neighbor of the pixel $\mathbf{x}_i$ in the spatial domain. Just like the membership function, the spatial function h_{ij} represents the membership degree of pixel $\mathbf{x}_i$ belonging to the jth cluster.

The spatial function of a pixel for a cluster is large if the majority of its neighbors belongs to the same clusters. The spatial function modifies the membership function of a pixel according to the membership statistics of its neighbors as follows:

$$u_{ij} = \frac{u_{ij}^p h_{ij}^q}{\sum_{k=1}^K u_{ik}^p h_{ik}^q} \tag{7.7}$$

where p and q are parameters to control the relative importance of both functions. Each iteration of the *SFCM* includes two steps. The first one is the same as in standard *FCM* to calculate the membership function in the feature domain. In the second step, the membership information of each pixel is mapped to the spatial domain, and the spatial function is computed from that. The *FCM* iteration proceeds with the new membership that is incorporated with the spatial function.

7.4 Example: Color Segmentation

In this section, we present some comparative results of color image segmentation using different clustering methods. These results were obtained using the *ImageJ* plugin *SFCM* available at [20] (see Appendix A). The *SFCM* plugin includes the following clustering plugins: *Jarek Sacha's K-Means* [12], *K-Means*, *Fuzzy C-Means*, *Spatial Fuzzy C-Means* [7].

The plugin works on different color spaces. The supported color spaces are: XYZ, La*b*, and HSB. One can initially transform a RGB color image into HSB or XYZ or La*b* color spaces and then segment the image in the new color space. Additional features of the plugin are: smart cluster centroid initialization by implementing *K-Means++*, speeding up convergence, stopping criterion selection (number of iterations, norm on parameter matrices, etc.), different methods for result visualization. The plugin configuration enables selection of the following parameters:

- Number of cluster
- Maximum number of iterations
- Stopping criterion selection
- Tolerance value: Threshold used to stop the algorithm
- Initialization criterion for the centers and the membership matrix (K-Means++ or Random)
- Randomized seed: Integer used as seed to initialize a random number sequence
- Fuzziness value m: if m is near 1, results are similar to those obtained by *K-Means*
- Parameters p and q used to control the relative importance of membership and spatial functions
- Radius r: the spatial function is evaluated on a $(2r+1)x(2r+1)$ window centered on the current pixel
- Visualization mode for displaying the segmented image. The regions can be labeled in different ways

    1. by the cluster centroid color: each point of a cluster is labeled with the color of its centroid (in case of color conversion the color space is converted back to RGB);
    2. by a gray level: each pixel is labeled with the number of the cluster it belongs to, and the range is stretched in 0–255;
    3. by a random RGB color: a random RGB value is generated for each cluster
    4. by a binary stack: the clustering is represented as a stack of binary images. Each binary image represents a cluster, each pixel shows a hard cluster membership. Thus it is possible to extract cluster regions from the original image by performing an AND operation between a slide of the stack and the original image.
    5. by using a fuzzy stack: a stack of gray-level images is used to show the membership values of each pixel to each cluster. Each pixel represents the soft cluster membership value of that pixel in the original image according to the currently selected cluster.

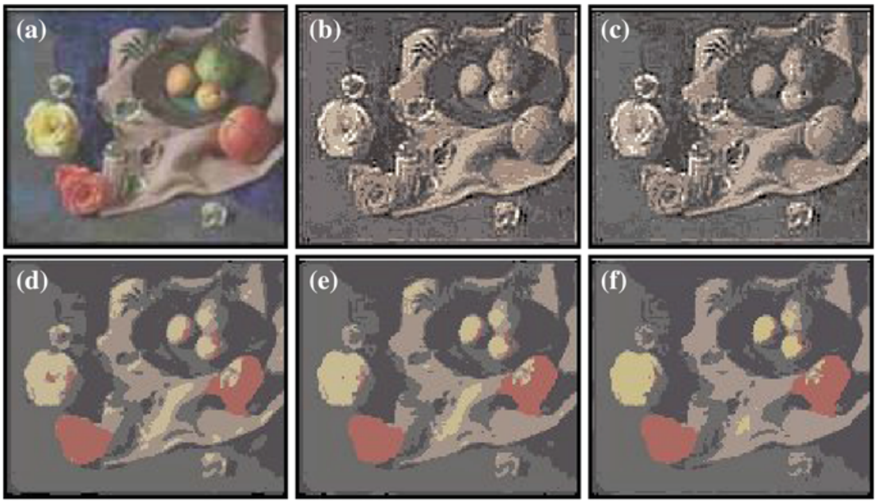


Fig. 7.1 **a** Original image. **b** Segmented image using *K-means* clustering with $K = 7$. **c** Segmented image using *FCM* with $K = 7$ and $m = 2$. Segmented image using *Spatial FCM* with $K = 7$, $p = 1$, $q = 2$, and $r = 2$ **d**, $r = 4$ **e** and $r = 6$ **f**

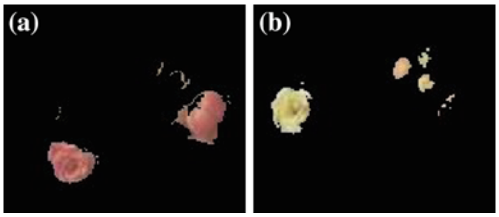


Fig. 7.2 Regions corresponding to **a** *red* and **b** *yellow* colors of the clustering results of Fig. 7.1

An example of color image is in Fig. 7.1. The RGB image has been segmented into seven clusters using the *K-means* plugin (Fig. 7.1b), the *FCM* plugin (Fig. 7.1c) and the *SFCM* plugin. It can be seen that the *SFCM* clustering reduces the number of spurious blobs, providing more homogeneous regions, by augmenting the radius of the spatial function. Figure 7.2 shows separately the regions corresponding to red and yellow color detected in the original image.

As an example in the medical domain, a brain MR image with the presence of tumor is considered (Fig. 7.3a). In this case the segmentation is aimed to detect issues of interest in the brain, including the white matter, the gray matter, the cerebrospinal fluid and the tumor region. The original MR image is a gray-level image in an RGB format such that each color component has the same value. The image has been segmented into five clusters using the *K-means* plugin (Fig. 7.3b), the *FCM* plugin (Fig. 7.3c) and the *SFCM* plugin ((Fig. 7.3d). It can be seen that the *SFCM* clustering reduces the number of spurious blobs, providing more homogeneous regions. Figure 7.4 show separately the regions corresponding to cluster number 3 and cluster number 5.

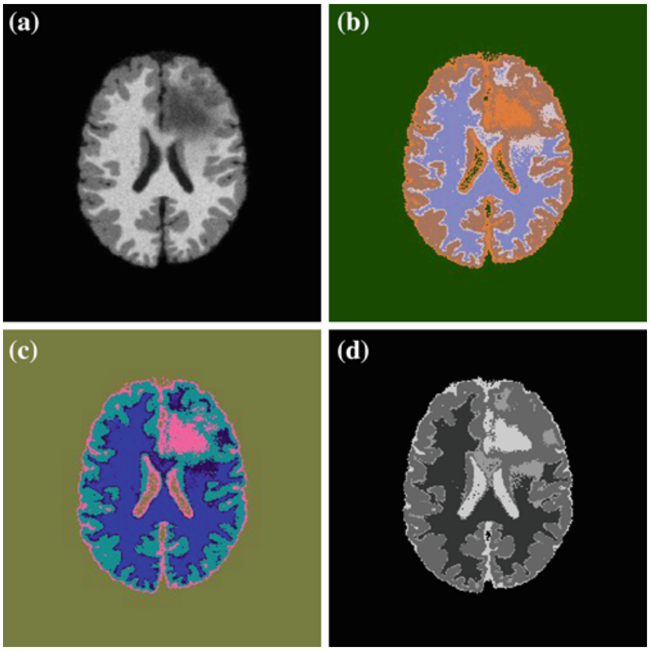


Fig. 7.3 **a** Original image. **b** Segmented image using K-means clustering with $K = 5$. **c** Segmented image using FCM with $K = 5$ and $m = 3$. **d** Segmented image using Spatial FCM with $K = 5$, $m = 2, p = 1, q = 2, r = 4$

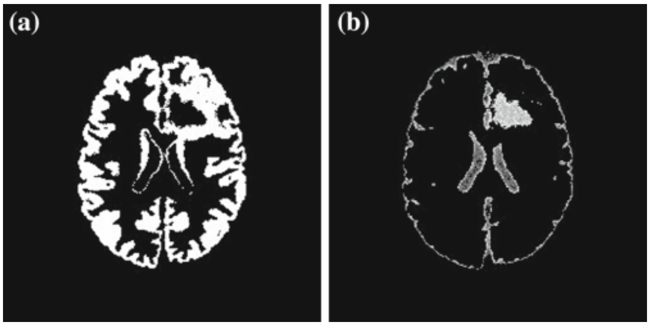


Fig. 7.4 **a** Region corresponding to cluster 3 and **b** cluster 5 of the segmentation results in Fig. 7.3d

7.5 Example: Texture Segmentation

In this section we present another application of spatial fuzzy clustering to segment a biological image on the basis of the texture information. This application is fully described in [4] where we address the problem of automatically analyzing the cytoplasm of human oocytes, in order to evaluate their quality during an assisted fertilization process. The ultimate goal of this work is to support the clinicians in the oocyte scoring by means of a system capable to derive a description of the oocyte cytoplasm in terms of different granular regions located in the cytoplasm [2]. A fundamental step in the quality assessment of the oocyte maturity is represented by the analysis of the whole cytoplasmic area in order to discover regions with different level of granularity inside the cytoplasm of a single oocyte. This analysis might highlight the presence or absence of a particular zone known as polarization or halo effect that is another important factor already studied in the literature. Hence the problem to detect different granular regions in the whole cytoplasmic area is of fundamental importance. To segment the cytoplasm region into different regions according to the granularity we used the spatial fuzzy clustering algorithm [7] applied to texture features. First, we extract of the cytoplasm region starting from the oocyte image. Then, the spatial fuzzy clustering is applied to some texture descriptors.

The preliminary step is devoted to extract the circular region that corresponds to the cytoplasm inside the oocyte. Indeed, different parts of the oocyte are visible in the image, such as the zona pellucida and the perivitelline space that enclose the cytoplasmic area. To extract the cytoplasm region representing the region of interest (ROI) we start from the assumption that the shape of the cytoplasm can be approximated by a circumference. Then the method consists of the following steps: (1) obtain the gradient image; (2) use the gradient image to obtain points that possibly belong to circumferences of different radius; (3) select the circumference that better approximates the cytoplasm boundary. In step (2) we use the Hough transform to obtain image points belonging to different circles. The Hough transform [18] is a powerful technique which can be used to isolate features of a particular shape in an image: it is most commonly used for the detection of regular curves such as lines, circles, etc. The main advantage of the Hough transform is that it is tolerant of gaps in curve descriptions and is relatively unaffected by image noise. In Fig. 7.5a we show the Hough space obtained for the image in Fig. 7.6a before circle fitting.

Once detected the cytoplasm ROI, we divide it into blocks of $w \times w$ pixels. For each block j we derive a feature vector $\mathbf{x}_j$ describing its texture according to a number of first order statistical characteristics

- mean and variance;
- a measure of relative smoothness;
- third moment, that is a measure of the symmetry of the histogram;
- a measure of uniformity, maximum for an image with all gray levels equal;
- a measure of average entropy, that equals to 0 for a constant image.

These features are computed on the first level of the multiresolution decomposition of each block. More specifically these features are evaluated on the histograms of the subbands obtained by the Haar wavelet decomposition (for more details see [8, 15]). The derived features are used for segmentation by means of spatial fuzzy clustering. Namely, we first calculate the membership function in the feature domain according to (7.4). Then, we update these values by computing the spatial function as follows:

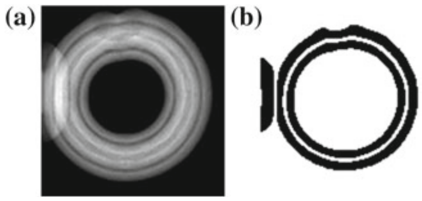


Fig. 7.5 **a** Hough space. **b** Circumferences that better approximate the cytoplasm boundary

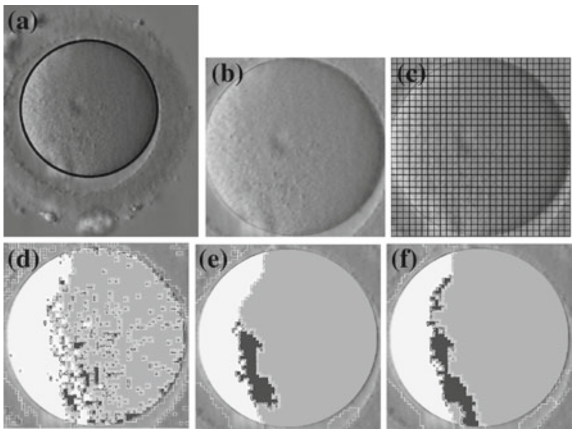


Fig. 7.6 Segmentation of a cytoplasm image. **a** Best fitting *circle* approximating the cytoplasm. **b** Detected cytoplasm ROI. **c** *Square* blocks partitioning the extracted ROI. **d** Segmentation results using FCM. **e** Segmentation results using Spatial FCM with $(p = 1, q = 1)$. **f** Segmentation results using Spatial FCM with $(p = 1, q = 2)$

$$h_{ij} = \sum_{k \in NB(b_j)} u_{ik} \tag{7.8}$$

where $NB(b_j)$ is the set of neighbors of the j-th block that is made up of the 8×8 surrounding blocks. Thus, in this case, the spatial function h_{ij} represents the membership degree of block b_j belonging to ith cluster. After application of the spatial fuzzy clustering, a defuzzification is applied to assign each block to the cluster for which the membership is maximal. As a result, our approach provides a segmented ROI, in which the regions are made up of the clusters obtained in the feature space.

Figure 7.6 show an example of the application to a light microscope images of human oocytes provided by the *Dipartimento di Endocrinologia ed Oncologia Molecolare e Clinica* of the University "Federico II" of Naples, Italy. The image has dimension 1280×960 pixels. First, the image was processed using the Hough transform, so as to detect the best circle fitting the real shape of the oocyte cytoplasm (Fig. 7.6a). This was done by searching circles of known radius R ranging from 230 to 250 pixels. In Fig. 7.5 we show the Hough space computed for the image in Fig. 7.6a.

Once the circular region of the cytoplasm was identified, a squared ROI surrounding the circular region was extracted (Fig. 7.6b). All steps involved in this phase were implemented using the *ImageJ* plugin Hough-circles [17] (see Appendix A). Then, the extracted ROI was splitted into blocks (Fig. 7.6c) and texture features were extracted from each block.

Several runs were carried out varying the block dimension (namely 8×8, 12×12, 24×24, 32×32) and the p and q parameters of the clustering algorithm. The best results were obtained with 8×8 blocks. Figure 7.6 shows the segmentation result obtained using the *FCM* algorithm ($K = 3$) and the *SFCM* algorithm applied with parameters ($p = 1, q = 1$) and ($p = 1, q = 2$). It can be seen that the conventional *FCM* can poorly segment the ROI image into three clusters; spurious blobs of one cluster appear inside other clusters. The *SFCM* drastically reduces the number of spurious blobs, hence the resulting regions are more homogeneous.

References

1. Ahmed, M.N., Yamany, S.M., Mohamed, N., Farag, A.A., Moriarty, T.: A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data. IEEE Trans. Med. Images **21**(3), 193–199 (2002)
2. Basile, T., Caponetti, L., Castellano, G., Sforza, G.: A texture-based image processing approach for the description of human oocyte cytoplasm. IEEE Trans. Instrum. Meas. **59**, 2591–2601 (2010)
3. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (1981)
4. Caponetti, L., Castellano, G., Corsini, V., Basile, T.M.: Cytoplasm image segmentation by spatial fuzzy clustering. In: Fuzzy Logic and Applications, pp. 253-260. Springer, Heidelberg (2011)
5. Chen, S., Zhang, D.: Robust image segmentation using FCM with spatial constrained based on new kernel-induced distance measure. IEEE Trans. Syst. Man Cybern. **34**(4), 1907–1916 (2004)
6. Cheng, H., Jiang, X., Sun, Y., Wang, J.: Color image segmentation: advances and prospects. Pattern Recognit. **34**, 2259–2281 (2001)
7. Chuang, K.S., Tzeng, H.L., Chen, S., Wu, J., Chen, T.J.: Fuzzy c-means clustering with spatial information for image segmentation. Comput. Med. Imag. Graph. **30**(1), 9–15 (2006)
8. Daubechies, I.: Ten lectures on wavelets. Philadelphia: Society for industrial and applied mathematics, pp. 198–202 (1992)
9. Freixenet, J., Munoz, X., Raba, D., Marti, J., Cufi, X.: Yet another survey on image segmentation: region and boundary information integration. In: European Conference on Computer Vision—ECCV 2002, pp. 408-422. Springer, Heidelberg (2002)
10. Fu, K., Mui, J.: A survey on image segmentation. Pattern Recognit. **13**, 3–16 (1981)
11. Ilea, D.E., Whelan, P.F.: Image segmentation based on the integration of colour-texture descriptors a review. Pattern Recognit. **44**(10), 2479–2501 (2011)
12. ImageJ Plugin K-means Clustering. Available at: http://ij-plugins.sourceforge.net/plugins/segmentation/k-means.html
13. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice Hall, Upper Saddle (1988)
14. MacQueen, J.B.: Some Methods for classification and Analysis of Multivariate Observations. In: Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley. University of California Press, vol. 1, 281–297 (1967)
15. Mallat, S.: A theory for multiresolution signal decomposition: the wavelet representation. IEEE Trans. Pattern Anal. Mach. Intell. **11**(7), 674–693 (1989)
16. Pal, N., Pal, S.: A review on image segmentation techniques. Pattern Recognit. **26**(9), 1277–1294 (1993)
17. Pistori H., Costa E.R.: ImageJ Plugin Hough Circles. Available at: http://rsb.info.nih.gov/ij/plugins/hough-circles.html

18. Sklansky, J.: On the hough technique for curve detection. IEEE Trans. Comput. **27**(10), 923–926 (1978)
19. Tolias, Y., Panas, S.: Image segmentation by a fuzzy clustering algorithm using adaptive spatially constrained membership functions. IEEE Trans. Syst. Man Cybern. Part A. **28**(3), 359–369 (1998)
20. Vergari, A., Tangari, F.: ImageJ Plugin Spatial Fuzzy c-Means. Available at: https://sites.google.com/site/cilabuniba/research/sfcm
21. Wang, X.Y., Bua, J.: A fast and robust image segmentation using FCM with spatial information. Digital Signal Process. **20**, 1173–1182 (2010)
22. Zhang, Y.J.: Evaluation and comparison of different segmentation algorithms. Pattern Recognit. Lett. **18**, 963–974 (1997)

Chapter 8
Morphological Analysis

As our circle of knowledge expands, so does the circumference of darkness surrounding it.

Albert Einstein

Abstract Fuzzy mathematical morphology is an extension of binary morphology to gray-scale images using techniques from fuzzy logic. Fuzzy mathematical morphology can be applied to process image data having characteristics of vagueness and imprecision. In this chapter, the main concepts from fuzzy mathematical morphology are briefly introduced and the results of applying fuzzy morphological operators to construct morphological gradient are reported in low-contrast biological images.

8.1 Mathematical Morphology

Morphology is a mathematical framework for the analysis of spatial structures. It is based on geometric, algebraic, and topological concepts as well as on set theory. The main idea lying behind mathematical morphology is assessing image geometric structures by overlapping small patterns, called *structuring elements*, in different parts of the same image.

Originally, mathematical morphology was developed for binary images and used simple concepts from set theory and geometry such as set inclusion, intersection, union, complementation, and translation. This resulted in a collection of tools, called morphological operators, which are eminently suited for the analysis of shape and structure in binary images. Thus mathematical morphology provides an approach for processing digital images which is based on shape. If properly used, morphological operations can be effective in extracting essential shape characteristics and eliminating irrelevancies.

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_8

The language of mathematical morphology is set theory. Sets in mathematical morphology represent objects in an image. For example, the set of all black pixels in a binary image is a complete morphological description of the image. These sets are members of the $2D$ integer space $\mathbb{Z}^2$ where each element of a set is a tuple whose coordinates are the coordinate (x, y) of a black pixel (or white depending on the convention) of the image. Gray-scale digital images can be represented as sets whose components are in $\mathbb{Z}^3$. In this case, two components of each element refer to the coordinates of a pixel and the third component corresponds to its discrete gray-level value.

8.1.1 Morphological Operators

Basic operations from set theory are union, intersection, complement, difference, reflection, and translation. The basic operations of mathematical morphology are dilation, erosion, opening, and closing [9, 13]. Dilation and erosion represent the primitive operations from which the others are derived. Their definition is based on operations from the set theory. These operations are the union, the intersection, the complement, the difference, the reflection, and the translation. Given an image A and a binary structuring element B, with A and B subsets in $\mathbb{Z}^2$, the morphological operators are defined as follows.

Dilation
The dilation of A by B is defined as

$$\delta_B(A) = A \oplus B = \bigcup_{y \in B} A_y = \{y \in \mathbb{Z}^2 \mid \check{B}_y \cap A \neq \emptyset\} \tag{8.1}$$

where B_y denotes the translation of B along the vector y, that is, $B_y = \{b+y | b \in B\}$ and $\check{B}$ denotes the reflection of B along the origin, that is $\check{B} = \{-b \mid b \in B\}$. In other words, the dilation of image A by element B is the union set of all translations y such that A and $\check{B}$ overlap by at least one element. To compute the dilation of a binary image A by an element B we overlap the matrix B on the image so that the origin of B corresponds to the current pixel. If the center of B corresponds to a foreground pixel of the image A then the current pixel is set to 1. The effect of this operator on a binary image is to enlarge the boundaries of the regions of foreground pixels (generally the white pixels). In Fig. 8.1c, we give an example of dilation applied to a binary image. The area of foreground (black) pixels grows in size, the gaps between regions are filled. This operator expands or inflates the black regions and fills in gaps.

Erosion
The erosion of A by B is defined as

$$A \ominus B = \{y \in \mathbb{Z}^2 \mid B_y \subseteq A\} \tag{8.2}$$

In other words, the erosion of A by B is the set of all points y such that B translated by y is contained in A. The most important relation between dilation and erosion is the *adjuction* relation, defined as

$$I \oplus B \subseteq A \Leftrightarrow I \subseteq A \ominus B \tag{8.3}$$

The adjunction relation is considered the most general as well as the most powerful duality relation between dilations and erosions. In general, dilation and erosion are not inverse operators. More specifically, if an image A is eroded by an element B and then dilated by B, the resulting set is not the original set A but a subset of it (see an example in Fig. 8.1). In most cases, this set is smaller than A. It is called the opening of A by B and denoted by $A \circ B$.

Opening
Formally, the opening of a binary image A by a binary structuring element B is defined by

$$A \circ B = (A \ominus B) \oplus B) \tag{8.4}$$

The geometric interpretation of the opening $A \circ B$ is that the opening is the union of all translations of B that are contained in A. Consequently, the binary opening will suppress small peaks and eliminate other small details. In either case, the result of iteratively applied dilations and erosions is the elimination of specific image details smaller than the structuring element used.

Closing
Dually, if A is first dilated by B and then eroded by B we get a set which contains the original set A, and in most cases, is larger than A. It is called the closing of A by B, denoted by $A \bullet B$. Formally, the closing of a binary image A by a binary structuring element B is defined as

$$A \bullet B = (A \oplus B) \ominus B \tag{8.5}$$

If we apply the definition of erosion and dilation in (8.5) we obtain a geometric interpretation of the closing $A \bullet B$, that consists of all points $y \in \mathbb{Z}^2$ for which any translation of B that contains y has a nonempty intersection with A. As a consequence, the binary closing will fill up small gaps on the contour, eliminate small holes, fuse narrow breaks and long thin gulfs, smooth the contours.

The residual of two morphological operations is their difference. The first residual that can be defined is the *morphological gradient*, which can be used as an edge detector and as a first approximation for a morphological segmentation. Finally, it is worth to note that the combination of dilation and erosion leads to other operators on binary images, such as connected components and region filling.

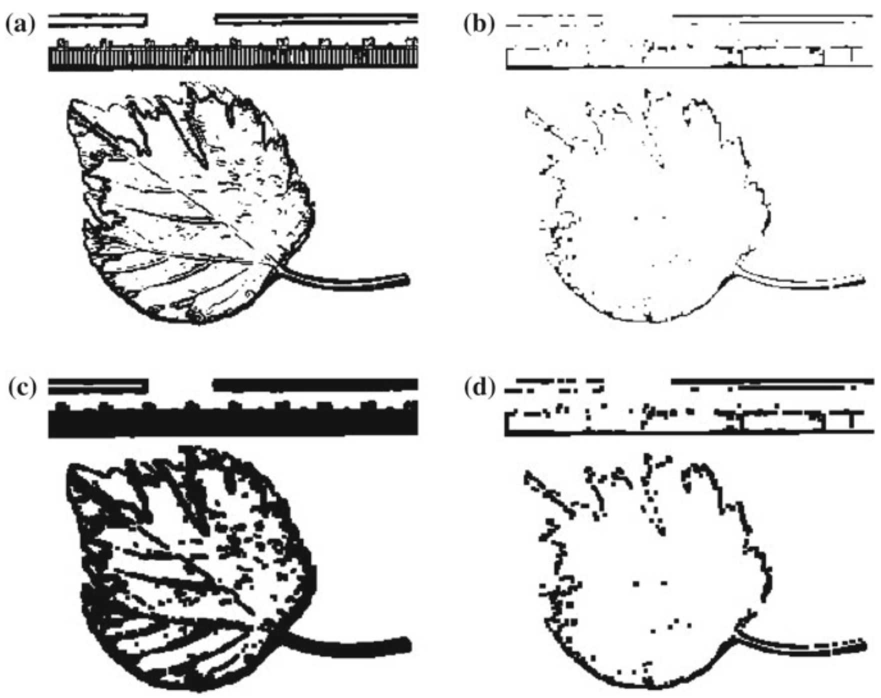


Fig. 8.1 **a** Original binary image. **b** Binary image obtained by the erosion of **a**. **c** Binary image obtained by the dilation of **a**. **d** Binary image obtained by the dilation of **b**

8.2 Fuzzy Morphology

When working with gray-level images, Boolean logic cannot be applied. As a consequence, binary morphological operators need to be extended. One possible extension is given by fuzzy mathematical morphology, which is a generalization of binary morphology using operators from the theory of fuzzy sets [2, 5, 6, 8]. Using the concept of fuzzy sets in morphology we can represent both imprecision and uncertainty from the signal level to the highest decision level [2, 7, 13].

The fuzzy operators used to build fuzzy morphological operators are conjunction and implication. The basic idea is to use fuzzy conjunctions and implications which are adjoined in the definition of dilation and erosion, respectively. Here we use the definitions given in [5], where fuzzy operators are obtained as an extension of the logical operators, i.e., the Boolean conjunction and the Boolean implication. As concerns the definition of the fuzzy conjunction $\mathscr{C}$ and implication $\mathscr{I}$ operators, several forms have been proposed [4, 6]. In the following, we report the most used definitions. Given a fuzzy set $A(x)$ where x is an element of the universe of discourse X, the *conjunction* operator C, intended as a t-norm, can be defined as

- the minimum $M(x; y) = min(x; y)$;
- the algebraic product $P(x; y) = x \cdot y$;
- the Lukasiewicz t-norm $W(x; y) = max(0; x + y1)$

The *implication* operator $\mathscr{I}$ can be defined as

- the Kleene-Dienes implicator $IKD(x; y) = max(1 - x; y)$; the Reichenbach implicator $IR(x; y) = 1 - x + x \cdot y$;
- the Lukasiewicz implicator $IL(x; y) = min(1; 1 - x + y)$

Using the extended logical operators, the Definitions (8.1) and (8.2) of binary dilation and binary erosion can be fuzzified as follows.

Let A be a gray-scale image and let B be a gray-scale structuring element, both represented as subset of $\mathbb{R}^3$. In order to model them as fuzzy sets, the values of A and B should be mapped in [0, 1] using any membership function. Once both A and B are fuzzified, we can apply the morphological operators redefined in the framework of the fuzzy sets. Let $\mathscr{C}$ be a fuzzy conjunction operator and let $\mathscr{I}$ be a fuzzy implication operator. The *fuzzy dilation* and *fuzzy erosion* of A by B are the gray-scale images defined by

$$D_{\mathscr{C}}(A, B)(y) = \sup_x \mathscr{C}(B(x - y), A(x)) \tag{8.6}$$

$$E_{\mathscr{I}}(A, B)(y) = \inf_x \mathscr{I}(B(x - y), A(x)) \tag{8.7}$$

Fuzzy closing and *fuzzy opening* are defined as in the binary case:

$$C_{\mathscr{C},I}(A, B)(y) = E_{\mathscr{I}}(D_{\mathscr{C}}(A, B), -B)(y) \tag{8.8}$$

$$O_{\mathscr{C},I}(A, B)(y) = D_{\mathscr{C}}(E_{\mathscr{I}}(A, B), -B)(y) \tag{8.9}$$

where the reflection of the gray-scale structuring element B is defined by $-B(x) = B(-x), \forall x \in \mathbb{R}^2$.

As concerns the definition of the fuzzy conjunction $\mathscr{C}$ and implication $\mathscr{I}$ operators, several forms have been proposed [2, 7]. In the following, we define the most used definitions. The *conjunction operator* $\mathscr{C}$, intended as a t-norm, can be defined as

- the minimum $M(x, y) = \min(x, y)$;
- the algebraic product $P(x, y) = x \cdot y$;
- the Lukasiewicz t-norm $W(x, y) = \max(0, x + y - 1)$

The *implication operator* $\mathscr{I}$ can be defined as

- the Kleene-Dienes implicator $I_{KD}(x, y) = \max(1 - x, y)$;
- the Reichenbach implicator $I_R(x, y) = 1 - x + x \cdot y$;
- the Lukasiewicz implicator $I_L(x, y) = \min(1, 1 - x + y)$

Table 8.1 Different types of fuzzy dilation/erosion operators

Operators	$\mathscr{C}$	$\mathscr{I}$	Dilation $D(A, B)(y)$	Erosion $E(A, B)(y)$
Kleene-Dienes	M	I_{KD}	$\sup_x \min(B(x-y), A(x))$	$\inf_x \max(1 - B(x-y), A(x))$
Reichenbach	P	I_R	$\sup_x (B(x-y)A(x))$	$\inf_x (B(x-y)A(x) + 1 - B(x-y))$
Lukasiewicz	W	I_L	$\sup_x \max[0, B(x-y) + A(x) - 1]$	$\inf_x \min[1, 1 - B(x-y) + A(x)]$

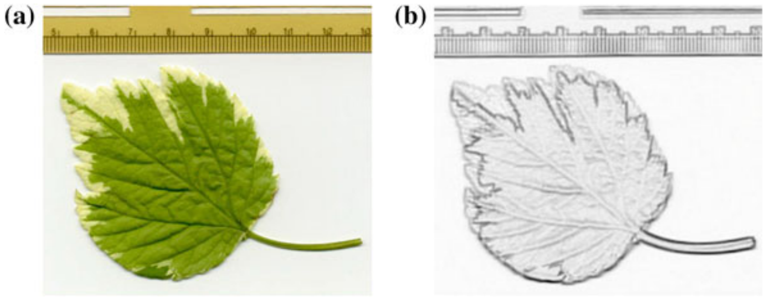


Fig. 8.2 **a** Original color image. **b** Image obtained by applying the morphological gradient to each RGB component of **a**

Combining the above definitions of $\mathscr{C}$ and $\mathscr{I}$ in (8.6) and (8.7) we obtain the different types of fuzzy dilation/erosion, namely Kleene-Dienes operators, Reichenbach operators, Lukasiewicz operators, as described in Table 8.1.

Morphological gradient
The combination of dilation and erosion lead to some operators on gray-level images, such as image smoothing and gradient. In particular, the morphological gradient, known as the Beucher gradient [1], is the difference between a dilation and an erosion, or between a dilation and the original image or between the original image and its erosion. The application of the morphological gradient enhances variations of pixel intensity in a given neighborhood, thus it is typically used for edge detection and segmentation. The fuzzy morphological gradient is defined using fuzzy dilation and fuzzy erosion:

$$\nabla_{\mathscr{C},\mathscr{I}}(A, B) = A - E_{\mathscr{I}}(A, B),$$

$$\nabla_{\mathscr{C},\mathscr{I}}(A, B) = D_{\mathscr{C}}(A, B) - A,$$

$$\nabla_{\mathscr{C},\mathscr{I}}(A, B) = D_{\mathscr{C}}(A, B) - E_{\mathscr{I}}(A, B).$$

where $\mathscr{C}$ is a conjunction operator and $\mathscr{I}$ an implication operator.

This operator highlights sharp gray-level transition in the input image (Fig. 8.2). It provides a suitable mean to derive and represent the intuitive concept of soft edges of objects into images. In the following section, we show how to use the fuzzy morphological gradient to extract soft edges in biological images.

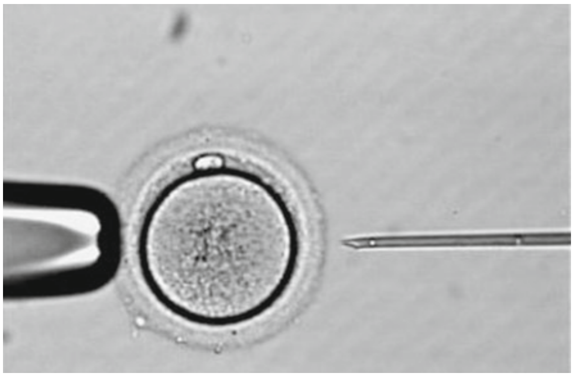


Fig. 8.3 A sample biological image

Morphological smoothing
Image smoothing can be achieved by applying opening followed by a closing. The results of these two operations is to attenuate both bright and dark artifacts.

8.3 Example: Biological Image Segmentation

In this section, we show how to apply morphological operators in order to segment an oocyte image. Given a microscope image of an oocyte, the problem faced here is to separate the oocyte region from other elements (background, injection pipette, and so on) so as to facilitate the extraction of significant regions from the oocyte, such as the cytoplasm and the zona pellucida. Figure 8.3 shows an example of oocyte image containing useless elements to be removed.

This process is a crucial step in the analysis of oocyte maturity which may be directly related to the success rate of Intra Cytoplasmic Sperm Injection (ICSI) [10, 14] technologies. There are currently some recognized parameters assessing the quality of oocytes, intended as its maturity: the oocyte and cytoplasm diameter, its granularity (whether it is central or on border, uniform or not), the presence of cytoplasmic refractive granules and vacuoles and their size, zona pellucida thickness, first polar globe diameter and perivitelline space dimension with respect to the first polar globe, the spindle birefringence and the extracellular dysmorphisms [3, 14]. Hence, the possibility to extract the oocyte from the image for further processing steps is of fundamental importance in this medical application domain.

The problem of oocyte region extraction in [4] is addressed as a problem of removing borders touching objects using morphological reconstruction operators. Indeed regions connected to the border of the oocyte image are not relevant for subsequent image analysis. Furthermore, it may occur that more than one oocyte is contained in the image, hence the segmentation process should be able to correctly separate the regions of different oocytes present in the image to enable further analysis steps.

In order to extract the oocyte region from the entire image in [4] we propose an approach consisting of two main steps: a soft edge detection based on fuzzy morphological gradient, followed by a morphological reconstruction phase. It is worth to note that in order to simplify the morphological reconstruction step, the resulting fuzzy edge image is binarized. Since the result of the fuzzy morphological gradient is a fuzzy set, we can apply a fuzzy thresholding method based on minimizing the measure of fuzziness [11].

Fuzzy morphological gradient.
This processing consists of finding edges in the image, by means of morphological gradient and successively binarize the result.

The first step is to fuzzify the image. As explained in Sect. 8.2 this can be done using different membership functions, depending on the preprocessing goal. In our case, we use the very simple N-function (see Eq. 4.1) that performs just a normalization:

$$N(x) = \begin{cases} 1 - (b - x)/w \text{ if } (b - w) \leq x \leq b \\ 1 \quad \text{if } x > b; \quad 0 \quad \text{if } x < (b - w) \end{cases} \tag{8.10}$$

where b is the maximum value in the image and $w = b - a$ defines the bandwidth of the value range.

The fuzzy structuring element B used was built as

$$B(i, j) = 1 - (1/4) * (i^2 + j^2)^{1/2}$$

where i and j are the coordinates of the entry relative to the center of the structuring element. On such fuzzified image, we calculate the fuzzy morphological gradient:

$$\nabla_{\mathscr{C},\mathscr{I}}(A, B) = D_{\mathscr{C}}(A, B) - E_{\mathscr{I}}(A, B)$$

Finally, the resulting soft edge image is binarized by means of a fuzzy thresholding algorithm [11].

Morphological reconstruction
After the binarization, elements that are not of interest surrounding the image borders, such as the holding pipette and the injection pipette, have to be taken out.

In order to eliminate irrelevant elements surrounding the oocyte border, such as the holding pipette and the injection pipette, we apply a reconstruction operator based on mathematical morphology. In particular, we perform the reconstruction of the intersection of the image border with a thick box boundary to eliminate particles touching the left and right borders of the image that are 4-connected. The extraction of connected components algorithm [9] is exploited, that is based on dilation and intersection. In the following, we briefly outline such algorithm.

Let X be a connected component[1] contained in a region A and let B be a structuring element. The algorithm operates by first selecting a point p in the region A belonging

[1]Two pixels are connected in a subset S of an image A if there exists a path between them made up of pixels belonging to S. The largest set of pixels connected to the pixel $p \in S$ is known as connected component of S.

to the initial connected component X_0 and, starting from it, performs the following iterative procedure:

$$X_k = (X_{k-1} \oplus B) \bigcap A \quad k = 1, 2, 3, \ldots \ and \ \ X_0 = p$$

The procedure stops when $X_k = X_{k-1}$ and then X is defined equal to the connected component X_k. In this way, if we select as initial pixel p a pixel of the image border—left or right—any object that is connected to the border of the image can be individuated as a connected component and successively removed. If we assume that all foreground points are labeled as 1, then the intersection with A at each iterative step eliminates dilations centered on elements labeled as 0.

In many cases the obtained image still presents holes in the interior of the object. These holes can be filled using a filling algorithm [9]. The goal of such an algorithm is to fill the region of the image by considering a hole as a background region surrounded by a connected border of foreground pixels. It is based on dilation, complementation, and intersection. First, a point p is selected in the region to fill, then, starting from that point, all pixels belonging to the region are set as foreground. Finally, by subtracting the obtained mask from the original image, the region of the oocyte is achieved on which a bounding oval region is detected. The complete procedure for oocyte region extraction is formalized in the following:

```
Require: J =8-bit gray-level image;
Ensure: R = the region that contains the oocyte;
1. J1= J; /*steps performed on J1*/
2. Apply morphological gradient;
3. Apply fuzzy thresholding;
  /* Morphological reconstruction */
4. Apply Kill Borders on the right and left borders;
5. Apply two dilation steps;
6. Fill Holes;
7. Open-Close;
8. J2= J1-J;
  /* Oocyte region extraction */
9. Trace the rays starting from the center mass of J2 in the
   directions defined by the 8-neighbor of that point.
10. Compute the four Euclidean distances (w, n, s, e) such as
    intersections with the rays, traced, respectively, in the
    directions West, North, East, South and the image J2;
11. Compute the oocyte diameter and overlap the circumference on
    the image J;
12. R = bounding rectangular region of J centered on the mass of
    J2 with width (w + e) and height (n + s).
```

This procedure has been implemented as a macro of *ImageJ*. The fuzzy operators have been applied using a modified version of the plugins available at [12]. To give some illustrative examples, the procedure was applied to segment some microscope images

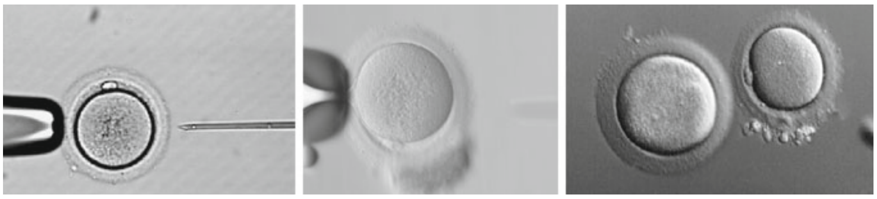


Fig. 8.4 Problematic cases requiring the detection of soft edges

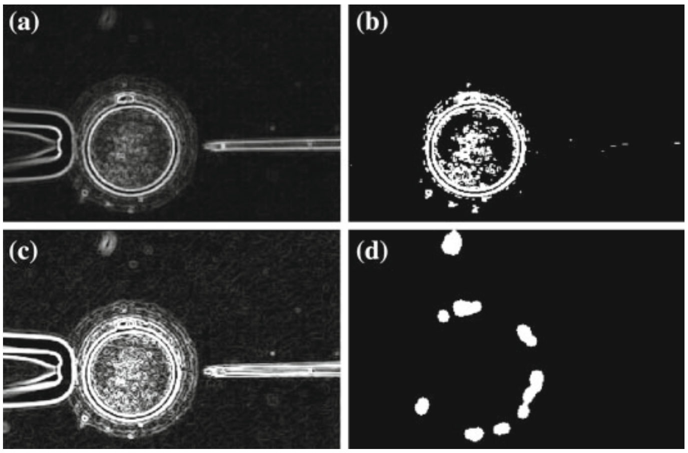


Fig. 8.5 Removal of connected components for the image in Fig. 8.3: **a** binary image obtained by fuzzy thresholding applied to the fuzzy gradient image; **b** image obtained after removing components connected to borders of image (**a**); **c** binary image obtained by thresholding the Sobel gradient image; **d** image obtained after removing components connected to the borders of image (**c**)

of human oocytes, provided by the *Dipartimento di Endocrinologia ed Oncologia Molecolare e Clinica* of the University "Federico II" of Naples, Italy. Images were acquired by means of the Nikon Eclipse TE200 Inverted Microscope.

First, we show the need of exploiting fuzzy edge detection instead of a crisp edge detection such as the Sobel operator. Indeed, strong edges do not allow to perfectly delineate the limits between regions that are not well separated. There are, for example, biological images containing more than one cell that are very close to each other or images containing the pipettes (injection and/or holding) that are very close to the oocyte (see Fig. 8.4 for some examples). A correct segmentation of such critical images is not possible using a crisp edge detection. This limitation is overcame by removing connected components after application of a soft edge detection, as explained in the previous section.

Figure 8.5 shows good results obtained by applying the soft edge detection to the sample image of Fig. 8.3 compared with the wrong behavior of the Sobel operator. In the image obtained by Sobel, not only the injection and the holding pipette have

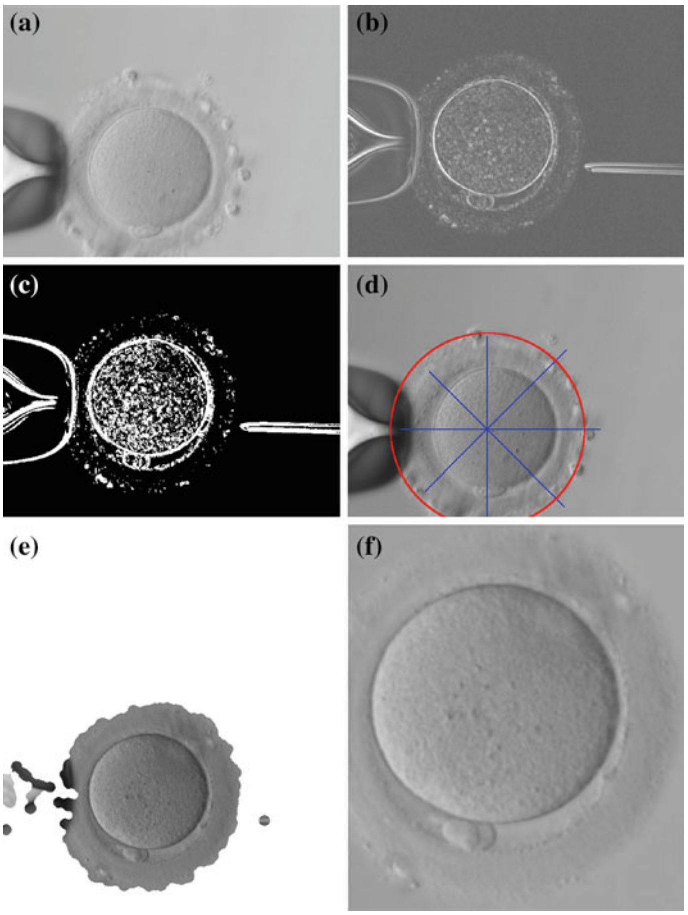


Fig. 8.6 **a** Original image; **b** Fuzzy gradient image; **c** binary image; **d** diameter oocyte; **e** final reconstruction; **f** detected oocyte region

been removed, but also a significant part of the oocyte region. Another example is shown in Fig. 8.6 that depicts the results of each step of the procedure.

The choice of the fuzzy (erosion/dilation) operators can influence the final result. In Fig. 8.7 we show the different result obtained by the Kleene-Dienes, Reichenbach, and Lukasiewicz operators on a sample image. It can be seen that Kleene-Dienes operators provide softer edges with respect to the other operators. On the overall, results show that the Kleene-Dienes operators outperform the other operators in this application domain. This behavior is confirmed by the trend of the relative histograms reported in Fig. 8.8.

Figure 8.9 show the results obtained on a sample images using Kleene-Dienes operators in the fuzzy dilation and erosion definitions. The outcomes show that our

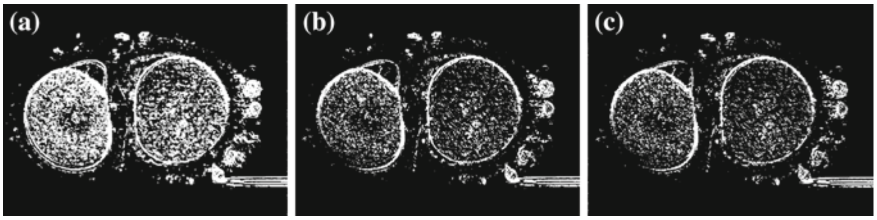


Fig. 8.7 Binary image obtained by exploiting **a** Kleene-Dienes. **b** Reichenbach. **c** Lukasiewicz operators

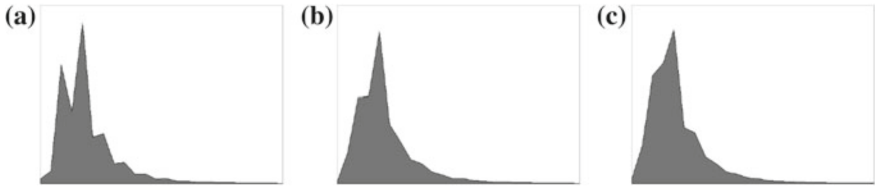


Fig. 8.8 Histograms of the fuzzy morphological gradient obtained by exploiting. **a** Kleene-Dienes. **b** Reichenbach. **c** Lukasiewicz operators

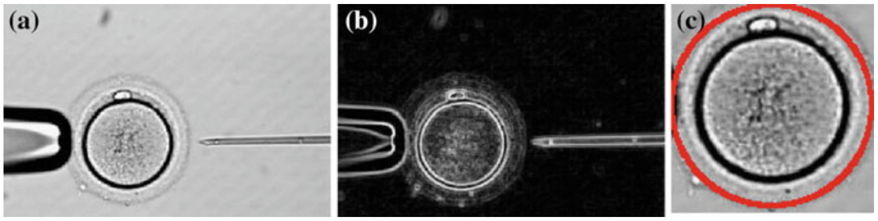


Fig. 8.9 **a** Original image. **b** Fuzzy morphological gradient. **c** Detected oocyte region

approach can well address the problems of removing the elimination of useless parts in the image (holding and injection pipettes).

References

1. Beucher, S.: Segmentation d'images et morphologie mathmatique. Doctoral dissertation, Ecole Nationale Suprieure des Mines de Paris (1990)
2. Bloch, I., Maytre, H.: Fuzzy mathematical morphologies: a comparative study. Pattern Recognit. **28**, 1341–1387 (1995)
3. Caponetti, L., Castellano, G., Corsini, V., Sforza, G.: Multiresolution texture analysis for human oocyte cytoplasm description. In: Proceedings of the MeMeA09, pp. 150–155 (2009)
4. Caponetti, L., Castellano, G., Basile, M.T., Corsini, V.: Fuzzy mathematical morphology for biological image segmentation. Appl. Intell. **40**(1), 1–11 (2014)
5. De Baets, B.: Fuzzy morphology: A logical approach. In: Uncertainty Analysis in Engineering and Science: Fuzzy Logic, Statistics, and Neural Network Approach, pp. 53–68. Kluwer

Academic Publishers, Norwel (1997)

6. De Baets, B., Kerre, E.E., Gupta, M.: The fundamentals of fuzzy mathematical morphologies part i: basics concepts. Int. J. Gen. Syst. **23**, 155–171 (1995)
7. Deng, T., Heijmans, H.: Grey-scale morphology based on fuzzy logic. J. Math. Imaging Vis. **16**(2), 155–171 (2002)
8. di Gesú, V., Maccarone, M.C., Tripiciano, M.: Mathematical morphology based on fuzzy operators. Fuzzy Logic 477–486 (1993)
9. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Prentice-Hall Inc, Upper Saddle River, NJ, USA (2006)
10. Hamamah, S.: Oocyte and embryo quality: is their morphology a good criterion?. Journal de gynecologie, obstetrique et biologie de la reproduction. **34**(7 Pt 2), 5S38–5S41 (2005)
11. Huang, L.-K., Wang, M.-J.J.: Image thresholding by minimizing the measure of fuzziness. Pattern Recognit. **28**(4), 41–51 (1995)
12. Landini G.: ImageJ plugin Morphology. Available at: http://sites.imagej.net/Landini/
13. Serra, J.: Image Analysis and Mathematical Morphology. Academic Press Inc., Cambridge (1983)
14. Wilding, M., Di Matteo, L., DAndretti, S., Montanaro, N., Capobianco, C., Dale, B.: An oocyte score for use in assisted reproduction. Journal of Ass. Repr. Gen. **24**(8), 350–358 (2007)

Chapter 9
Image Thresholding

Logic will get you from A to B. Imagination will take you everywhere.

Albert Einstein

Abstract This chapter deals with the methods for image thresholding. It introduces some basic concepts such as two-level and multilevel thresholding. The chapter also describes one of the main classical nonfuzzy thresholding methods—the Otzu method—and presents the Huang method based on minimization of fuzzy entropy. An application to document image analysis, using fuzzy techniques for segmentation and neuro-fuzzy system for classification is provided.

9.1 Introduction

Thresholding is a technique widely used in image segmentation. The objective of histogram thresholding is to determine a threshold value to partition the image space into meaningful regions. For example, thresholding is a necessary step in many image processing tasks such as automatic recognition of machine printed or handwritten texts, recognition of object shapes, and image enhancement. A thresholding process may be applied to values representing gray levels, or edge or properties such as average or texture.

Two-level thresholding segments pixels of an image into two regions, where one region contains pixels with gray values smaller than the threshold value and the other one contains pixels with gray values greater than the threshold value. Precisely, fixed a value T, a two-level thresholding transforms a gray-level image into a binary image $f'(x, y)$ so that

$$\text{if}\, f(x, y) \geqslant T \text{ then}\, f'(x, y) = 1 \text{ else}\, f'(x, y) = 0$$

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6_9

When the image is segmented into more than two regions, this is called multilevel thresholding. Generally in this case, an image $f(x, y)$ is transformed into an image $f'(x, y)$ having a restricted number of gray levels. Let D_i be a subset of gray levels, a multilevels thresholding transforms a gray-level image into a binary image $f'(x, y)$ so that

$$\text{if } f(x, y) \in D_i \text{ for } i = 1 \ldots r \text{ then } f'(x, y) = i \text{ else } f'(x, y) = 0$$

In general, the threshold is located at the deep valley of the image histogram and the thresholding results depend on the depth and the width of the valleys separating the modes of the histogram. However, when the valley is not so obvious, it is very difficult to determine the threshold. During the past decade, many research studies have been devoted to the problem of selecting the appropriate threshold value. A survey can be found in [12, 18].

Generally, the regions in an image may be ill-defined because of image data ambiguity, due to textured background, information noise, and so on. For this reason, some approaches based on fuzzy set theory have been proposed. In [6], a method has been described based on the minimization of the image fuzziness evaluated in the intensity and spatial domain. In [16] classes of fuzzy entropies are constructed which are useful for image thresholding based on cost minimization.

In the following, we describe one of the main classical nonfuzzy methods—the Otzu method—and some methods based on minimization of fuzzy measures [10]. Finally an application from [3] is presented.

9.2 Otzu Method

The Otzu method performs automatic selection of the threshold T. To introduce the method, we give first some notations.

Let f be an image of $M \times N$ pixels with L gray levels. Let n_i be the number of pixels having intensity i and $p_i = n_i/MN$ its occurrence frequency. Fixed a threshold value $T(k) = k$ for $0 < k < L - 1$, the image can be divided into two classes:

- C_1 containing all the pixels having intensity in the range $[0, k]$
- C_2 containing all the pixels having intensity in the range $[[k + 1, L - 1]$

The probability that a pixel value is assigned to the class C_1 called *occurrence probability* of the class C_1 is defined as a cumulative sum:

$$P_1(k) = \sum_{i=0}^{k} p_i \tag{9.1}$$

The occurrence probability of class C_2 is

$$P_2(k) = 1 - P_1(k)$$

The mean intensity value of the pixels assigned to classes C_1 and C_2 are

$$m_1(k) = \frac{1}{P_1(k)} \sum_{i=0}^{k} i p_i$$

$$m_2(k) = \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} i p_i$$

The cumulative mean up to level k is given by

$$m_c(k) = \sum_{i=0}^{k} i p_i \tag{9.2}$$

and the average intensity of the entire image, i.e., the global mean, is given by

$$m = \sum_{i=0}^{L-1} i p_i \tag{9.3}$$

Given the above equations, the validity of the following two equations can be easily verified: $P_1(k)m_1(k) + P_2(k)m_2(k) = m$ and $P_1(k) + P_2(k) = 1$.

To evaluate the quality of the threshold at level k the following normalized metric is considered:

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma^2} \tag{9.4}$$

where σ^2 is the is the intensity variance of all the pixels in the image

$$\sigma^2 = \sum_{i=0}^{L-1} (i - m)^2 p_i$$

and σ_B^2 is the between-class variance defined as

$$\sigma_B^2(k) = P_1(k)(m_1(k) - m)^2 + P_2(k)(m_2(k) - m)^2 \tag{9.5}$$

Equation (9.5) can be rewritten as

$$\sigma_B^2(k) = P_1(k)P_2(k)(m_1(k) - m_2(k))^2 = \frac{(mP_1(k) - m_c(k))^2}{P_1(k)(1 - P_1(k))} \tag{9.6}$$

The first expression in (9.6) shows that more distant the two means m_1 and m_2 are from each other, larger σ_B^2 will be. This indicates that the between-class variance measures the separability between classes. The computation of the second expression in (9.6) is more efficient since the global mean m is computed only once, so only two parameters m_c and P_1 need to be computed for any value of k.

Main peculiarities of the Otzu method are its efficiency since it operates directly on the gray-level histogram (e.g., a mono-dimensional array of 256 numbers), so it is fast once the histogram is computed.

The main step of the Otzu method can be summarized as follows:

---

1. Compute the components of the normalized histogram p_i, for $i = 1 \dots L - 1$.
2. Compute $P_1(k)$ for $k = 0 \dots L - 1$ according to (9.1)
3. Compute $m_c(k)$ for $k = 0 \dots L - 1$ according to (9.2)
4. Compute the global mean m according to (9.3).
5. Compute $\sigma_B^2(k)$ for $k = 0 \dots L - 1$ according to (9.6).
6. Obtaining the threshold k^* as the value of k for which $\sigma_B^2(k)$ is maximum. If the maximum is not unique, obtain k^* by averaging the values of k corresponding to the maxima.
7. Obtain the separability measure η^* by evaluating (9.4) at $k = k^*$.

Moreover, the Otzu method is optimal in the sense that it can find the threshold that maximizes the interclass variance. Figure 9.1 shows an example of thresholding results, obtained using the plugin [13].

9.3 Fuzzy Thresholding

Fuzzy interpretation of data images is based on the assumption that an $M \times N$ image f of L gray levels can be associated to an array X of fuzzy singletons $\mu_{mn} = \mu(f(m, n))$, each singleton denoting the membership of $f(m, n)$ to the set of gray levels.

Here we describe the fuzzy thresholding approach proposed in [10], where the membership function $\mu(f(m, n))$ is a characteristic function that represents the fuzziness of pixel (m, n) in X. For the purpose of image thresholding, each pixel in the image should possess a close relationship with its belonging region: the object or the background. Hence, the membership value of a pixel in X can be defined using the relationship between the pixel and its belonging region. Let $h(g)$ denote the number of occurrences of the gray level g in the image. Given a certain threshold value t, the average gray level of the background, denoted by μ_0, and the average gray level of the object, denoted by μ_1, can be obtained as follows:

$$\mu_0 = \frac{\sum_{g=0}^{t} g h(g)}{\sum_{g=0}^{t} h(g)} \tag{9.7}$$

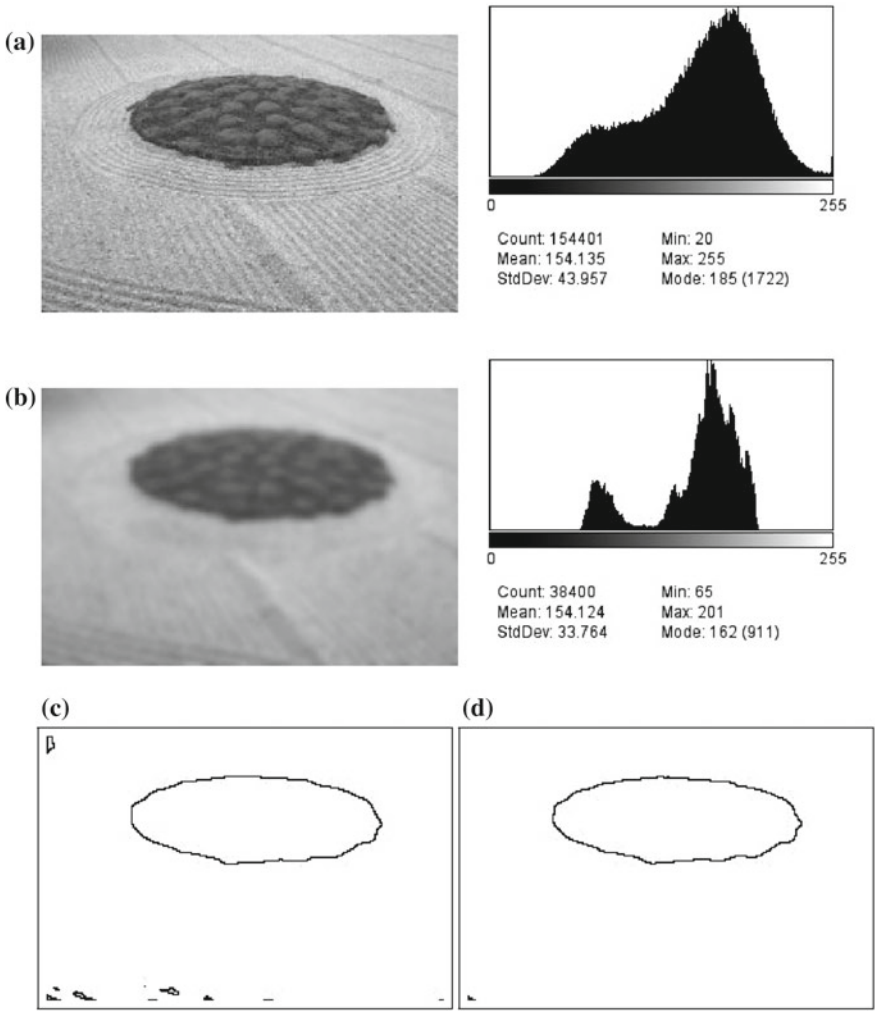


Fig. 9.1 **a** A *gray-level* image and its histogram. **b** Gaussian smoothed image and its histogram. **c** Binary image resulting from Otsu thresholding with T = 129. **d** Binary image resulting from the fuzzy thresholding

$$\mu_1 = \frac{\sum_{g=t+1}^{L-1} gh(g)}{\sum_{g=t+1}^{L-1} h(g)} \tag{9.8}$$

The values μ_0 and μ_1 can be considered as the target values of the background and the object for the given threshold value t. The relationship between a pixel in X and its belonging region should intuitively depend on the difference of its gray level and the target value of its belonging region. Thus, let the relationship possess the property that the smaller the absolute difference between the gray level of a pixel and its corresponding target value is, the larger membership value the pixel has. Hence, the membership function evaluating the above relationship for the pixel (m, n) can be defined as

$$\mu_X(x_{mn}) = \frac{1}{1 + \frac{|x_{mn} - \mu_0|}{C}} \text{ if } x_{mn} \leqslant t \tag{9.9}$$

$$\mu_X(x_{mn}) = \frac{1}{1 + \frac{|x_{mn} - \mu_1|}{C}} \text{ if } x_{mn} \geqslant t \tag{9.10}$$

where C is a constant value such that $\frac{1}{2} \leqslant \mu_X(x_{mn}) \leqslant 1$. Given a threshold t, any pixel in the image should belong to either the object or the background region. Hence, it is expected that the membership value of any pixel should be no less than $\frac{1}{2}$. The membership function really reflects the relationship of a pixel with its belonging region. As introduced in Chap. 4, a measure of fuzziness [5] indicates the degree of fuzziness of a fuzzy set. It is a function that associates to the fuzzy set A a value representing its degree of fuzziness. For a given image set X, the measure of fuzziness is expected to be as small as possible. Hence, the main purpose is to select an appropriate threshold value so as to minimize the fuzziness measure of X. Following [10], here we consider the Yager's fuzziness measure previously defined in (4.8) and based on the following distance:

$$D_p(X, \bar{X}) = \left[\sum_m \sum_n |\mu_{mn} - (1 - \mu_{mn})|^p \right]^{1/p} \text{ for } p = 1, 2, 3, \ldots \tag{9.11}$$

For convenience, the following variables are introduced:

$$S(t) = \sum_{g=0}^{t} h(g), \qquad \bar{S}(t) = \sum_{g=t+1}^{L-1} h(g) \qquad \text{and} \qquad \bar{S}(L-1) = 0$$

$$W(t) = \sum_{g=0}^{t} gh(g), \qquad \bar{W}(t) = \sum_{g=t+1}^{L-1} gh(g) \qquad \text{and} \qquad \bar{W}(L-1) = 0$$

where in the Eqs. 9.1 and 9.2 the occurrence p_i has been evaluated with the histogram $h(i)$.

The algorithm for Huang’s fuzzy thresholding is the following:

```
Require: The input image, with gray-level range $[g_{min}, g_{max}]$.
Ensure: The optimal threshold value.
Set the parameter $p$ in (9.11). Then, calculate $S(L-1)$ and $W(L-1)$ for the input
image. Given the threshold value $t = g_{min}$, let $S(t-1) = 0$ and $W(t-1) = 0$.
Repeat from step 1 to step 3 until $t = g_{max} - 1$.
1. Compute
    $S(t) = S(t-1) + h(t), \quad S(t) = S(L-1) - S(t),$
    $W(t) = W(t-1) + txh(t), \quad \bar{f}(t) = W(L-1) - W(t)$
    The average gray levels of the background and the object are, respectively,
    obtained by
        $\mu_0 = int\left[\frac{W(t)}{S(t)}\right]; \quad \mu_1 = int\left[\frac{\bar{f}(t)}{\bar{S}(t)}\right]$
    where $int[x]$ takes the integer value near the real $x$.
2. Compute the measure of fuzziness of the input image using (4.8).
3. Set $t = t + 1$
4. Find the minimum measure to determine the optimal threshold value.
```

Sometimes, the threshold value located by minimizing the measure of fuzziness is not necessarily the deepest valley between two peaks. To make sure that the threshold locates at the real valley, a fuzzy range is defined such that the measures within the range are equal to or less than a tolerance $\delta = min + (max - min)\alpha\%$ where α is a specified value $0 \leq \alpha \leq 100$ and *min* and *max* are the minimum and the maximum measure of fuzziness, respectively. Using the fuzzy range, we can further determine an improved threshold t^*, which is the best located one in the deep valley of the gray-level histogram. In other words, the threshold t^* can be obtained according to the following equation

$$\min_g[h(g-1) + h(g) + h(g+1)],$$

where g belongs to the fuzzy range. Theoretically, the threshold t^* should have a better chance of being located at the real valley than the threshold obtained by minimizing the measure of fuzziness, and it should have a better thresholding result in practice.

A Java plugin implementing Huang’s fuzzy thresholding is available at [11].

9.4 Example: Document Image Analysis

Extraction of the structure or the layout from a document is referred to as document analysis. Mapping the layout structure into a logical structure is referred to as document understanding. Document analysis and understanding are relevant tasks for the automatic processing of paper documents [14], enabling the recognition of document contents and the simplification of a number of complex tasks, such as reediting, storage, maintenance, retrieval, and transmission.

In this section, we focus on document image analysis that provides techniques for partitioning a document into a hierarchy of physical components such as pages, columns, paragraphs, words, tables, figures, halftones, etc. [15]. In this context, segmentation of document pages into coherent regions containing homogeneous information such as text, graphics, pictures or background, can be seen as a preliminary phase in the construction of the physical and logical layout. Once the regions of a document image have been classified, more specific techniques can be applied. For example, the text regions can be separated in columns, paragraphs, text lines, words, and characters; then the individual words or characters may be converted into a character code like ASCII. The graphics regions, such as line drawings, can be further decomposed in primitives such as strength lines, curve segments, and so on, and then interpreted.

Several methods for document image segmentation and classification have been developed [14, 15]. Here we describe an approach designed to solve two document analysis tasks: segmentation of a document in connected components and classification of each component as text region or graphics region. More specifically the approach, presented in [2, 3, 7], uses a fuzzy technique for segmentation and a neuro-fuzzy system for classification.

9.4.1 Document Segmentation

A fuzzy technique is applied to segment a document image into homogeneous regions, also called connected components. Precisely, given a document image, we initially compute the fuzzy gradient and then we apply the Huang's fuzzy thresholding described above to the obtained image. Finally the flood-filling operator is applied to the image resulting from thresholding.

The main steps of the procedure are the ones introduced in Sect. 8.3 for oocyte image segmentation. The procedure for the extraction of connected components from a document image can be summarized as follows:

```
Require: f = 8-bit gray-level image;
Ensure: R = set of connected components;
/*steps performed on f*/
1. Apply fuzzy morphological gradient;
2. Apply fuzzy thresholding;
3. Apply dilation;
4. Fill Holes;
5. Open-Close;
```

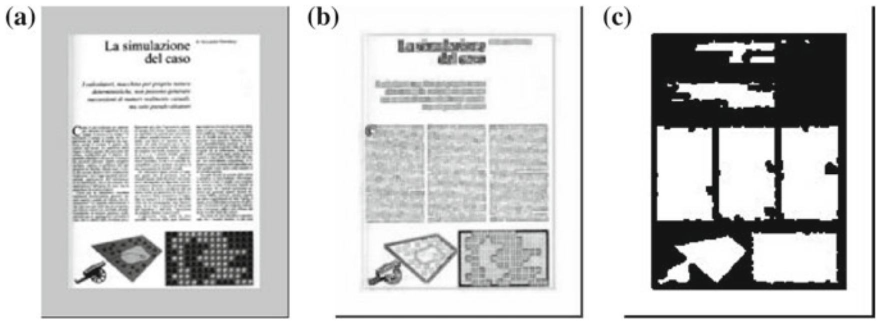


Fig. 9.2 **a** Original document image. **b** Image resulting from the inversion of the fuzzy gradient image. **c** Extracted connected components

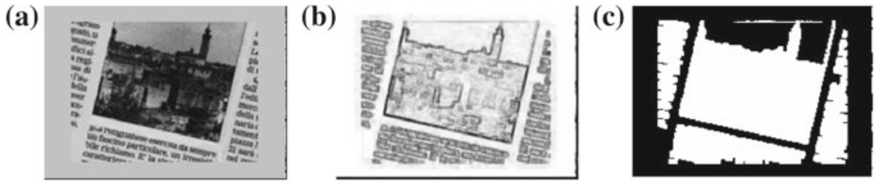


Fig. 9.3 **a** Original document image. **b** Image resulting from the inversion of the fuzzy gradient image. **c** Extracted connected components

Figures 9.2 and 9.3 show the result of applying the above procedure to segment two sample document images.

9.4.2 Region Classification

The idea at this stage is to exploit a neuro-fuzzy network to classify each component (or region) into text, graphics, or background. As described in Sect. 3.4.2, a neuro-fuzzy model is a fuzzy rule-based model whose rules are automatically defined by means of a neural network training. Here a neuro-fuzzy network is employed to learn fuzzy classification rules that enable recognition of text and no-text (graphics) regions inside the segmented document.

Features Extraction
In order to create the training set for the neuro-fuzzy learning, the Document Image Database available from the University of Oulu [20] is considered. This database includes different pages scanned from magazines, newspapers, books, and manuals. Each image is processed to extract connected components as explained above. Then, for each region, a feature extraction process is performed in order to describe the presence (or absence) of text lines in that region. It consists in detecting the skew

angle ϕ of the region as the dominant orientation of the straight lines passing through the region. Inside the text regions, being composed of characters and words, the direction of the text lines will be highly regular. The dominant orientation of the text lines determines the skew angle.

This regularity can be captured by means of the Hough transform [8, 9]. The Hough transform maps each point of the original plane to all points of the Hough plane describing for each point the possible lines through that point with slope and distance from origin. The dominant lines are found from peaks in the Hough space and thus the orientation. Particularly, the skew angle is detected as the angle for which the Hough transform has the maximum value. The retrieved skew angle ϕ is used to obtain the projection profile of the region. The profile is calculated by accumulating pixel values in the region along the direction of its skew angle, so that the one-dimensional projection vector v_p is obtained. For a text region, v_p should have regular, high frequency sinusoidal shape with peaks and valleys corresponding to the text lines and the interline spaces, respectively. Conversely, such regularities can not be observed in graphic regions.

As pointed out in [8, 19] text regions visually present well-defined structures of lines with specific orientation and periodicity; on the contrary, graphics regions present uniformity without any specific structure. This means that only text regions present periodicity of the peaks values in the Hough space. Several techniques can be used to measure such a periodicity, as Fourier transform, autocorrelation, and power spectrum density. To measure the regularity of the v_p vector, the Power Spectral Density (PSD) [17] analysis can be performed [2]. In fact, for large text regions, the PSD coefficients show a significant peak as a consequence of the frequency content in the region and for nontext the spectrum is almost flat. Then for each region a small number n of PSD coefficients is selected in the Hough space and used as inputs to the classifier of document image regions.

A vector $\mathbf{x}$ of n coefficients is calculated as follows:

$$\mathbf{x} = |FT(v_p)|^2$$

where $FT(\cdot)$ denotes the Fourier Transform [1] and v_p is the profile vector in the direction of the skew angle ϕ.

By processing the set of available document regions, a number P of input vectors are generated and labeled as belonging to two classes: text and no text. This set of labeled vectors represents the training set used to create the classifier.

Neuro-fuzzy Classifier

To classify each region encoded as a vector X of n coefficients, a neuro-fuzzy network that learns fuzzy classification rules is used. Assume that the classifier is based on K rules of the following type:

$$\text{IF}(x_1 \text{ is } A_{1k})\text{AND} \ldots \text{AND}(x_n \text{ is } A_{nk})\text{THEN} \tag{9.12}$$

$$(\mathbf{x} \in C_1 \text{ with degree } b_{1k}), (\mathbf{x} \in C_2 \text{ with degree } b_{2k})$$

for $k = 1 \ldots K$ where K is the number of rules, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are the input values (i.e., the n PSD coefficients), A_{ik} are fuzzy sets defined on the input variables and b_{jk} are fuzzy singletons representing the degrees to which a region $\mathbf{x}$ belongs to class $C_j, j = 1, 2$. The fuzzy sets A_{ik} are defined by Gaussian membership functions:

$$\mu_{ik}(x_i) = \exp\left[-\frac{(x_i - w_{ik})^2}{2\sigma_{ik}^2} \right]$$

As explained in Sect. 3.4, the neuro-fuzzy network encodes fuzzy rules of the form (9.12) in its topology, and processes information in a way that matches the fuzzy inference scheme. The adjustable parameters (weights) of the network are the premise parameters (w_{ik}, σ_{ik}) and the consequent parameters b_{jk} of fuzzy rules.

The fuzzy rule base is automatically defined by a two-step learning of the neuro-fuzzy network, performed on the training set of these P regions. The first learning phase consists in clustering the input space via the FCM algorithm introduced in Sect. 7.3.2. The clustering process is the basis for arranging an initial fuzzy rule base, referring each rule antecedent to each obtained cluster. For each rule R_k the center w_{ik} of the i-th Gaussian function is defined as the i-th coordinate of the k-th cluster center, and the width σ_{ik} is assigned to the value of the cluster radius. The consequent parameters b_{jk} are initialized by taking into account how much input vectors belonging to class C_j are covered by the k-th cluster, namely:

$$b_{jk} = \frac{\sum_{\mathbf{x} \in C_j} \mu_k(\mathbf{x})}{\sum_{\mathbf{x}} \mu_k(\mathbf{x})}$$

The second learning phase, based on a gradient descent technique, adjusts the free parameters of the neuro-fuzzy network, thus improving the accuracy of the information embedded in the fuzzy rule base. Details of the learning algorithm can be found in [4] where the neuro-fuzzy methodology has been applied in different contexts.

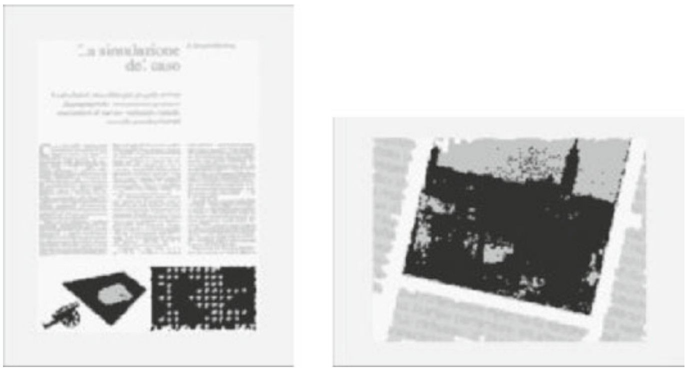


Fig. 9.4 Result of the neuro-fuzzy classification on the document images of Figs. 9.2a and 9.3a. *White* regions corresponding to background, *gray* regions are classified as text, *black* regions are classified as no-text (graphics)

After the learning phase, the neuro-fuzzy network is applied to classify regions of testing documents through the inference of the learned fuzzy rules. To obtain a “hard” (crisp) classification from the output of the neuro-fuzzy classifier, the class with the highest membership value is selected and associated to the input region. Figure 9.4 shows the classified regions in the two sample documents depicted in Figs. 9.2a and 9.3a. It can be seen that in both documents all the identified components are correctly classified into text regions and graphics regions.

References

1. Bracewell, R.: The Fourier Transform and its Applications. New York (1965)
2. Caponetti, L., Castellano, G., Fanelli, A.M.: A neuro-fuzzy system for document image segmentation and region classification. In: Proc. of the 2nd IEEE International Workshop on Intelligent Signal Processing (WISP2001), pp. 27-32 (2001)
3. Caponetti, L., Castiello, C., Goreki, P.: Document pagesegmentation using neuro-fuzzy approach. Appl. Soft Comput. **8**, 118–126 (2008)
4. Castellano, G., Castiello, C., Fanelli, A.M., Mencar, C.: Knowledge discovering by a neuro-fuzzy modeling framework. Fuzzy Sets Syst. **149**(1), 187–207 (2005)
5. De Luca, A., Termini, S.: A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory. Inf. Control **20**(4), 301–312 (1972)
6. Di Zenzo, S., Cinque, L., Levialdi, S.: Image thresholding using fuzzy entropies. IEEE Trans. Syst. Man Cybern. **28**(1), 15–23 (1998)
7. Górecki, P., Caponetti, L., Castiello, C.: Fuzzy techniques for text localisation in images. In: Computational Intelligence in Multimedia Processing: Recent Advances, pp. 233–270. Springer, Heidelberg (2008)
8. Hinds, S., Fisher, J., D’Amato, D.: A document skew detection method using run-length encoding and Hough transform. In: Proceedings of the 10th International Conference on Pattern Recognition (ICPR), pp. 464–468 (1990)
9. Hough, P.: Machine analysis of bubble chamber pictures. In: International Conference on High Energy Accelerators and Instrumentation, CERN (1959)
10. Huang, L.K., Wang, M.J.J.: Image thresholding by minimizing the measures of fuzziness. Pattern Recognit. **28**(1), 41–51 (1995)
11. Landini, G.: ImageJ Plugin Threholding. Available at: http://imagej.net/User:Landini
12. Lee, S.U., Chung, S.Y.: A comparative performance study of several global thresholding techniques for segmentation. Comput. Vis. Graph. Image Process. **52**, 171–190 (1990)
13. Mei, C.: ImageJ Plugin Otzu Thresholding. Available at: http://rsb.info.nih.gov/ij/plugins/otsu-thresholding.html
14. Nagy, G.: Twenty years of document image analysis in PAMI. IEEE Trans. Pattern Anal. Mach. Intell. **22**(1), 38–62 (2000)
15. O’Gorman, L., Kasturi, R.: Document Image Analysis. IEEE Computer Society Press, Washington (1995)
16. Pal, S.K., Rosenfeld, A.: Image enhancement and thresholding by optimization of fuzzy compactness. Pattern Recognit. Lett. **7**, 77–86 (1988)
17. Pratt, W.: Digital Image Processing, 3rd edn. Wiley, New York, NY (2001)
18. Sahoo, P.K., Soltani, S., Wong, A.K.C.: A survey of threshold techniques. Comput. Vis. Graph. Image Process. **41**, 233–260 (1988)
19. Srihari, S.N., Govindaraju, V.: Analysis of textual images using the Hough transform. Mach. Vis. Appl. **2**, 141–153 (1989)
20. University of Oulu, Finland, Document Image Database. http://www.ee.oulu.fi/research/imag/document/

Appendix A
Java Code References

All's well that ends

Arthur Bloch–Murphy' Law

In this Appendix we report some plugins that are available online for image processing tasks. Reference URLs are listed below.

Chapter 2:

- The Imagej plugin implementing the histogram of each component RGB of a color image is available at: http://rsb.info.nih.gov/ij/plugins/color-histogram.html

- The Imagej plugin implementing the Canny algorithm is available at: http://rsbweb.nih.gov/ij/plugins/canny/index.html

Chapter 7:

- The Imagej plugin implementing the *K-means* algorithm for image segmentation is available at: http://ij-plugins.sourceforge.net/plugins/segmentation/k-means.html
- The Imagej plugin implementing the Hough transform for detecting circles of various radius is available at: http://rsb.info.nih.gov/ij/plugins/hough-circles.html
- The SFCM ImageJ plugin implementing K-means, FCM and SFCM clustering is available at: https://github.com/arranger1044/SFCM. More information can be found at: https://sites.google.com/site/cilabuniba/research/sfcm. The web page is shown in Figs. A.1 and A.2.

Chapter 8:

- The ImageJ plugin implementing the Otzu thresholding method is available at: http://rsb.info.nih.gov/ij/plugins/otsu-thresholding.html/
- The ImageJ plugin implementing some thresholding methods is available at: http://imagej.net/Auto_Threshold/

L. Caponetti and G. Castellano, *Fuzzy Logic for Image Processing*,
SpringerBriefs in Electrical and Computer Engineering,
DOI 10.1007/978-3-319-44130-6

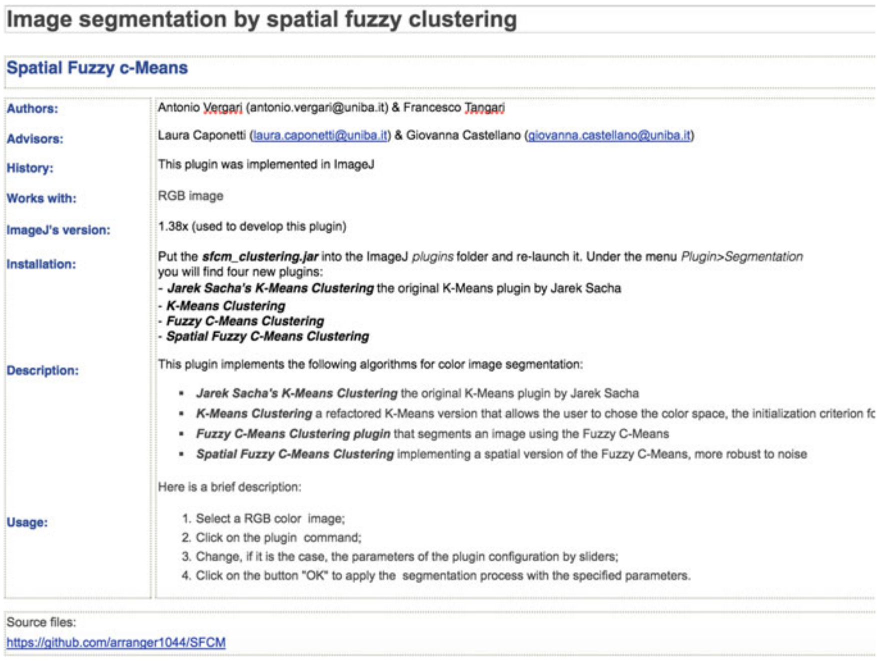


Image segmentation by spatial fuzzy clustering

Spatial Fuzzy c-Means

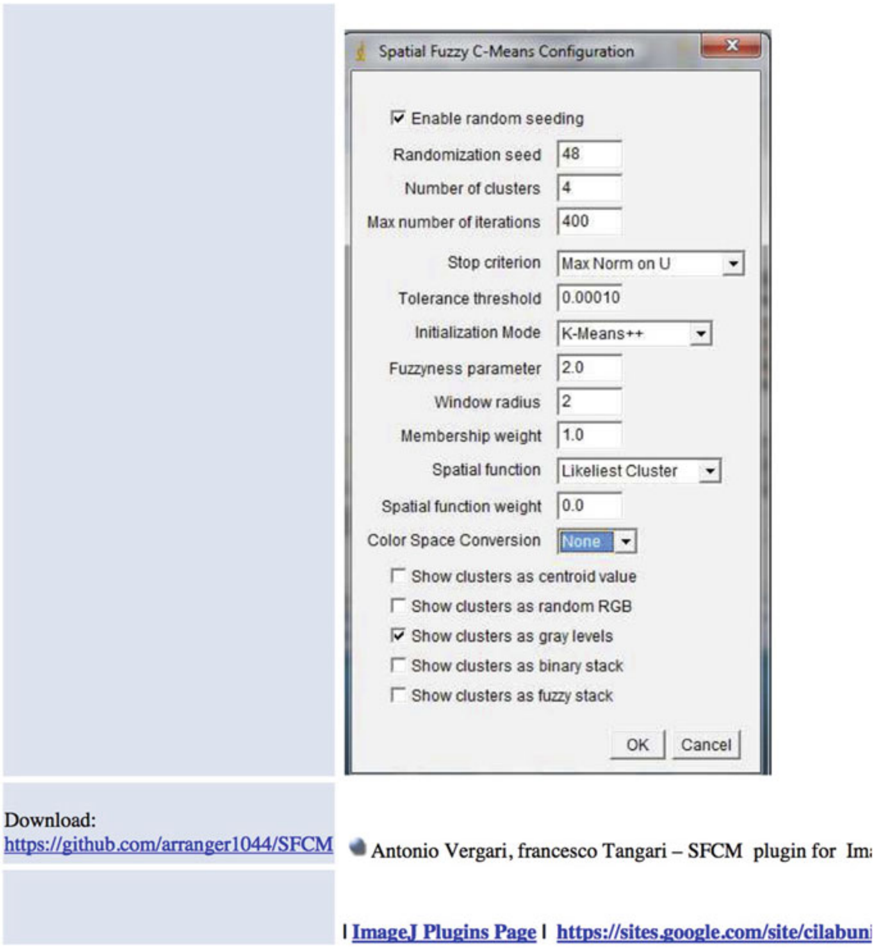
Authors:	Antonio Vergari (antonio.vergari@uniba.it) & Francesco Tangari
Advisors:	Laura Caponetti (laura.caponetti@uniba.it) & Giovanna Castellano (giovanna.castellano@uniba.it)
History:	This plugin was implemented in ImageJ
Works with:	RGB image
ImageJ's version:	1.38x (used to develop this plugin)
Installation:	Put the ***sfcm_clustering.jar*** into the ImageJ *plugins* folder and re-launch it. Under the menu *Plugin>Segmentation* you will find four new plugins: - ***Jarek Sacha's K-Means Clustering*** the original K-Means plugin by Jarek Sacha - ***K-Means Clustering*** - ***Fuzzy C-Means Clustering*** - ***Spatial Fuzzy C-Means Clustering***
Description:	This plugin implements the following algorithms for color image segmentation: • ***Jarek Sacha's K-Means Clustering*** the original K-Means plugin by Jarek Sacha • ***K-Means Clustering*** a refactored K-Means version that allows the user to chose the color space, the initialization criterion fc • ***Fuzzy C-Means Clustering plugin*** that segments an image using the Fuzzy C-Means • ***Spatial Fuzzy C-Means Clustering*** implementing a spatial version of the Fuzzy C-Means, more robust to noise. Here is a brief description:
Usage:	1. Select a RGB color image; 2. Click on the plugin command; 3. Change, if it is the case, the parameters of the plugin configuration by sliders; 4. Click on the button "OK" to apply the segmentation process with the specified parameters.

Source files:
https://github.com/arranger1044/SFCM

Fig. A.1 The web page describing the SFCM plugin

Chapter 9:

- The ImageJ plugin implementing the morphological operators is available at: http://sites.imagej.net/Landini/



Spatial Fuzzy C-Means Configuration

[✓] Enable random seeding

Randomization seed	48
Number of clusters	4
Max number of iterations	400
Stop criterion	Max Norm on U
Tolerance threshold	0.00010
Initialization Mode	K-Means++
Fuzzyness parameter	2.0
Window radius	2
Membership weight	1.0
Spatial function	Likeliest Cluster
Spatial function weight	0.0
Color Space Conversion	None

[ ] Show clusters as centroid value
[ ] Show clusters as random RGB
[✓] Show clusters as gray levels
[ ] Show clusters as binary stack
[ ] Show clusters as fuzzy stack

Download:
https://github.com/arranger1044/SFCM

Antonio Vergari, francesco Tangari – SFCM plugin for Im

| ImageJ Plugins Page | https://sites.google.com/site/cilabuni

Fig. A.2 SFCM plugin configuration

Index