Chapter 1

Digital Image Processing with MATLAB

Mahmut Sinecen

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/63028

Abstract

The chapter relates to the Image Processing Toolbox in MATLAB. We learn about its general information and some examples will be solved using it. After finishing this chapter, you can use MATLAB Image Processing Toolbox and write script for processing of images.

Keywords: MATLAB, digital, image, processing, Fundamental

1. Digital image processing

The image may be defined as a two-dimensional visual information that are stored and displayed. An image is created by photosensitive devices which capture the reflection light from two-dimensional surface of object in the three-dimensional real world (**Figure 1**). Each image has intensity or gray value in x – y coordinate plane. If it is finite and discrete quantities, image is called digital image. In **Figure 2**, some digital images are shown.

Digital image processing (DIP) has the different techniques for processing of digital images. DIP has been applying many fields with technological advances, such as Medicine, Geographical Information Technologies, Space Sciences, Military Applications, Security, Industrial Applications.

1.1. Pixel

Pixels, which are called pel or picture elements, may be defined as the smallest addressable element in the digital image. Pixels of a color image have Red, Green, and Blue gray values (**Figure 3**).

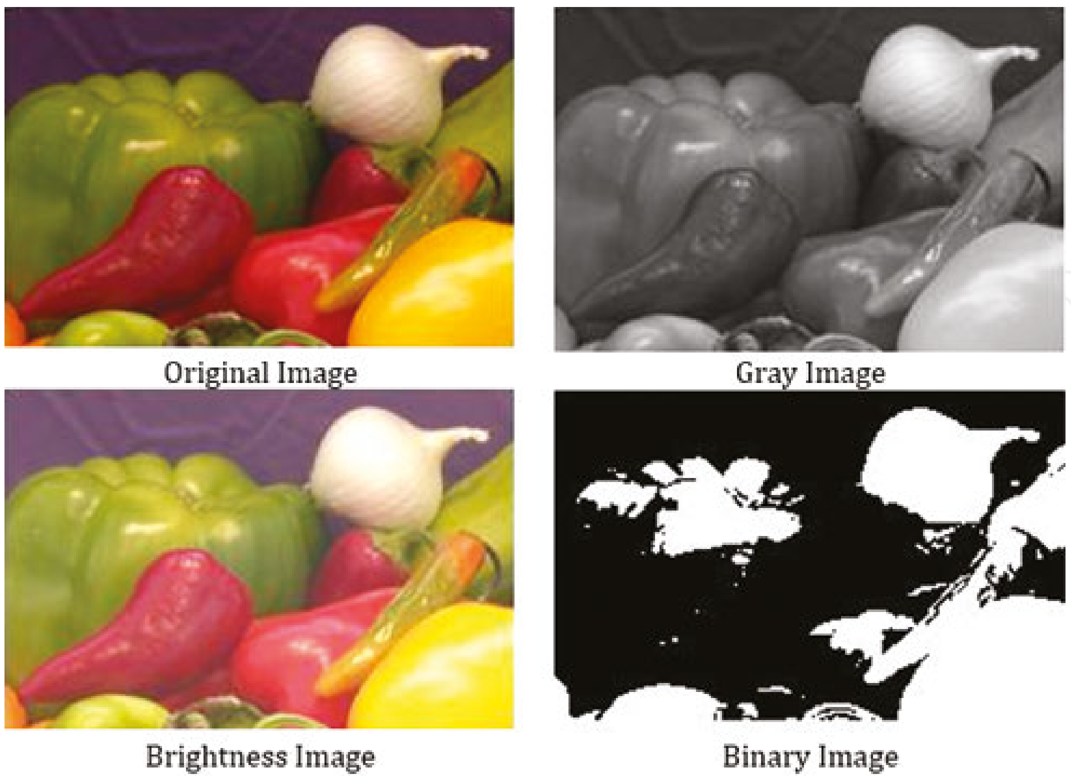


Figure 1. Image.

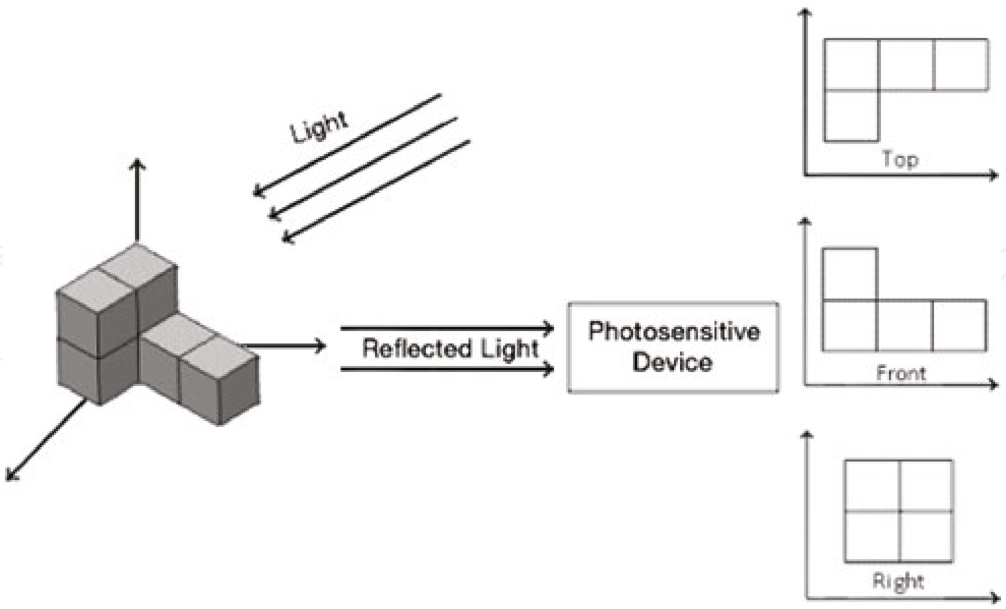


Figure 2. Digital images.

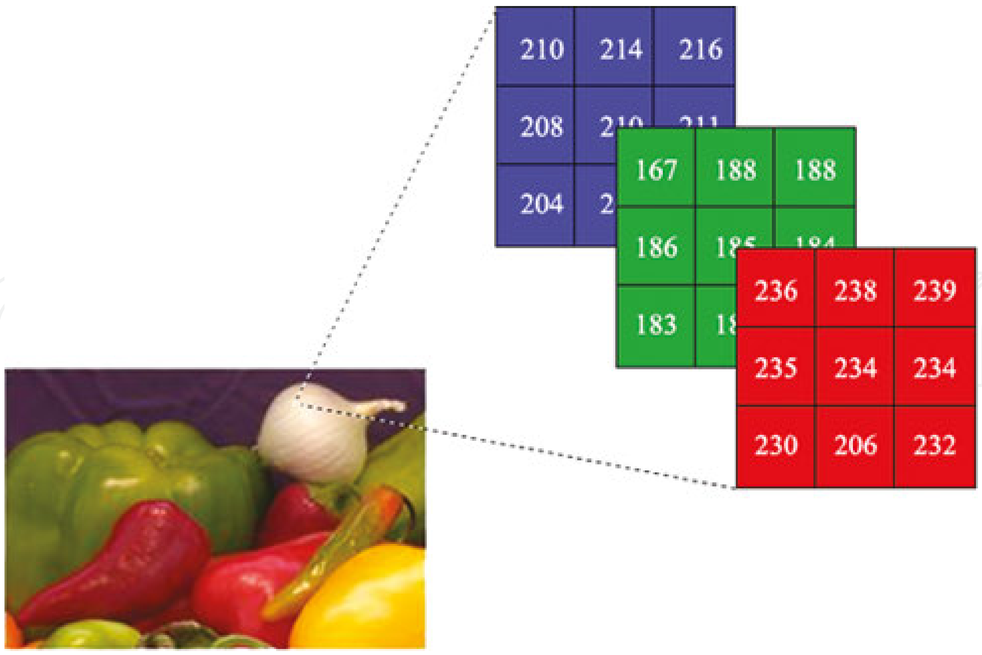


Figure 3. Pixels of a color image.

1.1.1. Pixels relationships

1.1.1.1. Neighbors of a pixel

A pixel has three different neighbor types that are 4, 8, and diagonal. As shown in **Table 1**, neighbor of a pixel (p) in the x, y point of image (f) is defined in that 4-neighbors;

f(x - 1, y - 1)	f(x - 1, y)	f(x - 1, y + 1)
f(x, y - 1)	p	f(x, y + 1)
f(x + 1, y - 1)	f(x + 1, y)	f(x + 1, y + 1)

Table 1. Neighbor of a pixel.

N_4(p) is shown as 4-neighbor of p pixel. Any pixel p in the image has two vertical and horizontal neighbors, and each of them is a unit distance of p, given by

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

Diagonal neighbors;

Although diagonal neighbors are the same of 4-neighbor, neighbor pixels are the corner of pixels (p) and each of them is at Euclidean distance of p, given by

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

8-neighbors;

8-neighbors is a combination of $N_4(p)$ and $N_D(p)$ and shown as $N_8(p)$.

$$N_8(p) = \begin{Bmatrix} f(x-1, y-1), f(x-1, y+1), f(x+1, y+1), f(x+1, y-1), \\ f(x, y-1), f(x-1, y), f(x, y+1), f(x+1, y) \end{Bmatrix}$$

1.1.1.2. Adjacency

If two pixels are neighbors and their gray level values satisfy some specified criterion, then they are connected. A set of intensity values (V) is used to define adjacency and connectivity. There are three types of adjacency (**Figure 4**).

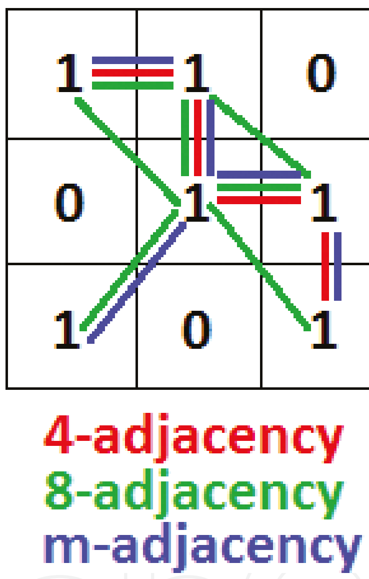


Figure 4. Pixel adjacency.

4-adjacency

p and q pixels are 4-adjacency if they are $N_4(p)$ with values from V.

8-adjacency

p and q pixels are 8-adjacency if they are $N_8(p)$ with values from V.

m-adjacency (mixed)

p and q pixels are m-adjacency if;

- q is in $N_4(p)$ or,

- q is in $N_D(p)$ and,
- $N_4(p) \cap N_4(q) = \circledcirc$ with values from V.

1.1.1.3. Path

A path from pixel p with coordinate (x, y) to pixel q with coordinate (s, t) with values from V is defined as 4- ,8- , or m-paths depending on the type of adjacency specified.

According to V = {2,3,5}, If we want to find p and q pixels 4-, 8- and m-path, (**Figure 5**)

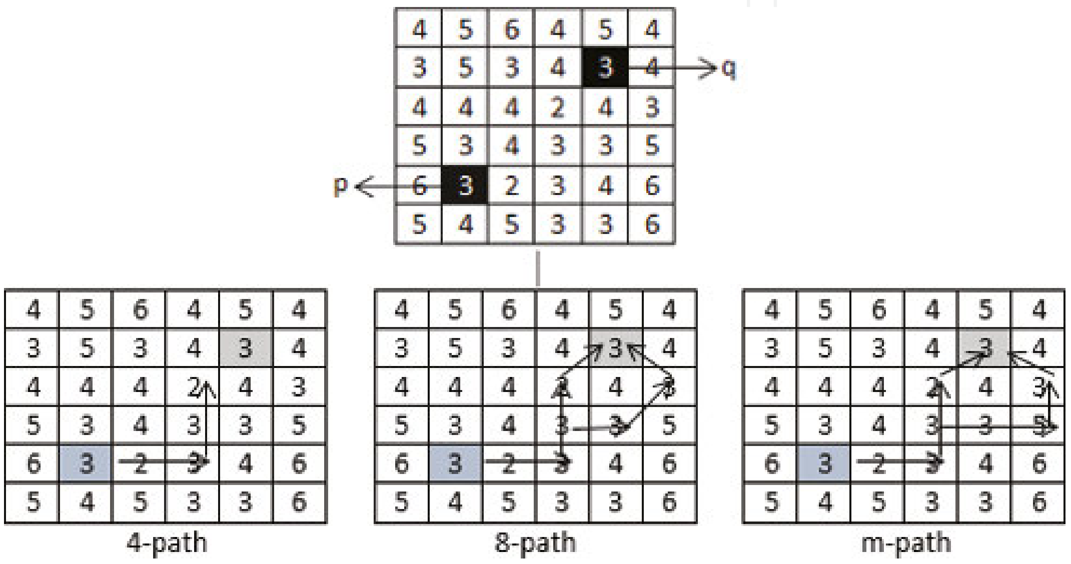


Figure 5. Finding 4-, 8-, m-path between p and q pixels.

1.1.1.4. Distance measures of pixels

- Euclidean Distance (D_e)

$$D_e(p,q) = \sqrt{(x-s)^2 + (y-t)^2}$$

- City-block Distance (D_4)

$$D_4(p,q) = |x-s| + |y-t|$$

- Chessboard Distance (D_8)

$$D_8(p,q) = \max(|x-s|,|y-t|)$$

- D_m Distance; it is defined as the shortest m-path.

According to V = {2,3,5}, if we want to find D_m distance from p pixel to q pixel (**Figures 6**, **7**, **8**);

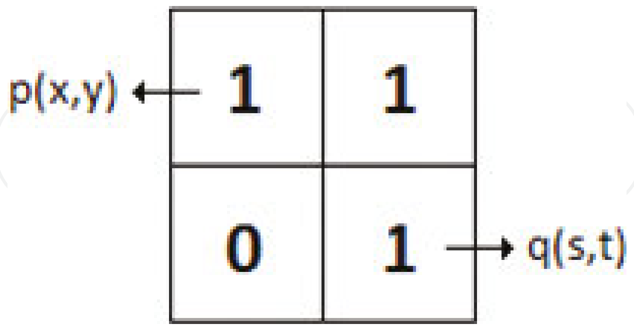


Figure 6. Distance between p and q pixels.

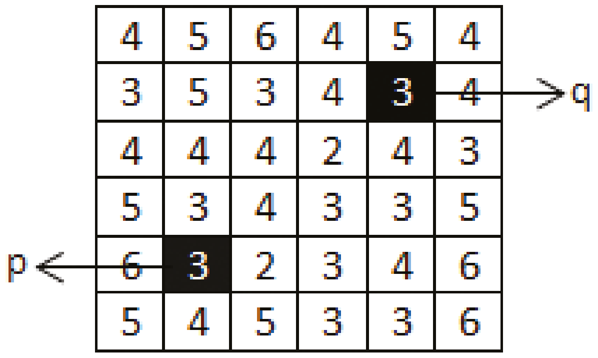


Figure 7. Example about the shortest m-path.

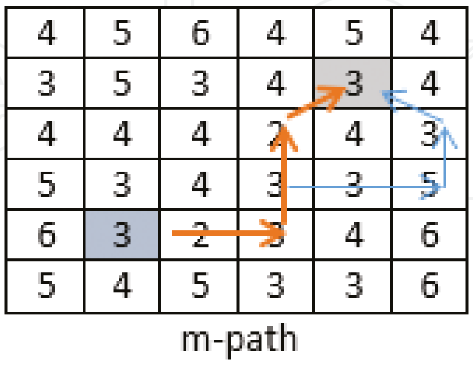


Figure 8. Solving example in the Figure 7.

D_m is 5 because orange path is shorter than blue path.

```
bw = zeros(200,200);
bw(50,50) = 1;
bw(50,150) = 1;
bw(150,100) = 1;

D1 = bwdist(bw,'euclidean');
D2 = bwdist(bw,'cityblock');
D3 = bwdist(bw,'chessboard');
D4 = bwdist(bw,'quasi-euclidean');

figure
subplot(2,2,1);
subimage(mat2gray(D1));hold on, imcontour(D1);title('Euclidean');
subplot(2,2,2);
subimage(mat2gray(D2));hold on, imcontour(D2);title('City Block');
subplot(2,2,3);
subimage(mat2gray(D3));hold on, imcontour(D3);title('Chessboard');
subplot(2,2,4);
subimage(mat2gray(D4));hold on, imcontour(D4);title('Quasi-Euclidean');
```

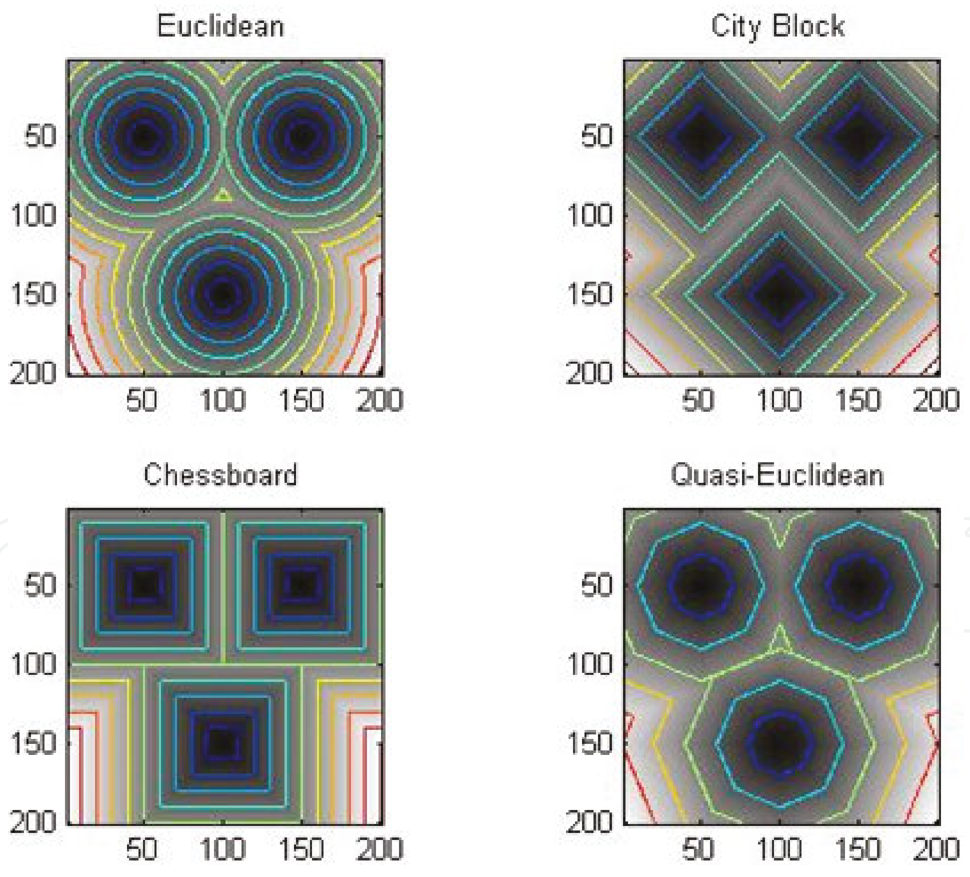


Figure 9. Distance measuring types.

1.2. Spatial resolution

Spatial resolution can be defined as the number of pixels per inch. Different spatial resolutions of same image are shown in the **Figure 10**. Spatial resolution has different measuring methods for different devices.

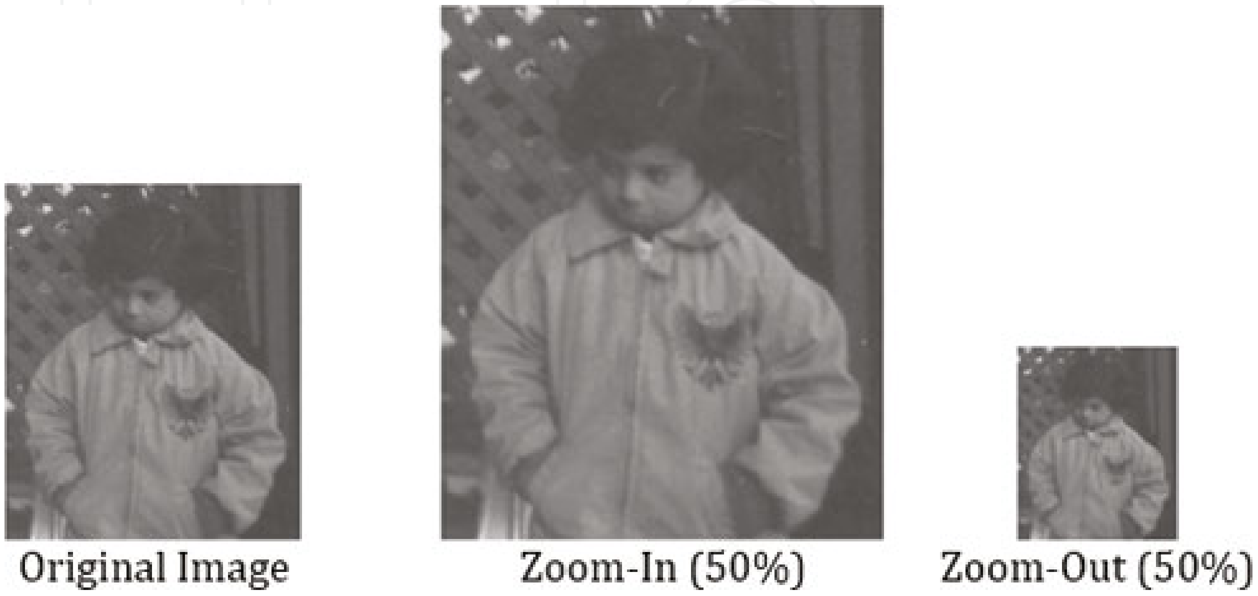


Figure 10 .Different spatial resolutions of same image.

1.2.1. Dots per inch (DPI)

DPI is generally used in monitors. Sometimes it is called PPI (Pixels Per Inch). But the two expressions have a difference. DPI is also used for measuring spatial resolution of printers. It means DPI defines how many dots of ink on printed image per inch.

1.2.2. Pixels per inch (PPI)

PPI is generally used in tablets, mobile phones, etc. If a and b are height and width resolutions of image, we can calculate ppi value of any device using Equation 1.

$$PPI = \frac{\sqrt{a^2 + b^2}}{Diagonal\ Size\ of\ Devices} \tag{1}$$

For example; 1080 × 1920 pixels, 5.5 inch Iphone 6s Plus PPI value;

$$PPI = \frac{\sqrt{1080^2 + 1920^2}}{5.5} \cong 401\ \ (it\ \ is\ \ shown\ \ in\ \ apple\ \ web\ \ site)$$

$$PPI = \frac{\sqrt{1080^2 + 1920^2}}{5.5} \cong 401 \; (\textit{it is shown in apple web site})$$

1.2.3. Lines per inch (LPI)

LPI is referred lines of dots per inch of printers. Printer has different LPI values as shown in **Tables 2**.

Printer	LPI value
Screen printing	45–65 LPI
Laser printing (300 dpi)	65 LPI
Laser printing (600 dpi)	85–105 LPI

Table 2. LPI Value of Printer.

1.3. Image file formats

Image file formats are important for printing, scanning, using on the Internet, etc. The different formats are used in the world. The most common formats are jpg, tif, png, and gif (**Figures 11**). In this section, the most common file formats (JPG, TIF, PNG, and GIF) are explained.

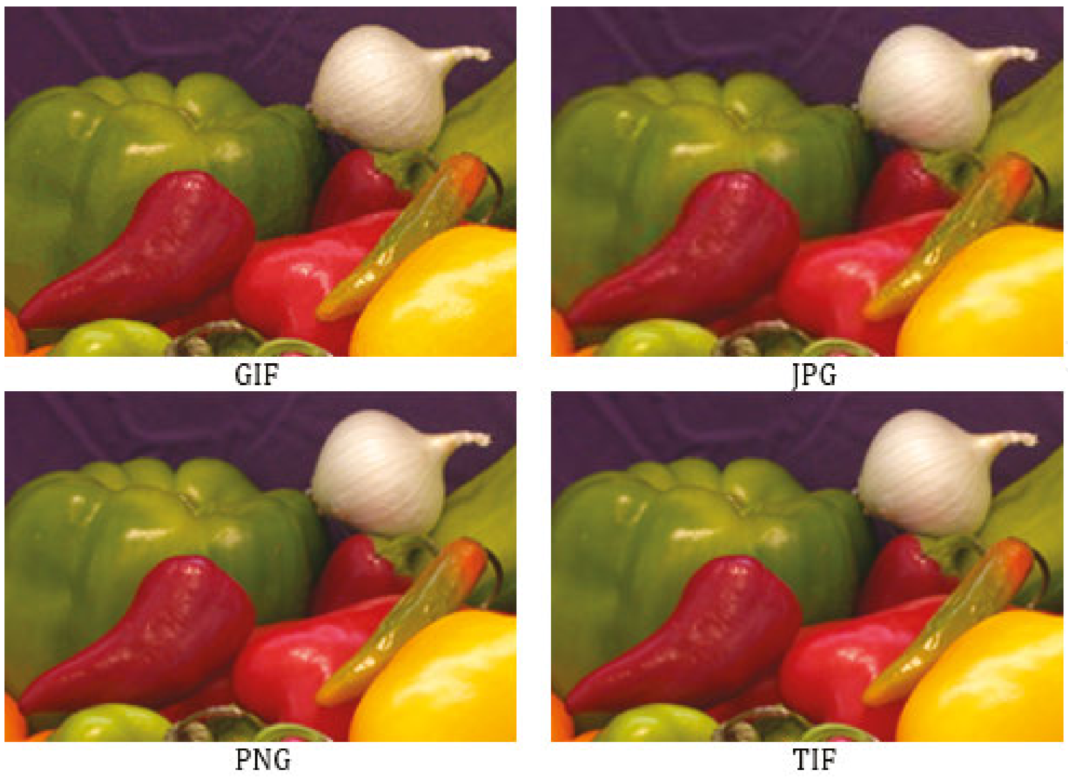


Figure 11. Image formats.

1.3.1. JPG (Joint Photographic Expert Group)

JPEG or JPG is the most common standard for compressing digital image. It is used in web pages, document, email, etc. Because digital images have smaller size than other file formats. However, JPEG images have very low resolution.

1.3.2. TIF (Tagged Image File Format)

TIFF or TIF has the best resolution for using commercial works. Although it is very high quality, the files have very big size.

1.3.3. GIF (Graphics Interchange Format)

GIF, which was used 8-bit video for the people connecting to internet using dial-up modem, was designed by CompuServe.

1.3.4. PNG (Portable Network Graphics)

PNG file format has smaller size than TIF and more resolution than GIF and JPG. Nowadays, it is used in the web pages because of having transparency property.

2. Basic image processing with MATLAB

MATLAB is a very simple software for coding. All data variable in MATLAB are thought a matrix and matrix operations are used for analyzing them. MATLAB has the different toolboxes according to application areas. In this section, MATLAB Image Processing Toolbox is presented and the use of its basic functions for digital image is explained.

2.1. Read, write, and show image

imread() function is used for reading image. If we run this function with requiring data, image is converted to a two-dimensional matrix (gray image is two-dimensional, but, color image is three-dimensional) with rows and columns including gray value in the each cell.

```
I = imread('path/filename.fileextension');
```

imread() function only needs an image file. If the result of imread() function is equal to a variable, a matrix variable (I) is created. File name, extension, and directory path that contains image must be written between two single quotes. If script and image file are in the same folder, path is not necessary.

The matrix variable of image is showed using imshow() function. If many images show with sequence on the different figure windows, we use “figure” function for opening new window.

imwrite() function is used to create an image. This function only requires a new image file name with extension. If the new image is saved to a specific directory, the path of directory is necessary.

```
% Clear all workspace
clear
% Close All Open Images
close all
% Clear Command Windows
clc
% Read Image
I = imread('onion.png');
% Show Image
imshow(I);
% Create New Image that has different format
imwrite(I,'onion.jpg');
% Create New Image that has different format and save specific directory
imwrite(I,'C:\NewImageDirectory\onion.jpg');
```

2.2. Image reverse

Image reserve technique, each all elements of the matrix is replaced to be the top row elements to bottom row and the bottom row elements to top row. In the other words, the image rotates on the vertical axis.

MATLAB Image Processing Toolbox does not have function for it. Either the script is written or flipdim function can be used (**Figure 12**).

```
I = imread('onion.png');            % Original Image
I_mirror  = flipdim(I,2);           % Mirror Image
I_reverse = flipdim(I,1);           % Reverse Image
I_mirrev  = flipdim(I_reverse,2);   % Reverse + Mirror Image

figure,
subplot(2,2,1), imshow(I);title('Original Image');
subplot(2,2,2), imshow(I_mirror);title('Mirror Image');
subplot(2,2,3), imshow(I_reverse);title('Reverse Image');
subplot(2,2,4), imshow(I_mirrev);title('Reverse + Mirror Image');
```

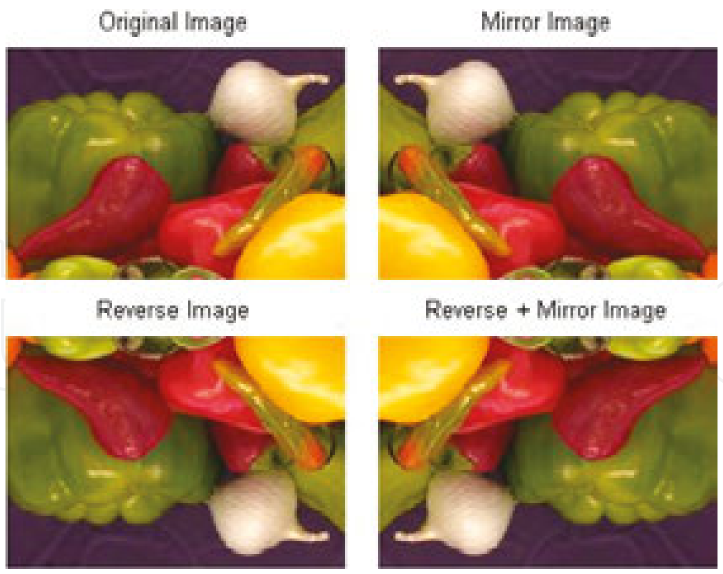


Figure 12. Vertical and horizontal reverse.

2.3. Image mirroring

Mirroring technique is the rotating of reversed image on the horizontal axis. In MATLAB Image Processing Toolbox has imrotate() function for rotating image. This function needs three properties which are image matrix variable, rotating angle, and interpolation method (**Figure 13**).

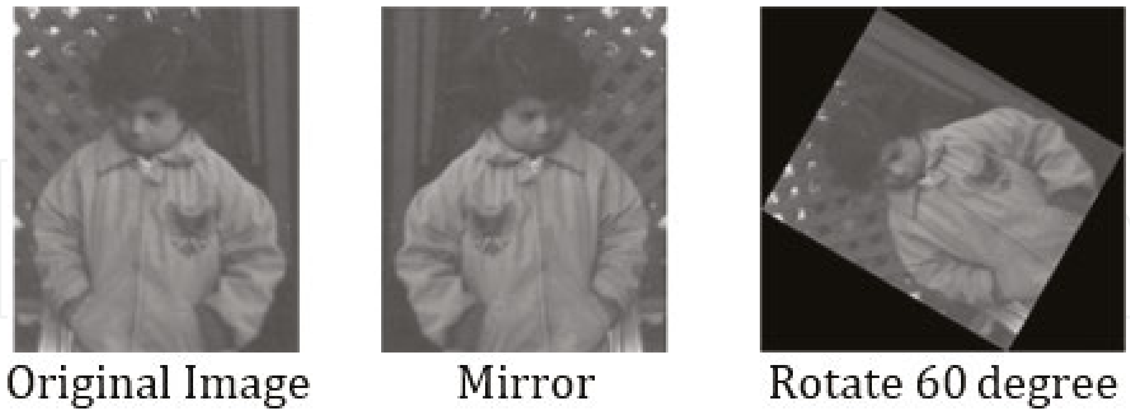


Figure 13. Image rotate.

I_rotate = imrotate(Image Matrix Variable, Angle, Interpolation Method)

Interpolation method

- 'nearest': Nearest-Neighbor Interpolation

- 'bilinear': Bilinear Interpolation
- 'bicubic': Bicubic Interpolation

Example

```
I = imread('pout.tif');
imshow(I);
I_mirror = flipdim(I,2);
figure, imshow(I_mirror);
I_rotate = imrotate(I,60,'bilinear');
figure, imshow(I_rotate);
```

2.4. Image shift

Sometimes, an image can be wanted to shift up to certain pixel value on the horizontal and vertical axis. imtranslate() function is used to shift of an image. In the **Figure 14** the image shifts 15 px right and 25 px bottom.

```
I = imread('pout.tif');
imshow(I);
I_shift = imtranslate(I,[15, 25]);
figure, imshow(I_shift);
```

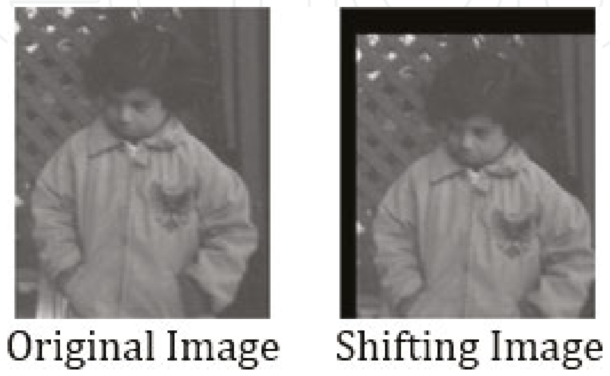


Figure 14. Image shift.

2.5. Image resize

If an image is displayed big or small size for showing details or general view, its resolution must be changed. These situations are called zoom-in and zoom-out. Digital cameras or photosensitive devices use optic lenses for zoom-in and zoom-out. But, interpolation methods are only used for digital images. Most common problem of interpolation methods is the changing quality of image (**Figures 15, 16**).

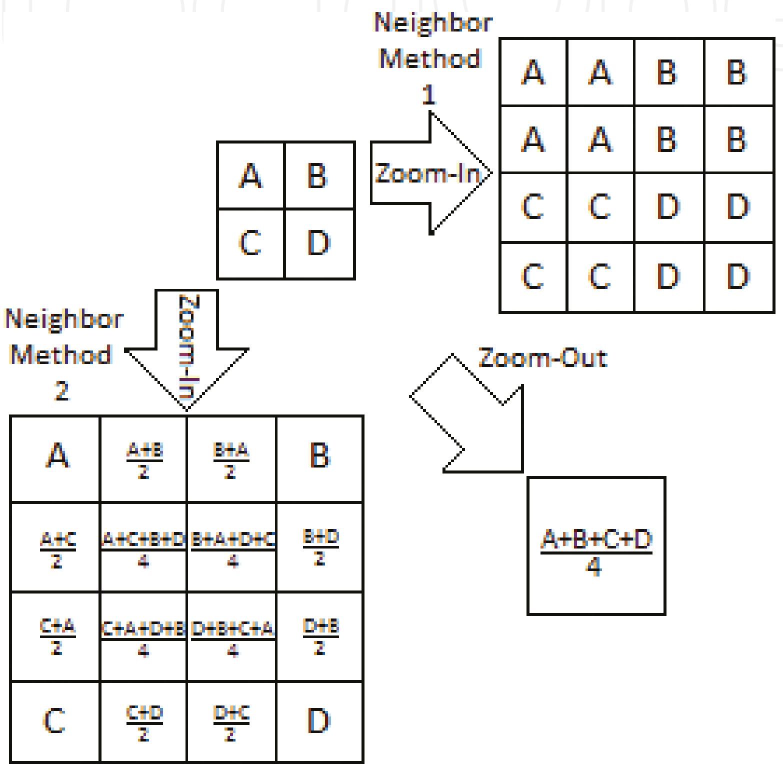


Figure 15. Zoom-in and zoom-out.

I_resize = imresize(I, Resize Rate, Interpolation Method)

I is image variable, if Resize Rate is bigger than 1, it means zoom-in, otherwise zoom-out.

```
I = imread('pout.tif');
imshow(I);
% 50% Increase Image Size (Zoom-In)
I_resize_big = imresize(I, 1.5, 'nearest');
figure, imshow(I_resize_big);
% 50% Decrease Image Size (Zoom-Out)
I_resize_small = imresize(I, 0.5, 'nearest');
figure, imshow(I_resize_small);
```

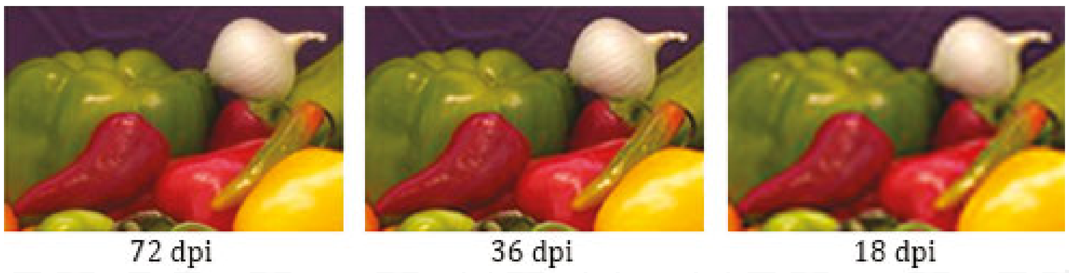


Figure 16. Image resize.

3. Image enhancement

In some cases, an image has useless or insufficient information for extracting objects because of different defects. So that, the image must be processed using different digital image processing techniques for removing the defects or artifacts. In this section, some principal methods are explained for increasing the visibility and decreasing defects.

3.1. Brightness

Brightness of an image is adjusted with adding or subtracting a certain value to gray level of each pixel.

$$G(i,j) = F(i,j) + b \quad \begin{array}{l} b > 0\,\textit{Brightness incerease} \\ b < 0\,\textit{Brightness decrease} \end{array}$$

I_adjust = imadjust(I, [low_in; hig_in], [low_out;high_out])

New image (I_adjust) intensity values are between low_out and high_out gray values (**Figure 17**).

```
I = imread('pout.tif');
figure, imshow(I);
I_adjust = imadjust(I);
figure, imshow(I_adjust);
I_adjust = imadjust(I, [0.2; 0.3], [0.5; 0.7]);
figure, imshow(I_adjust);
```

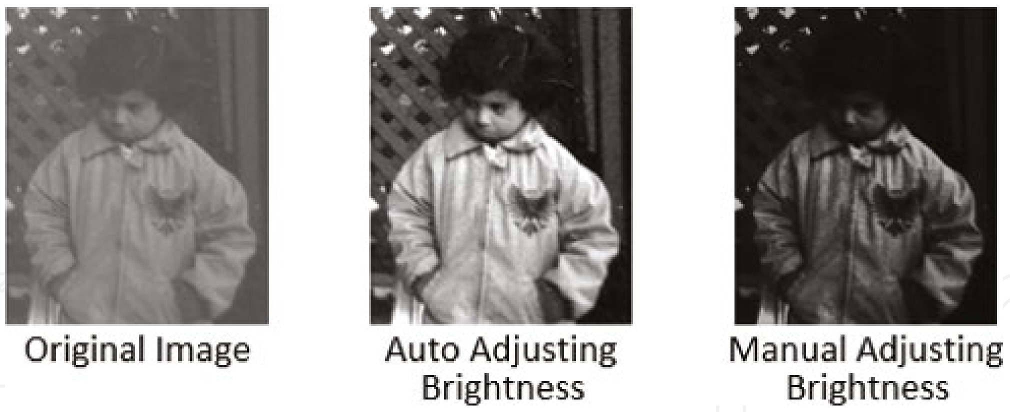


Figure 17. Changing brightness of the image.

```
I = imread('onion.png');
imshow(I);
% [low_in(R) low_in(G) low_in(B); high_in(R) high_in(G) high_in(B)] map to
% [low_out(R) low_out(G) low_out(B); high_out (R) high_out (G) high_out(B)]
I_adjust = imadjust(I, [0.2 0.3 0.4; 0.3 0.5 0.5], [0.3 0.5 0.6; 0.7 0.6
0.8]);
figure, imshow(I_adjust);
```

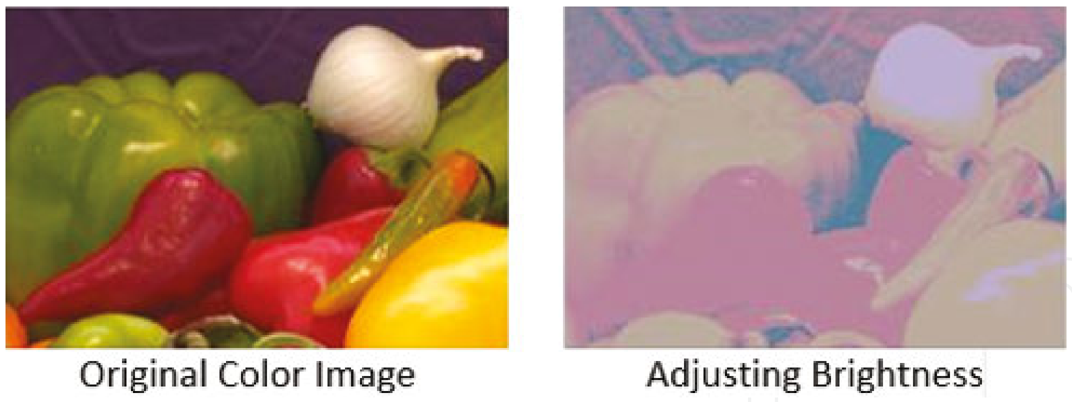


Figure 18. Adjusting Brightness of Color Image.

The MATLAB script above is used for gray image, but we want to change brightness of color image, so, we must change all intensity values of R (red), G (green) and B (blue) channel of the image (**Figure 18**).

3.2. Contrast

Contrast of an image can be changed by multiplying all pixel gray value by a certain value.

$$G(i,j) = c * F(i,j) \begin{matrix} c > 0\, contrast\, increase \\ c < 0\, contrast\, decrease \end{matrix}$$

MATLAB Image Processing Toolbox has the Contrast Adjust tool to change contrast of an image. As shown in **Figure 19**, GUI allows the user for changing contrast using handling.

```
I = imread('pout.tif');
imshow(I);
% Open Contrast Tool
imcontrast();
```

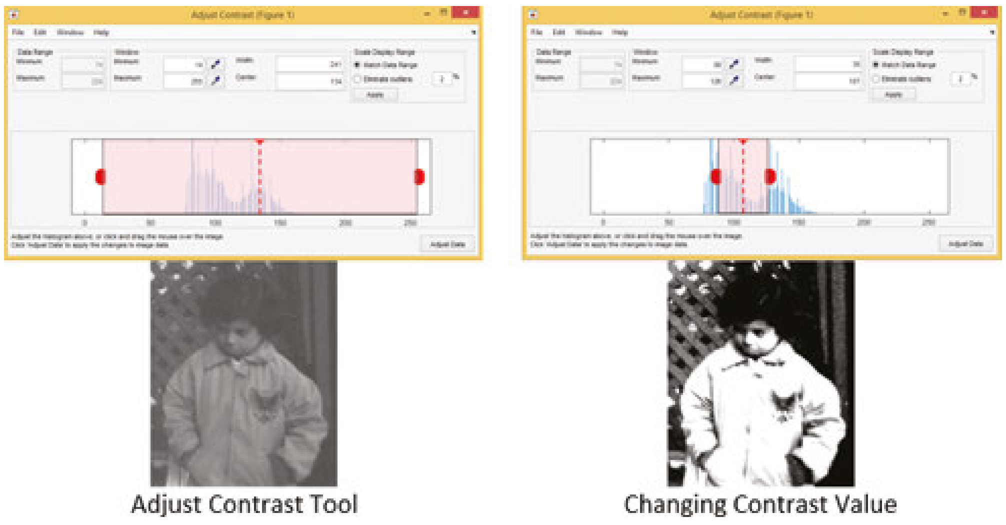


Figure 19. Adjust contrast tool.

3.3. Negative

Intensity values of image are reversed as linear for negative image (**Figure 20**).

```
I = imread('pout.tif');
% Negative Image
I_neg = imcomplement(I);
subplot(1,2,1);imshow(I);title('Original Image');
subplot(1,2,2);imshow(I_neg);title('Negative Image');
```

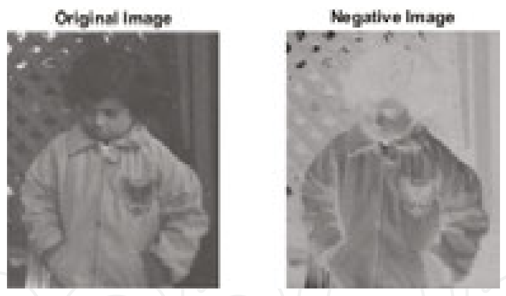
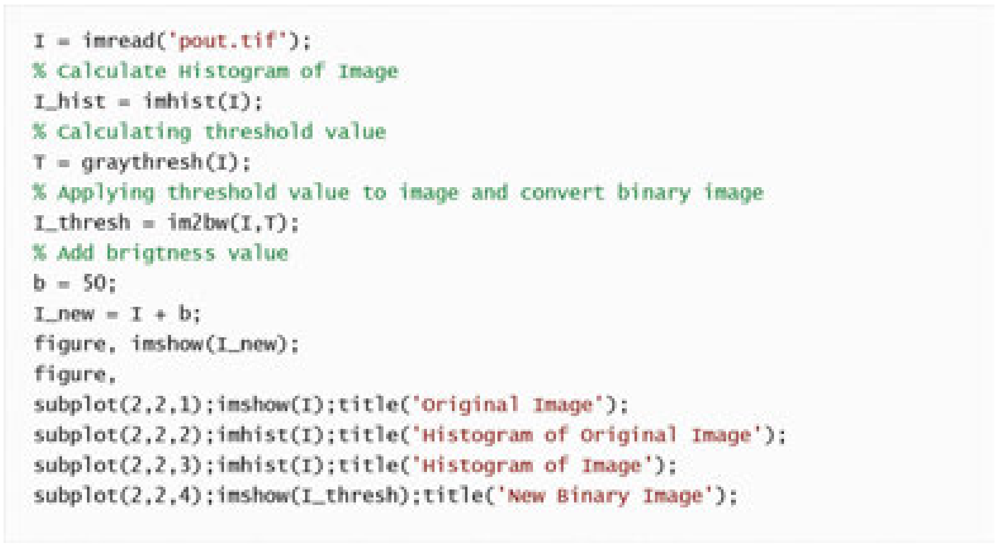


Figure 20. Negative Image.



```
I = imread('pout.tif');
% Calculate Histogram of Image
I_hist = imhist(I);
% Calculating threshold value
T = graythresh(I);
% Applying threshold value to image and convert binary image
I_thresh = im2bw(I,T);
% Add brigtness value
b = 50;
I_new = I + b;
figure, imshow(I_new);
figure,
subplot(2,2,1);imshow(I);title('Original Image');
subplot(2,2,2);imhist(I);title('Histogram of Original Image');
subplot(2,2,3);imhist(I);title('Histogram of Image');
subplot(2,2,4);imshow(I_thresh);title('New Binary Image');
```

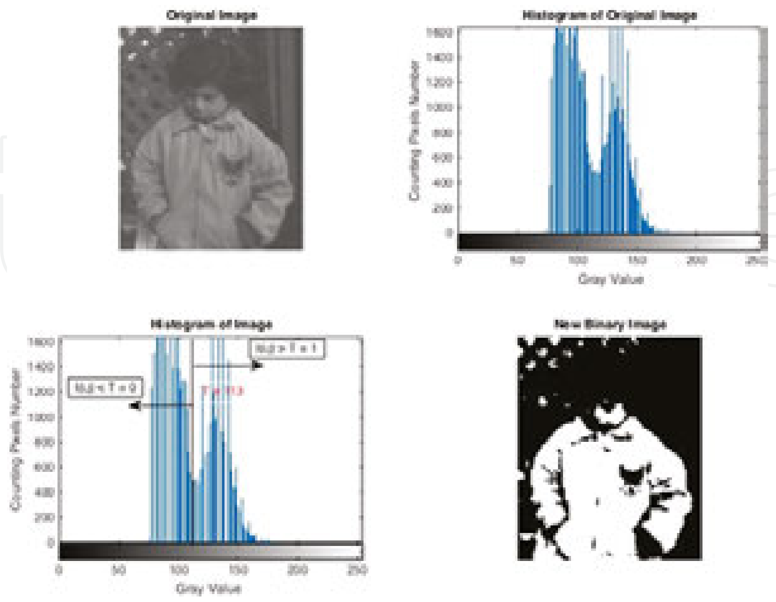


Figure 21. Thresholding.

3.4. Thresholding

Thresholding is generally used for converting image to binary image (0 and 1 gray value). The algorithm of thresholding is defined in Equation 2.

$$G(x,y)=\begin{cases} I(x,y)\geq T & 1 \\ I(x,y)<T & 0 \end{cases} \tag{2}$$

I is original image, x and y are row and column number, T is threshold value, G is new image fter applying threshold (**Figure 21**).

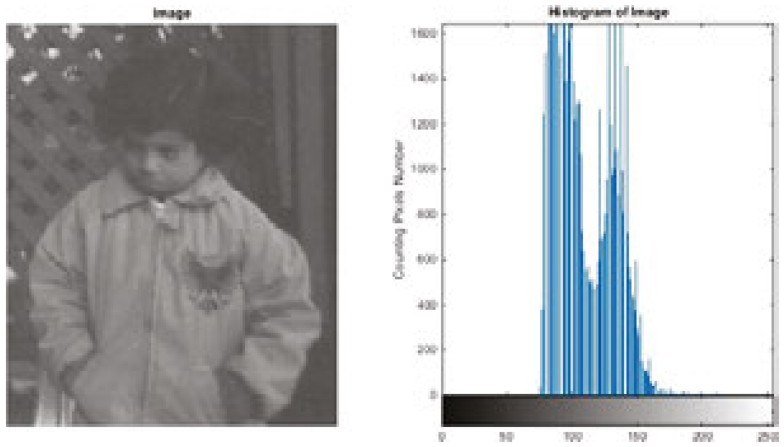


Figure 22. Histogram of the image.

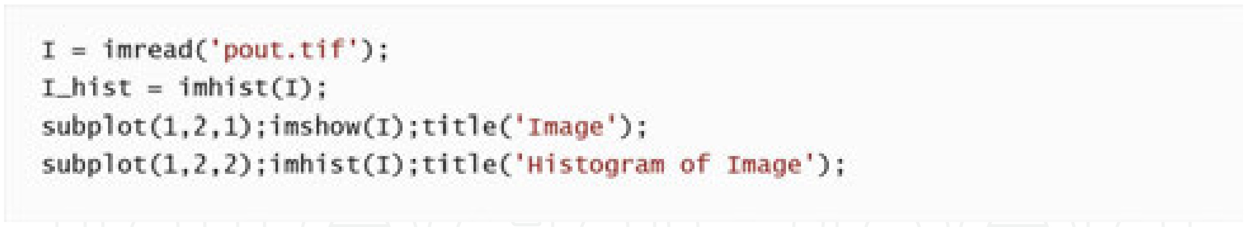
Gray value	Counting of pixel number
75	2
76	38
77	0
78	389
79	1245
80	0
81	1518

Table 3. Histogram of Specific Gray Values.

3.5. Histogram

Histogram counts the number of gray value of pixels in the images. Each pixel in the image has gray value between 0 and 255. As shown in **Table 3**, counting pixels give us information about image or objects in the image. The histogram of image is shown in **Figure 22**.

The histogram of image is calculated using imhist(image) function in the MATLAB.



```
I = imread('pout.tif');
I_hist = imhist(I);
subplot(1,2,1);imshow(I);title('Image');
subplot(1,2,2);imhist(I);title('Histogram of Image');
```

3.6. Histogram equalization

Histogram equalization is defined a technique for adjusting contrast of an image using all gray values to equalize as much as possible. Some situations work fine and image are shown very well; sometimes, it is not good and new image is darker than original image (**Figure 24**).

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

Table 4. An Image Pixel Gray Values.

We explain histogram equalization with an exam. You think about an image, and, its intensity mapping is shown below. There are eight possible gray levels from 0 to 7. If we apply histogram equalization to the image pixel gray values that are shown in **Table 4**, how new image histogram will be?

Step 1: Find histogram of the image (**Table 5**)

I	0	1	2	3	4	5	6	7
f(I)	2	5	5	4	2	2	3	2

Table 5. Histogram.

Step 2: Calculate Cumulative Frequency Distribution (CFD)

I	0 1	2	3	4	5	6	7
f(I)	2 5	5	4	2	2	3	2
CFD	2 2 + 5 = 7	7 + 5 = 12	12 + 4 = 16	16 + 2 = 18	18 + 2 = 20	20 + 3 = 23	23 + 2 = 25

Table 6. Calculate Cumulative Frequency Distrubiton (CFD).

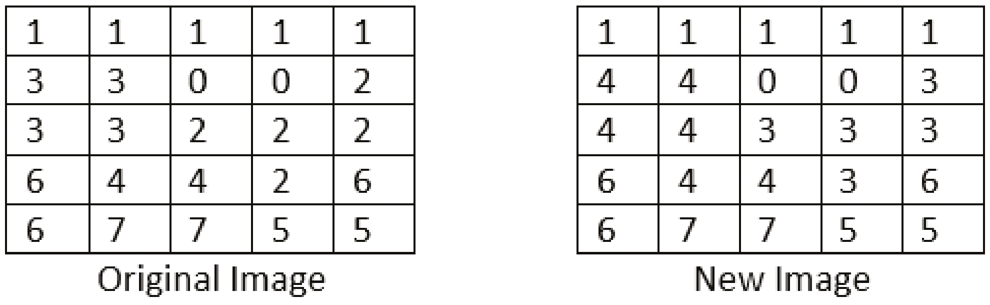
Step 3: Calculate new pixel gray value using **Equation 3**

$$h(v) = floor\left(\frac{CFD(v) - CFD_{min}}{(MxN) - CFD_{min}} x(L-1)\right) \tag{3}$$

h is new gray value, v is pixel number, MxN is image row and column value, L is gray level (in our image L is 8)

If we calculate the 4 number pixel;

$$h(4) = floor((16-2)/((5x5)-2)x(8-1)) \cong floor(4,26) \cong 4$$



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

Original Image

1	1	1	1	1
4	4	0	0	3
4	4	3	3	3
6	4	4	3	6
6	7	7	5	5

New Image

Figure 23. Original and new image gray values.

After all pixel gray values are calculated using Equation 3 the results of new gray values will be like in **Table 7**.

I	0	1	2	3	4	5	6	7
f(I)	2	5	5	4	2	2	3	2
CFD	2	7	12	16	18	20	23	25
h	0	1	3	4	4	5	6	7

Table 7. New Gray Values.

After histogram equalization is applied to the image, new gray values are shown in the **Figure 23**.

```
I = imread('pout.tif');
I_histeq = histeq(I);
subplot(2,2,1);imshow(I);title('Original Image');
subplot(2,2,2);imhist(I); title('Histogram of Original Image');
subplot(2,2,3);imshow(I_histeq);title('New Image');
subplot(2,2,4);imhist(I_histeq);title('Histogram of New Image');
```

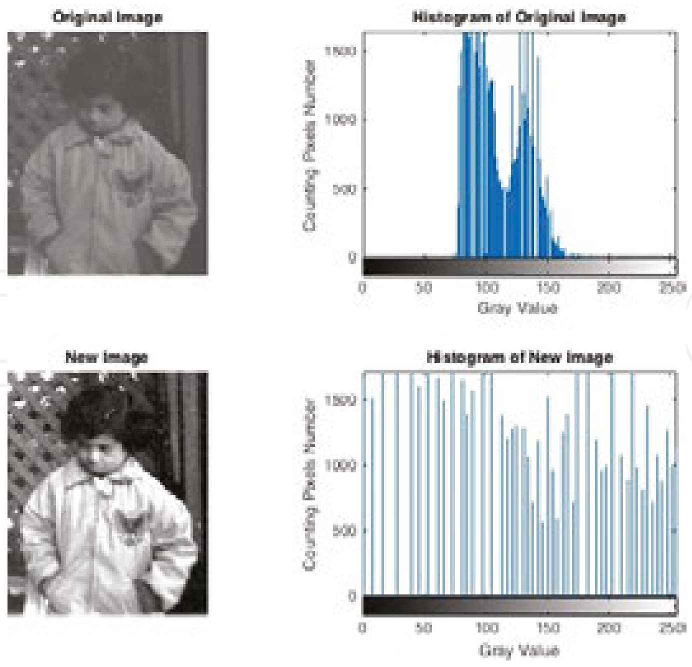


Figure 24. Histogram equalization.

MATLAB Image Processing Toolbox has the different filter types as shown in **Table 8**.

4. Color

Humans have very good photosensitive devices that are called eyes. Newton discovered that the light has different color spectrum passing through the glass prism. We think human eye is a glass prism that is called the lens. The lens focuses light to the retina of eyes. So that, humans see visible color spectrum of light reflected from the objects. Color spectrum is shown in the **Figure 25**. Human senses wavelength of light between 400 and 700 nm.

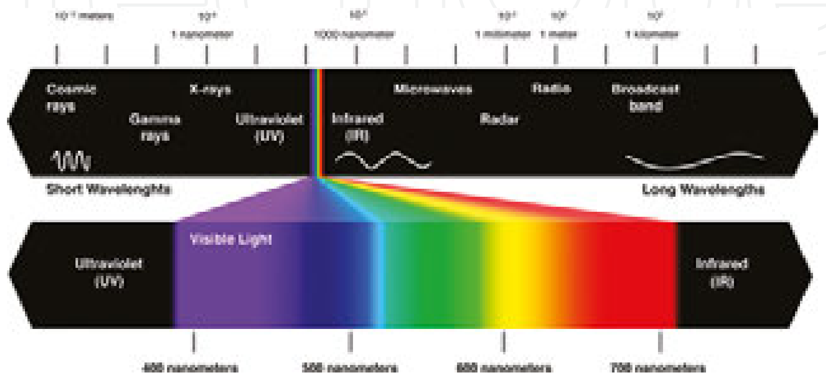


Figure 25. Color spectrum.

Eyes see colors as combining of primary that are Red (R), Green (G), and Blue (B). So that, all visible colors are produced from primary colors. Secondary colors, which are produced with adding of primary colors, are Yellow (Red + Green), Magenta (Red + Blue), and Cyan (Green + Blue) as shown in **Figure 26**.

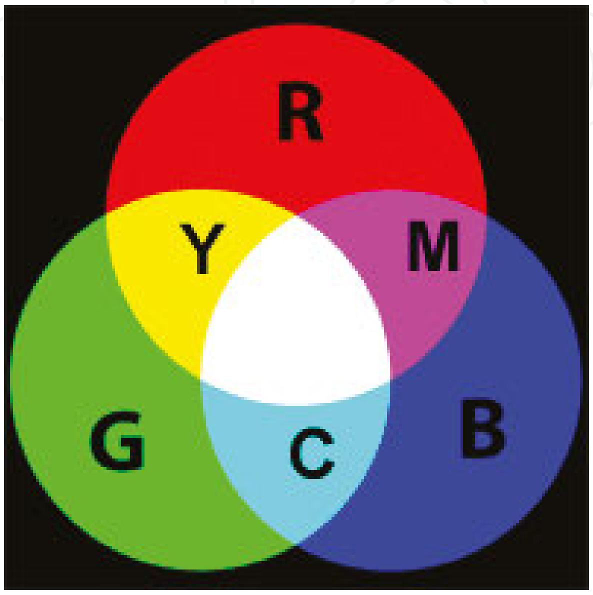
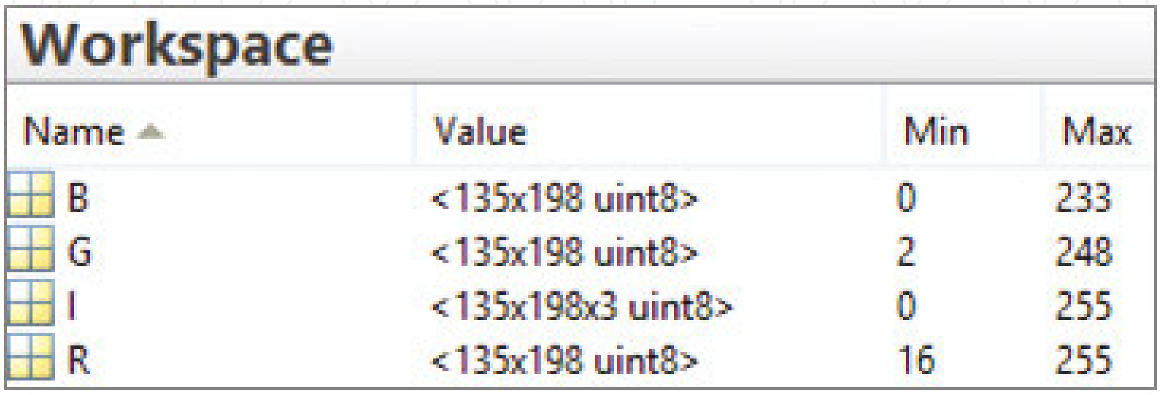


Figure 26. Primary and secondary colors.

In the MATLAB Image Processing Toolbox, a color image has three-dimensional uint8 (8-bit unsigned integer) data. Each dimension corresponds to a color channel that is Red, Green, or Blue channel. If we want, we can process each color channel. As shown in **Figure 27**, each color channel splits from image.



Workspace

Name	Value	Min	Max
B	<135x198 uint8>	0	233
G	<135x198 uint8>	2	248
I	<135x198x3 uint8>	0	255
R	<135x198 uint8>	16	255

Figure 27. R, G, B channel values in the MATLAB workspace.

```
I = imread('onion.png');
imshow(I);
% Red Channel
R = I(:,:,1);
% Green Channel
G = I(:,:,2);
% Blue Channel
B = I(:,:,3);
subplot(2,2,1);imshow(I);title('Original Image');
subplot(2,2,2);imshow(R);title('R');
subplot(2,2,3);imshow(G);title('G');
subplot(2,2,4);imshow(B);title('B');
```

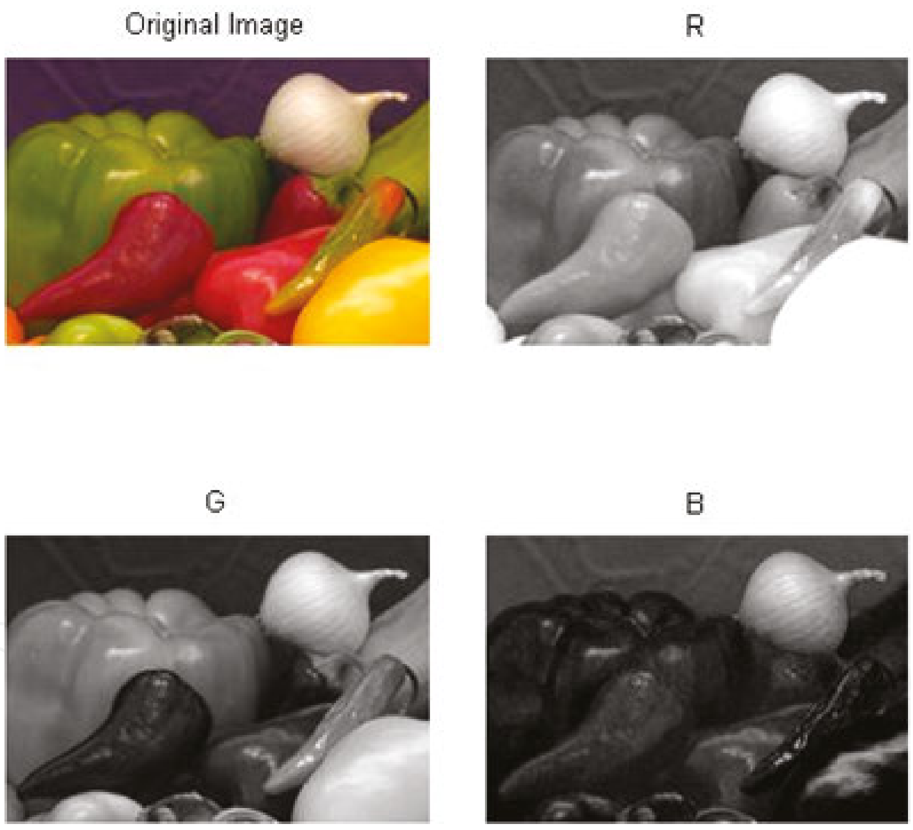


Figure 28. R, G, B channel.

4.1. HSI

As shown in **Figure 29** each color represents three components as H (Hue), S (Saturation), I (Intensity). The Hue, which can be defined rate of pure color, is an angle form between 0° and 360°. Red, Green, Blue are 0°, 120°, and 360°, and Yellow, Cyan, and Magenta are 60°, 180°, 300°. The Saturation, which shows how the color to be pure, takes value between [0, 1]. The intensity is the dimensions of lightness or darkness. The range of intensity is between 0 (black) and 1 (white).

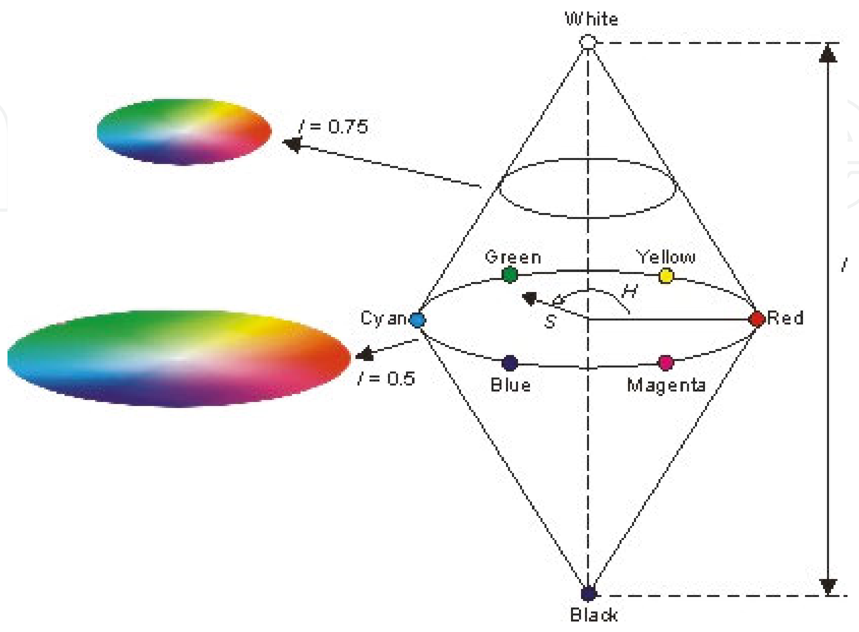
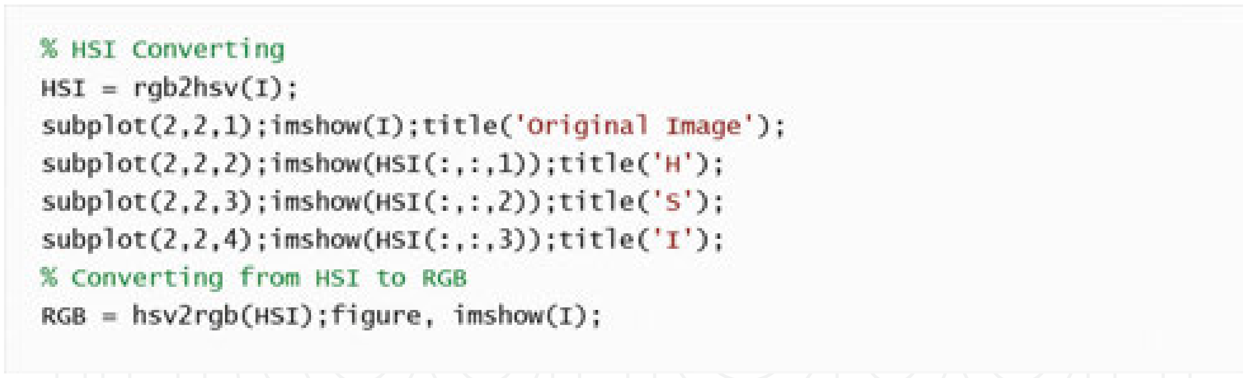


Figure 29. HSI components.



```
% HSI Converting
HSI = rgb2hsv(I);
subplot(2,2,1);imshow(I);title('Original Image');
subplot(2,2,2);imshow(HSI(:,:,1));title('H');
subplot(2,2,3);imshow(HSI(:,:,2));title('S');
subplot(2,2,4);imshow(HSI(:,:,3));title('I');
% Converting from HSI to RGB
RGB = hsv2rgb(HSI);figure, imshow(I);
```

MATLAB use rgb2hsv(image) or write script using **Eqs. (4)–(6)** for converting the color image to HSI components. If we want to convert from HSI image to RGB image, we use hsv2rgb(hsi image).

$$H = \begin{cases} \theta & if\ B \le G \\ 360 - \theta & if\ B > G \end{cases} with\,\theta = cos^{-1}\left\{ \frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}} \right\} \tag{4}$$

$$S = 1 - \frac{3}{(R+G+B)}\left[min\left(R,G,B\right)\right] \tag{5}$$

$$I = \frac{1}{3}\left[R+G+B\right] \tag{6}$$

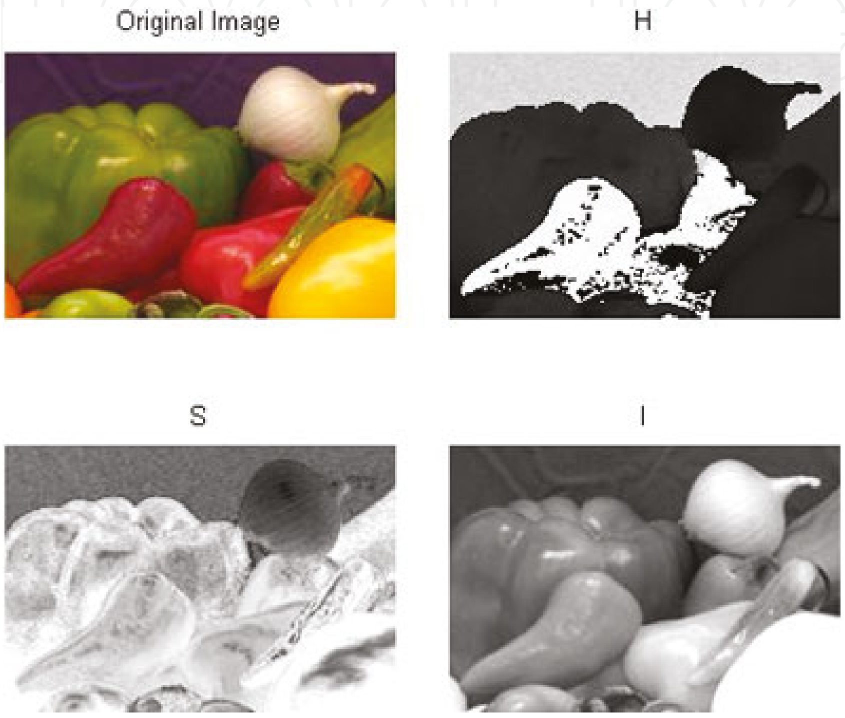


Figure 30. HIS of the image.

4.2. YIQ

YIQ, which is defined by the National Television System Committee (NTSC), produces the luminance and the chrominance. We use Equation 7 for producing of YIQ components from RGB image (**Figure 31**), and Equation 8 is used for converting from YIQ to RGB.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -.0274 & -0.322 \\ 0.211 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{7}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.986 & 0.621 \\ 1 & -.0272 & -0.649 \\ 1 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \tag{8}$$

```
RGB = imread('onion.png');
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y =  0.299   * R + 0.587   * G + 0.114   * B;
I = -0.14713 * R - 0.28886 * G + 0.436   * B;
Q =  0.615   * R - 0.51499 * G - 0.10001 * B;
YIQ = cat(3,Y,I,Q);
```

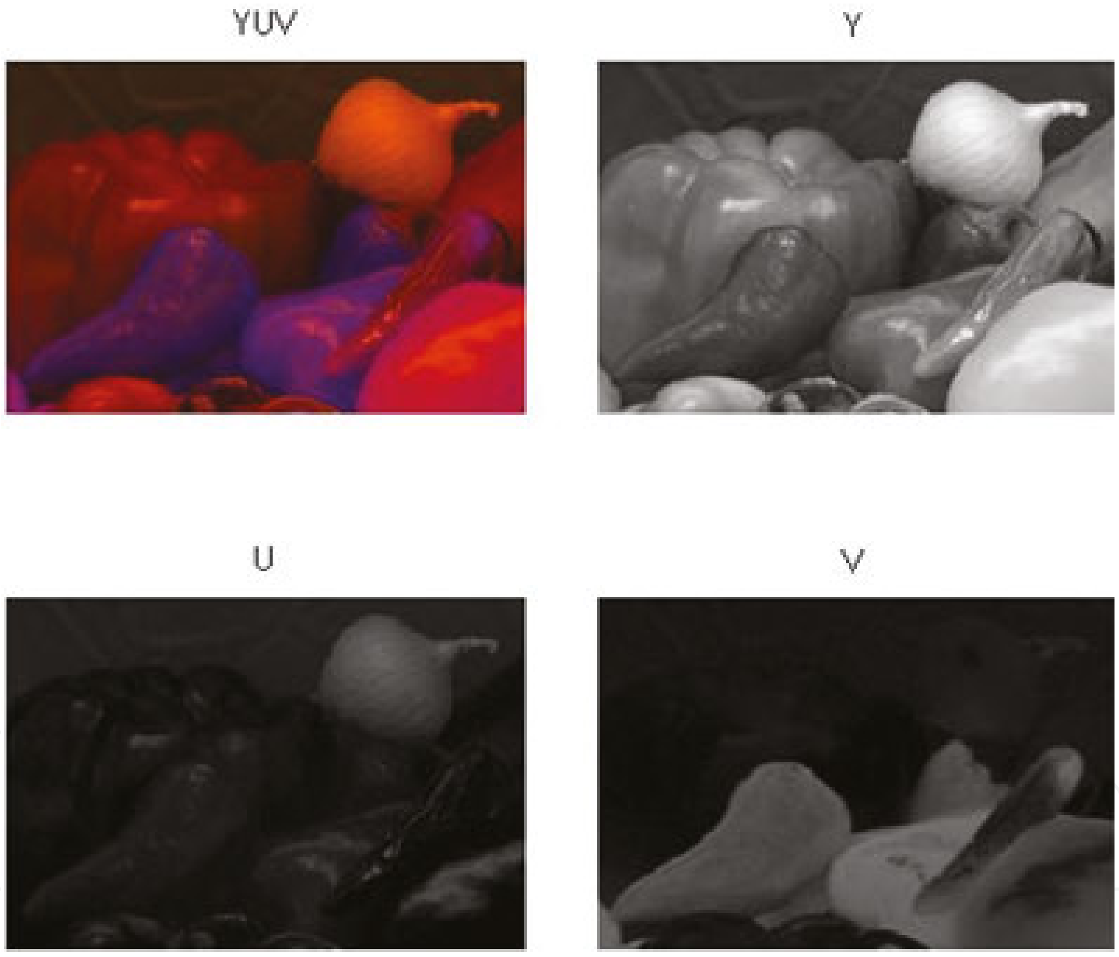


Figure 31. YIQ of the Image.

4.3. Gray image

Gray image is produced using Equation 9 by NTSC standards. However, we can calculate different methods, but MATLAB uses NTSC standards and it has rgb2gray(RGB image) function (**Figure 32**).

$$GI = 0.299R + 0.587G + 0.114B \tag{9}$$

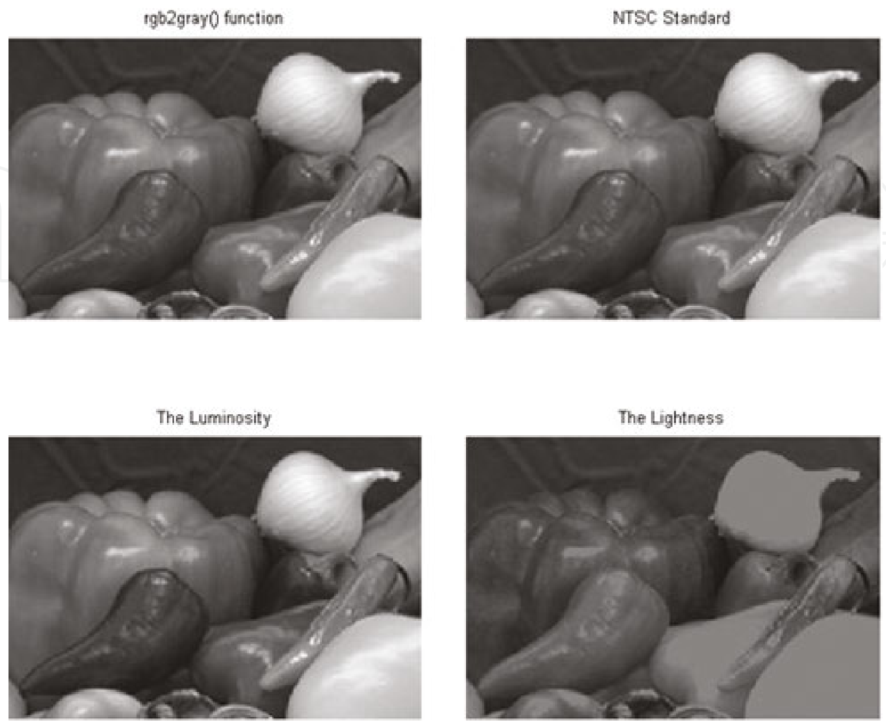


Figure 32. Gray image.

Other methods;

- The average; $GI = 0.33R + 0.33G + 0.33B$
- The lightness; $GI = (max(R,G,B) + (min(R,G,B))/2$
- The luminosity; $GI = 0.21R + 0.72G + 0.07B$

```
RGB = imread('onion.png');
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
RGBgraymet1 = rgb2gray(RGB);
RGBgraymet2 = 0.299.*R + 0.587.*G + 0.114.*B;
RGBgraymet3 = 0.21.*R + 0.72.*G + 0.07.*B;
RGBgraymet4 = (max(RGB, [], 3)+min(RGB, [], 3))/2;
subplot(2,2,1);imshow(RGBgraymet1);title('rgb2gray() function');
subplot(2,2,2);imshow(RGBgraymet2);title('NTSC Standard');
subplot(2,2,3);imshow(RGBgraymet3);title('The Luminosity');
subplot(2,2,4);imshow(RGBgraymet4);title('The Lightness');
```

http://dx.doi.org/10.5772/63028

5. Morphologic operations

MATLAB Image Processing Toolbox only use binary image for morphologic operations such as opening, closing, etc.

5.1. Structuring element

Structuring element (SE) is a shape that has different sizes (3 × 3, 4 × 4, 5 × 5, etc.) and shapes (**Figure 33**). SE is applied to an image for drawing results on how the objects change in the image (**Figure 34**). SE is generally used for dilation, erosion, opening, closing operations.

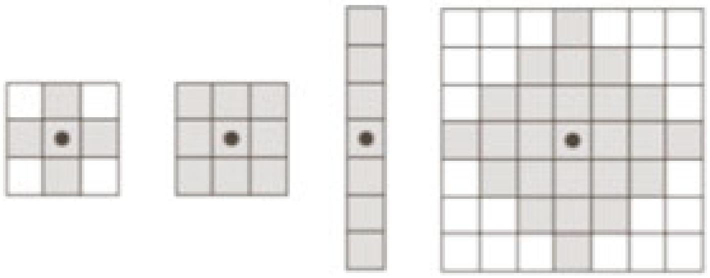


Figure 33. Different structuring elements.

5.2. Dilation

Dilation is a morphologic processing for growing an object in the binary image. It is shown with ⊕ image (**Figure 35**).

$$C = A \oplus B$$

C is the new image, A is the original image, and B is the structuring element.

```
se = strel('square', 3);
dilationsq = imdilate(binary, se);
se = strel('diamond', 3);
dilationdm = imdilate(binary, se);
se = strel('line', 9, 0);
dilationbl = imdilate(binary, se);
figure,
subplot(2,2,1);imshow(binary);title('Original Binary Image');
subplot(2,2,2);imshow(dilationsq);title('Dilation Image (3x3) Square SE');
subplot(2,2,3);imshow(dilationdm);title('Dilation Image (3x3) Diamond SE');
subplot(2,2,4);imshow(dilationbl);title('Dilation Image Line SE');
```

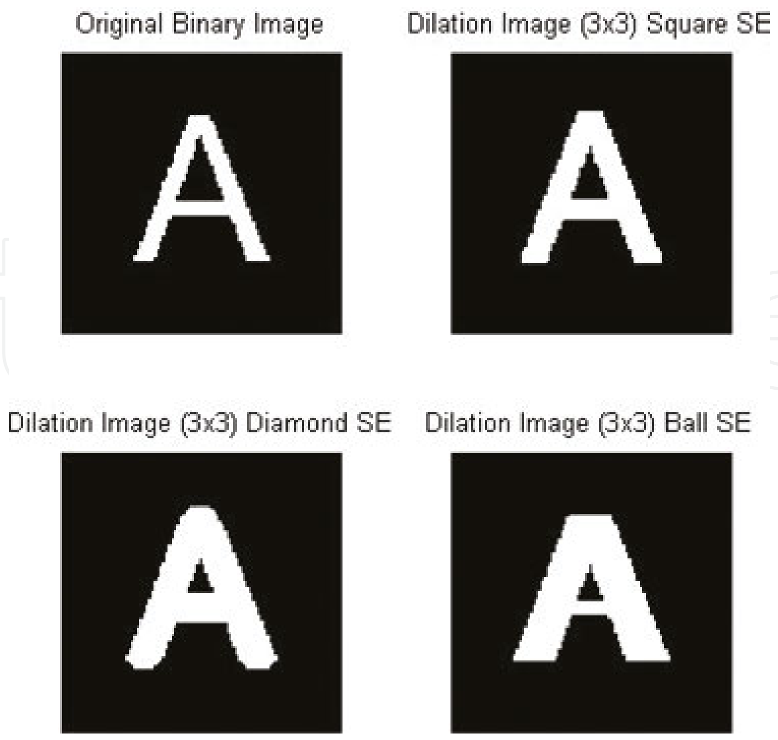
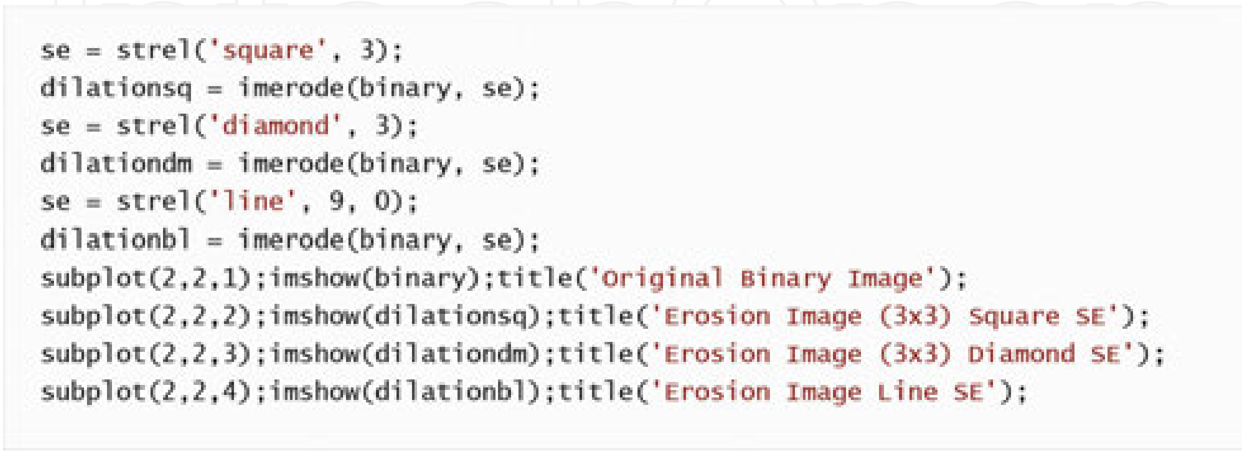

Original Binary Image
Dilation Image (3x3) Square SE
Dilation Image (3x3) Diamond SE
Dilation Image (3x3) Ball SE

Figure 34. Dilation.

5.3. Erosion

Erosion is the other morphologic operator of a binary image for using eroding the pixels of objects in the image. It is shown as ⊖ symbol.

$$C = A \ominus B$$



```
se = strel('square', 3);
dilationsq = imerode(binary, se);
se = strel('diamond', 3);
dilationdm = imerode(binary, se);
se = strel('line', 9, 0);
dilationbl = imerode(binary, se);
subplot(2,2,1);imshow(binary);title('Original Binary Image');
subplot(2,2,2);imshow(dilationsq);title('Erosion Image (3x3) Square SE');
subplot(2,2,3);imshow(dilationdm);title('Erosion Image (3x3) Diamond SE');
subplot(2,2,4);imshow(dilationbl);title('Erosion Image Line SE');
```

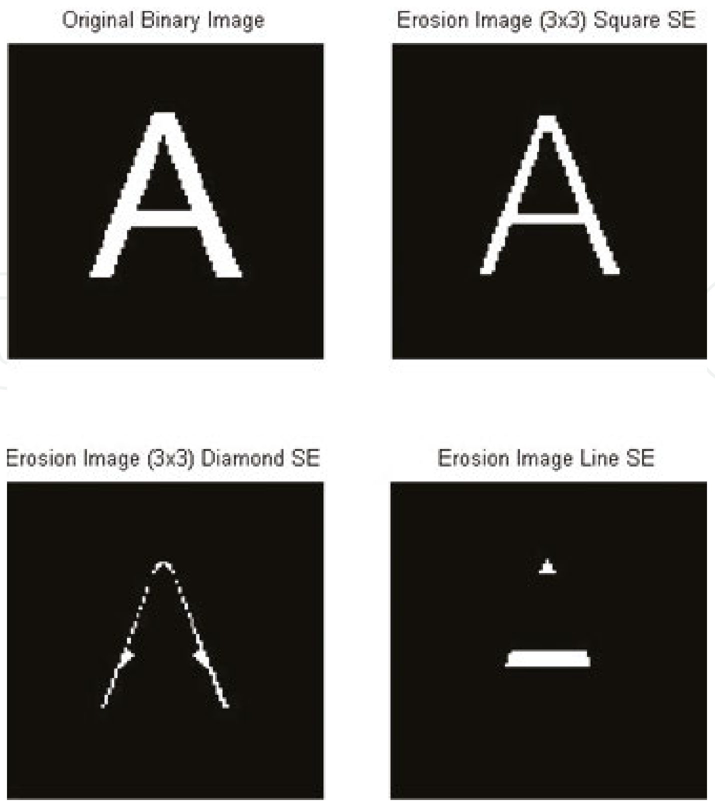


Figure 35. Erosion.

5.4. Opening and closing

As shown in (**Figure 36**), opening and closing are the combination of erosion and dilation operators as shown in Equations 10 and 11.

$$C = (A \ominus B) \oplus B \tag{10}$$

$$C = (A \oplus B) \ominus B \tag{11}$$

```
se = strel('ball', 10,10);
open = imopen(binary, se);
close = imclose(binary, se);
closeopen = imopen(imclose(binary,se), se);
subplot(2,2,1);imshow(binary);title('Original Binary Image');
subplot(2,2,2);imshow(open);title('Opening');
subplot(2,2,3);imshow(close);title('Closing');
subplot(2,2,4);imshow(closeopen);title('Closing + Opening');
```

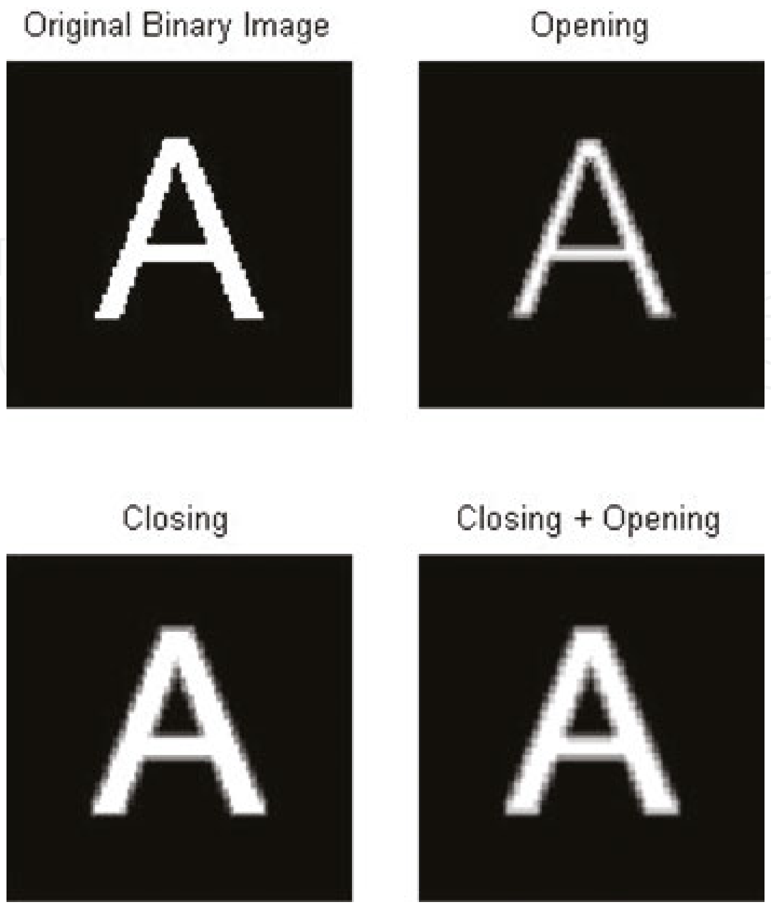


Figure 36. Opening and closing.

5.5. Convolution

Convolution is generally used for modifying the spatial characteristic of an image (**Figure 38**). In the convolution, a new pixel gray value is found by the weighted average pixels that are neighbor of it. Neighbor pixels gray value is weighted by a matrix coefficient that is called convolution kernel. According to the applications, kernel matrix has different sizes such as 3 × 3, 5 × 5, 7 × 7 (**Figure 37**).

Mathematical definition of convolution is shown in Equation 12;

$$G(x,y) = \sum_{n}^{j=-n} \sum_{m}^{i=-m} k(i,j) F(x-i, y-j) = k * F \tag{12}$$

k: convolution kernel matrix

F: processing image

if w and h are row and column of image ⇒(m = (w - 1)/2) | (n = (h - 1)/2)

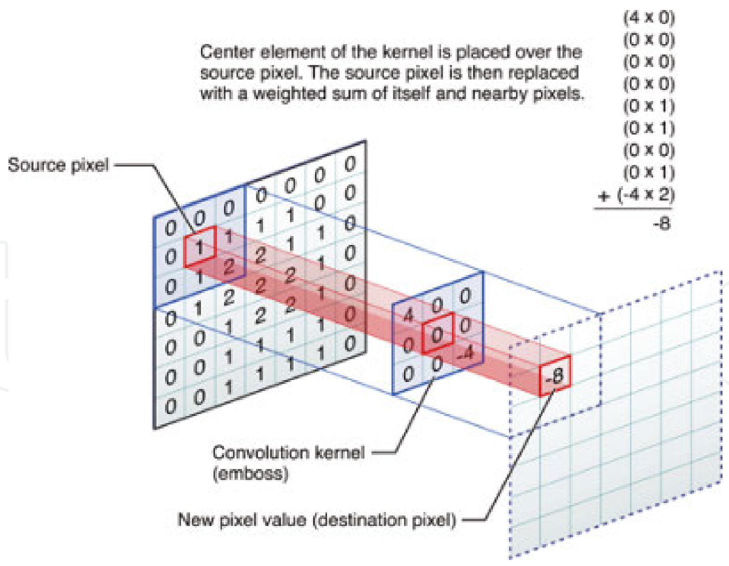

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Figure 37. Kernel matrices.

```
I = imread('onion.png');
H = fspecial('disk',10);
blurred = imfilter(I,H,'replicate');
H = fspecial('motion',20,45);
MotionBlur = imfilter(I,H,'replicate');
H = fspecial('sobel');
sobel = imfilter(I,H,'replicate');
subplot(2,2,1);imshow(I);title('Original Image');
subplot(2,2,2);imshow(blurred);title('Blur Kernel');
subplot(2,2,3);imshow(MotionBlur);title('Motion Blur Kernel');
subplot(2,2,4);imshow(sobel);title('Sobel Kernel');
```

MATLAB Image Processing Toolbox has the different filter types as shown in **Table 8**.

Value	Description
average	Averaging filter
disk	Circular averaging filter (pillbox)
gaussian	Gaussian low-pass filter
laplacian	Approximates the two-dimensional Laplacian operator
log	Laplacian of Gaussian filter
motion	Approximates the linear motion of a camera
prewitt	Prewitt horizontal edge-emphasizing filter
sobel	Sobel horizontal edge-emphasizing filter

Table 8. MATLAB Image Processing Toolbox Filter Types.

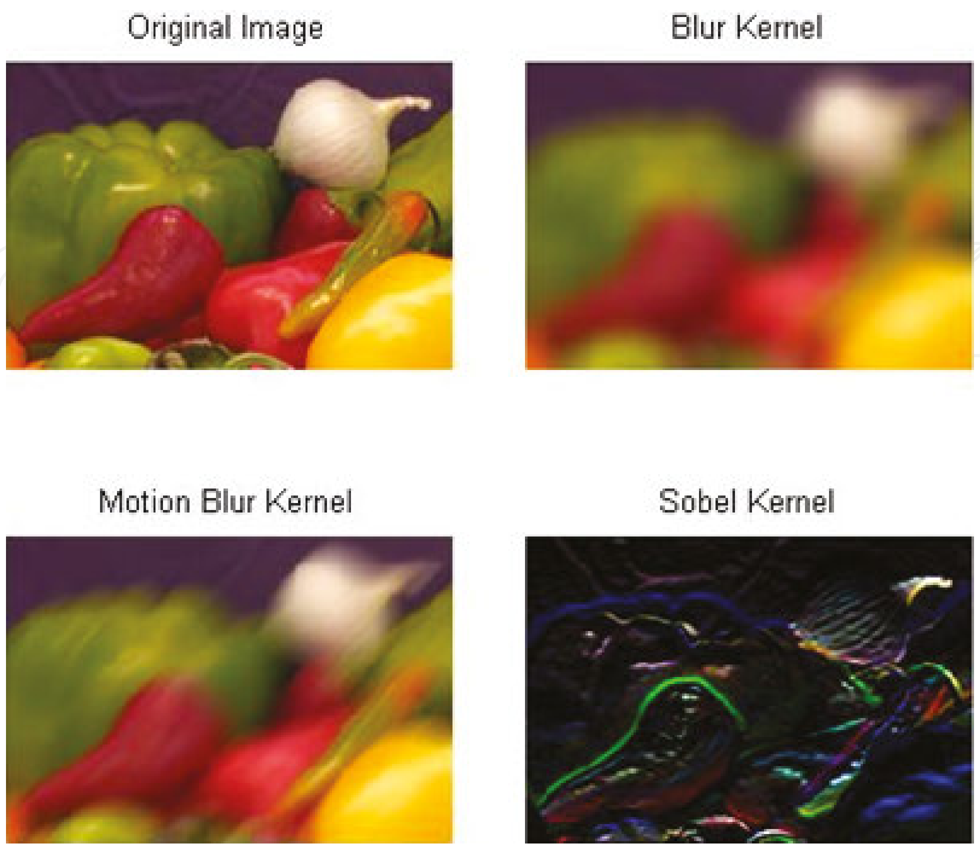


Figure 38. Applying different filters.

5.6. Edge detection

Edge detection is used for finding the border of objects in the image (**Figure 39**). Common edge detection algorithms are Sobel, Canny, Prewitt, Roberts, etc.

```
I = imread('pout.tif');
Iprewitt = edge(I,'prewitt');
Icanny = edge(I,'canny');
Isobel = edge(I,'sobel');
subplot(2,2,1);imshow(I);title('Original Image');
subplot(2,2,2);imshow(Iprewitt);title('Prewitt');
subplot(2,2,3);imshow(Icanny);title('Canny');
subplot(2,2,4);imshow(Isobel);title('Sobel');
```

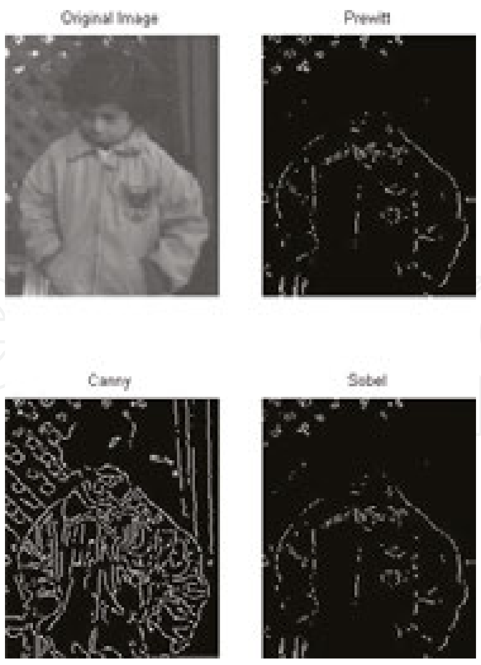


Figure 39. Edge detection.

5.7. Labeling

Pixels are assigned different labels because of belonging to different regions (or components or objects). In **Figure 40**, the objects in the image have the different label values and show different colors in the MATLAB.

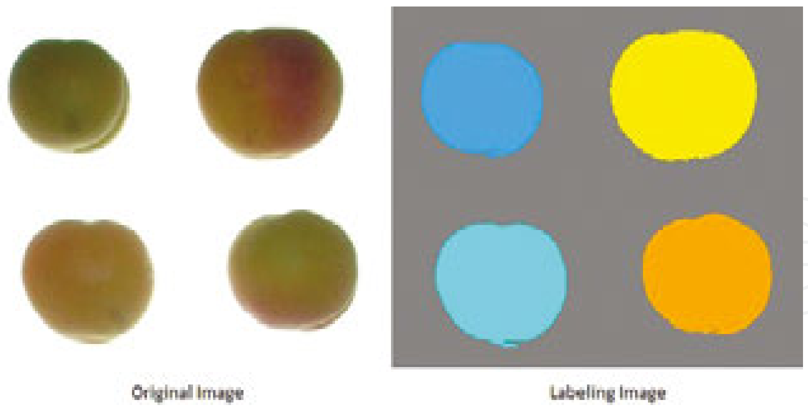


Figure 40. Labeling.

[Boundary,Labels] = bwboundaries(binary image, ‘noholes') function uses for labeling. Firstly, the image must be binary image, if it is not, you must convert to binary image. Secondly, all objects must be white (1) and background must be black (0) for using ‘noholes’ method. We use this function with two variables. One of them is address of boundary pixels, and other one is label numbers and their addresses.

6. Sample application

The last section in this chapter is a sample application that is about extraction of some morphological features of multiple apricots in a digital image. Firstly, original and background digital images are read (**Figures 41**, **42**). After that, cropped original image is subtracted from background image. Cropping process is used for extracting specific area from original image (**Figure 42**). Subtracted image (**Figure 44**) converts to gray image as shown in **Figure 45**. Thresholding process is applied to gray image for converting binary image (**Figure 46**). Sometimes, some artifacts can occurred in binary images. Before labelling, connecting pixel groups which are smaller than specific value (smaller than 50 px in this application) are removed (**Figure 47**). After labelling (**Figure 48**), we can find all objects morphological features as shown in **Figure 49**.

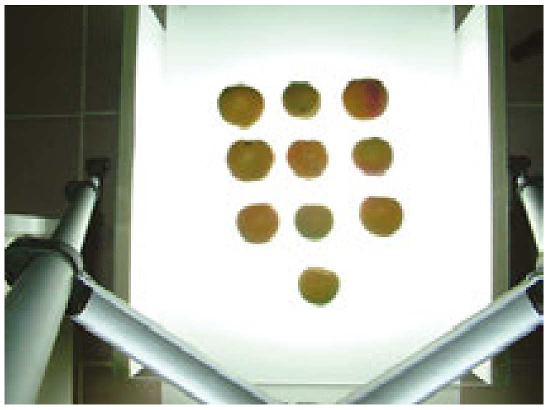


Figure 41. Original digital image.

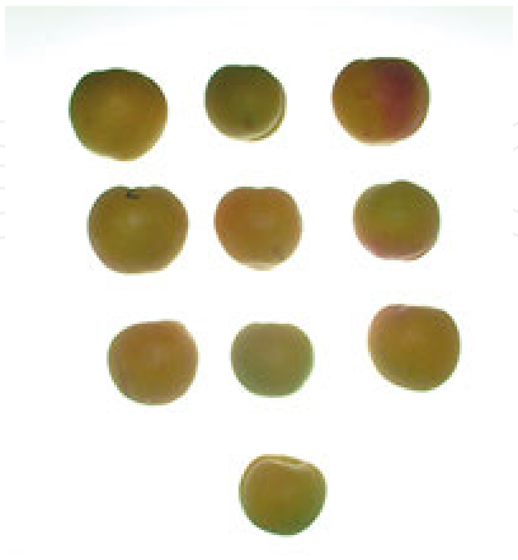


Figure 42. Background image.

Figure 43. Background image.

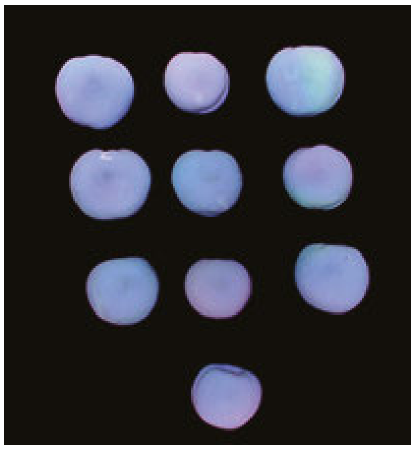


Figure 44. Subtracting from background to cropped image.

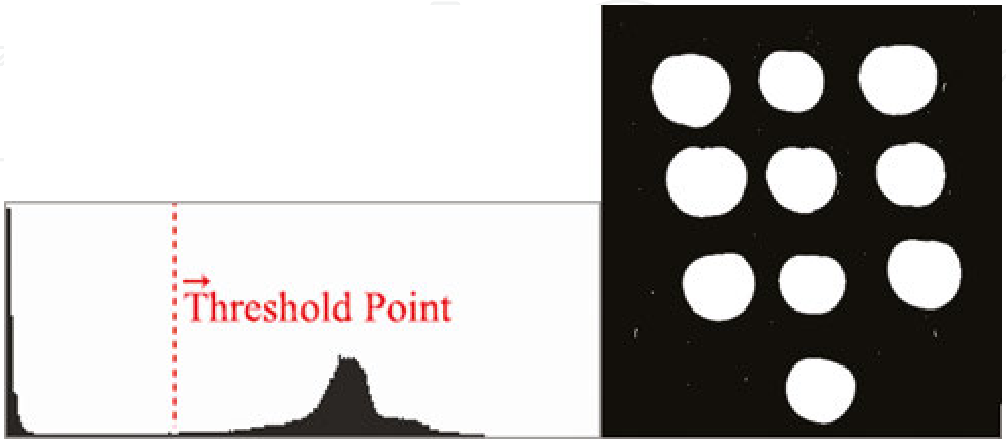


Figure 45. Converting gray image.

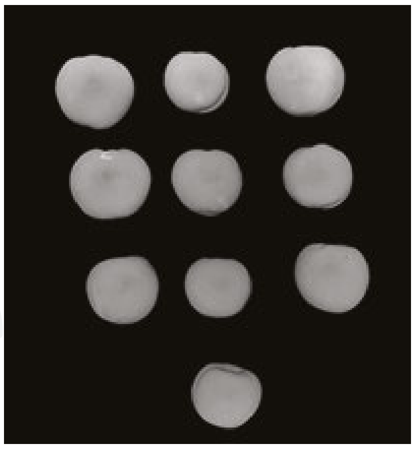


Figure 46. Thresholding.

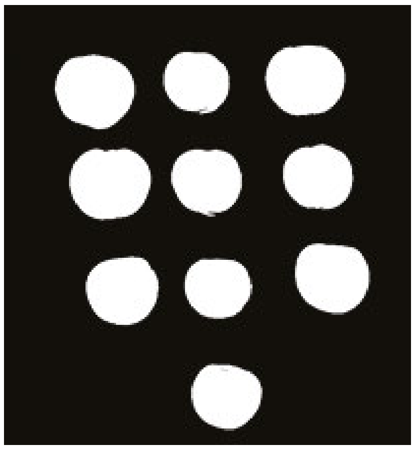


Figure 47. Remove artifacts.

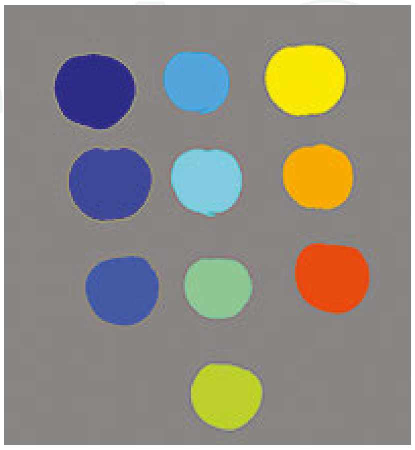


Figure 48. Labelling.

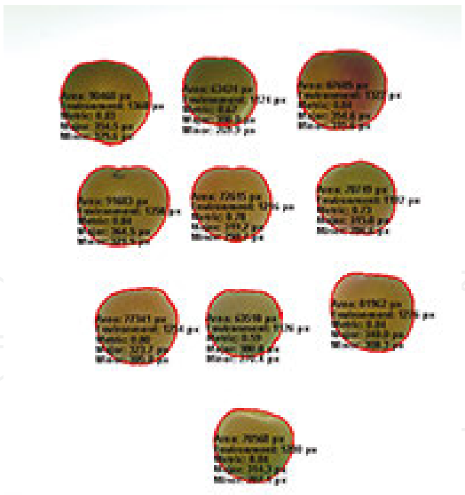
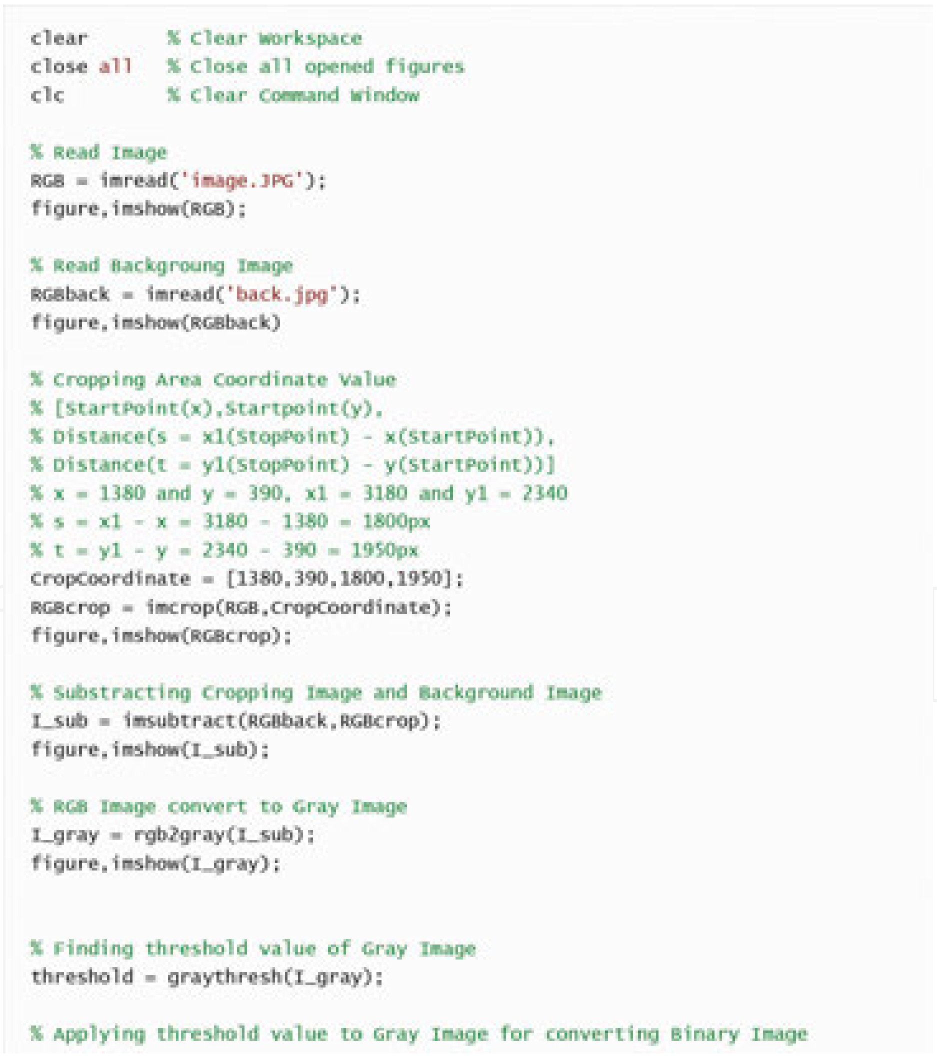
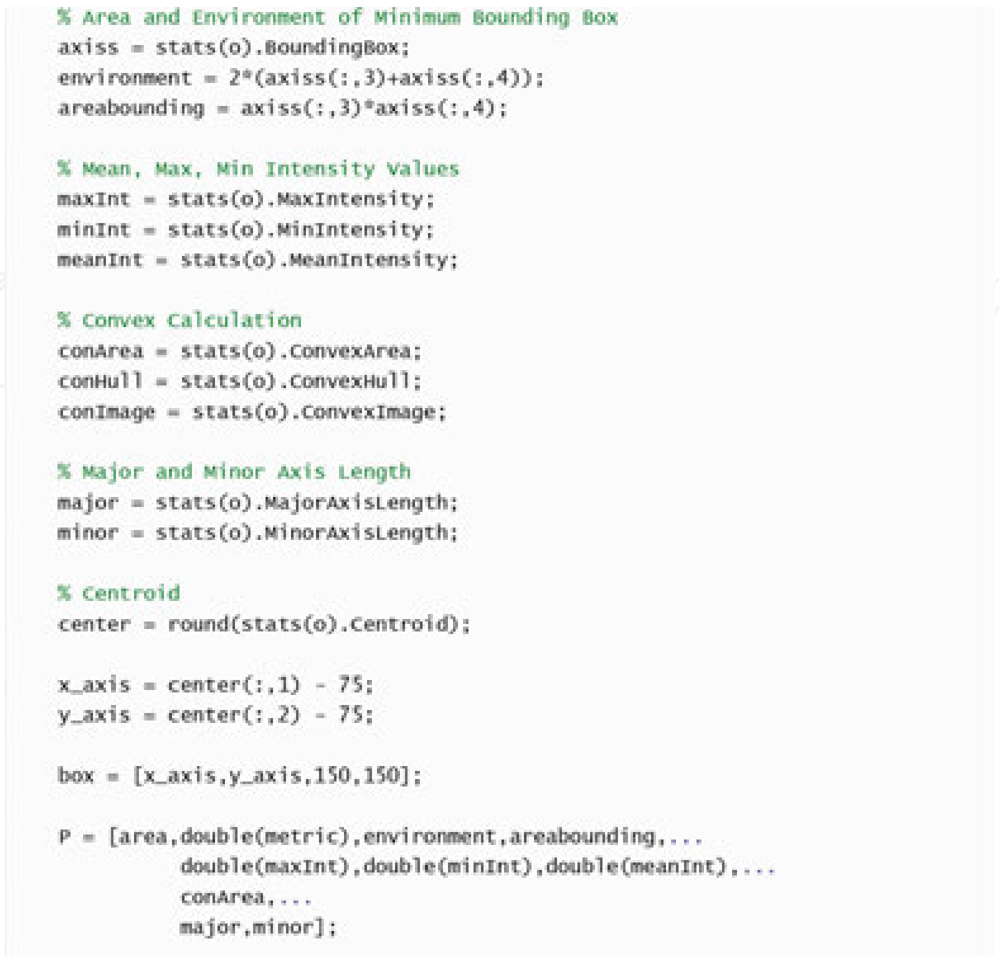


Figure 49. Show morphological features.

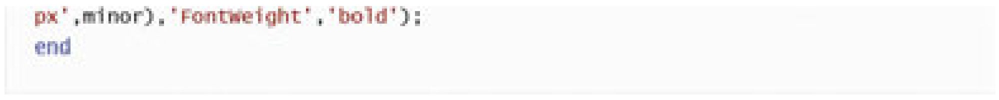


```
clear         % Clear Workspace
close all     % Close all opened figures
clc           % Clear Command Window

% Read Image
RGB = imread('image.JPG');
figure,imshow(RGB);

% Read Backgroung Image
RGBback = imread('back.jpg');
figure,imshow(RGBback)

% Cropping Area Coordinate Value
% [StartPoint(x),Startpoint(y),
% Distance(s = x1(StopPoint) - x(StartPoint)),
% Distance(t = y1(StopPoint) - y(StartPoint))]
% x = 1380 and y = 390, x1 = 3180 and y1 = 2340
% s = x1 - x = 3180 - 1380 = 1800px
% t = y1 - y = 2340 - 390 = 1950px
CropCoordinate = [1380,390,1800,1950];
RGBcrop = imcrop(RGB,CropCoordinate);
figure,imshow(RGBcrop);

% Substracting Cropping Image and Background Image
I_sub = imsubtract(RGBback,RGBcrop);
figure,imshow(I_sub);

% RGB Image convert to Gray Image
I_gray = rgb2gray(I_sub);
figure,imshow(I_gray);


% Finding threshold value of Gray Image
threshold = graythresh(I_gray);

% Applying threshold value to Gray Image for converting Binary Image
```

```
bw = im2bw(I_gray,threshold);
figure,imshow(bw);

% If count of connecting pixel groups is smaller than 500,they are removed
bw = bwareaopen(bw,500);
figure,imshow(bw)

% If objects in the image have not holes, they are labeling.
% B is boundary coordinate values, L is labeling values.
[B,L] = bwboundaries(bw,'noholes');

% Show labeling objects in different colors.
figure,imshow(label2rgb(L, @jet, [.5 .5 .5]))

figure,imshow(RGBcrop);
% Open the last figure (Cropping Image)
hold on

% Plot the boundary of objects in the image
for n = 1:length(B)
  boundary = B{n};
  plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 2)
end

% Finding specific morphological values
stats = regionprops(L,bw,'Area','Centroid','BoundingBox',...
     'MaxIntensity','MinIntensity','MeanIntensity','ConvexArea',...
     'ConvexHull','ConvexImage','Eccentricity','MajorAxisLength',...
     'MinorAxisLength','Solidity','EquivDiameter','Extent');

% Each object has the different morphological values, so that, finding
% all values and saving to a variable.

% How many objects in the image are calculated length(B).
for o = 1:length(B)

  boundary = B{o};
  % Calculating Area and Metric
  delta_sq = diff(boundary).^2;
  perimeter = sum(sqrt(sum(delta_sq,2)));
  area = stats(o).Area;
  metric = 4*pi*area/perimeter^2;
```



```
% Area and Environment of Minimum Bounding Box
axiss = stats(o).BoundingBox;
environment = 2*(axiss(:,3)+axiss(:,4));
areabounding = axiss(:,3)*axiss(:,4);

% Mean, Max, Min Intensity Values
maxInt = stats(o).MaxIntensity;
minInt = stats(o).MinIntensity;
meanInt = stats(o).MeanIntensity;

% Convex Calculation
conArea = stats(o).ConvexArea;
conHull = stats(o).ConvexHull;
conImage = stats(o).ConvexImage;

% Major and Minor Axis Length
major = stats(o).MajorAxisLength;
minor = stats(o).MinorAxisLength;

% Centroid
center = round(stats(o).Centroid);

x_axis = center(:,1) - 75;
y_axis = center(:,2) - 75;

box = [x_axis,y_axis,150,150];

P = [area,double(metric),environment,areabounding,...
          double(maxInt),double(minInt),double(meanInt),...
          conArea,...
          major,minor];
```



```
px',minor),'FontWeight','bold');
end
```

Author details

Mahmut Sinecen

Address all correspondence to: mahmut@adu.edu.tr

Adnan Menderes University, Computer Engineering Department, Aydin, Turkey

References

[1] Rafael C. Gonzalez, Richard E. Woods. Digital image processing. 3rd ed. United States of America: Pearson; 2008. 943 p.

[2] John C. Russ. The image processing handbook. 5th ed. United States of America: CRC Press; 2007. 795 p.

[3] Henri Maitre. Image processing. 1st ed. Great Britain: Wiley-ISTE; 2008. 359 p.

[4] Maria Petrou; Costas Petrou. Image processing the fundamentals. 2nd ed. Singapore: Wiley; 2010. 781 p.

[5] John D. Cook. Three algorithms for converting color to grayscale [Internet]. 24.08.2009. http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/. Accessed 02.01.2016

[6] Shindong Kang. http://www.slideshare.net/uspace/ujavaorg-deep-learning-with-convolutional-neural-network [Internet]. 26.05.2015 . http://www.slideshare.net/uspace/ujavaorg-deep-learning-with-convolutional-neural-network. Accessed 29.12.2015

[7] Sung Kim. Applications of Convolution in Image Processing with MATLAB [Internet]. 20.08.2013. http://www.math.washington.edu/~wcasper/math326/projects/sung_kim.pdf. Accessed 12.12.2016