Ex. No.	ACQUIRE AND DISPLAY AN IMAGE, NEGATIVE OF AN IMAGE (BINARY & GRAY SCALE)	Date

AIM:

The aim is to acquire an image, convert it into its negative, and display both the original and negative images in both binary and grayscale formats using MATLAB.

SOFTWARE REQUIRED:

MATLAB 2013b

THEORY:

Image Acquisition: To acquire an image, you can use the `imread` function in MATLAB. This function reads an image file and stores it as a matrix. The image can be in various formats such as JPEG, PNG, or BMP.

Negative of an Image:

For Grayscale Images: The negative of a grayscale image can be obtained by subtracting each pixel value from the maximum intensity value. For an 8-bit image, this maximum value is 255.

For Binary Images: In binary images, where pixel values are typically 0 or 1, the negative is obtained by swapping 0s with 1s and vice versa.

Displaying Images: You can use the `imshow` function in MATLAB to display images. For binary images, you may want to adjust the colormap to have a clear representation of 0s and 1s.

PROCEDURE:

1. Image Acquisition - Read the original image with the image filename
2. Negative of an Image:
   a) Convert the original image to grayscale if needed
   b) Calculate the negative of the grayscale image
   c) For binary images, you should threshold the image first
   d) Assuming your binary image is already threshold
   e) If not, use an appropriate thresholding technique
   f) Adjust the threshold value as needed
   g) Calculate the negative of the binary image
3. Displaying Images:
   a) Display the original grayscale image
   b) Display the negative grayscale image
   c) Display the original binary image
   d) Display the negative binary image
4. Save the displayed images to files using `imwrite` if needed.

PROGRAM:

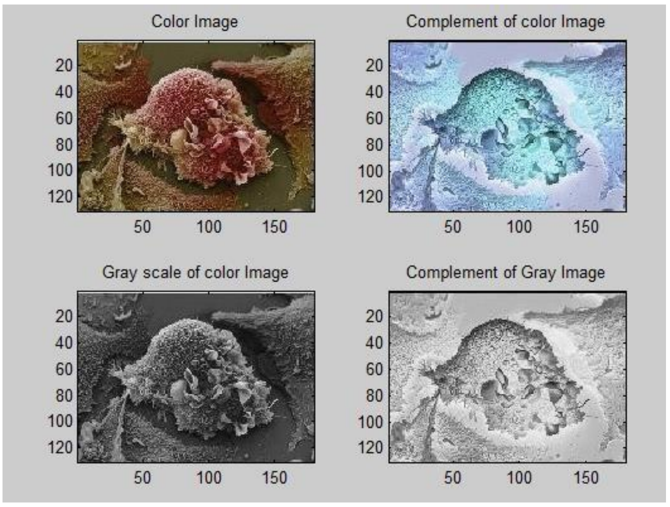
```
% Red Blue and Green and Gray Components
 i=imread('cancercell.jpg');
subplot(3,2,1); imshow(i); title('Original Image');
%Red Component
r=i(:,:,1);
subplot(3,2,2); imshow(r);title('Red Component');
%Green Component
g=i(:,:,2);
subplot(3,2,3); imshow(g); title('Green Component');
```

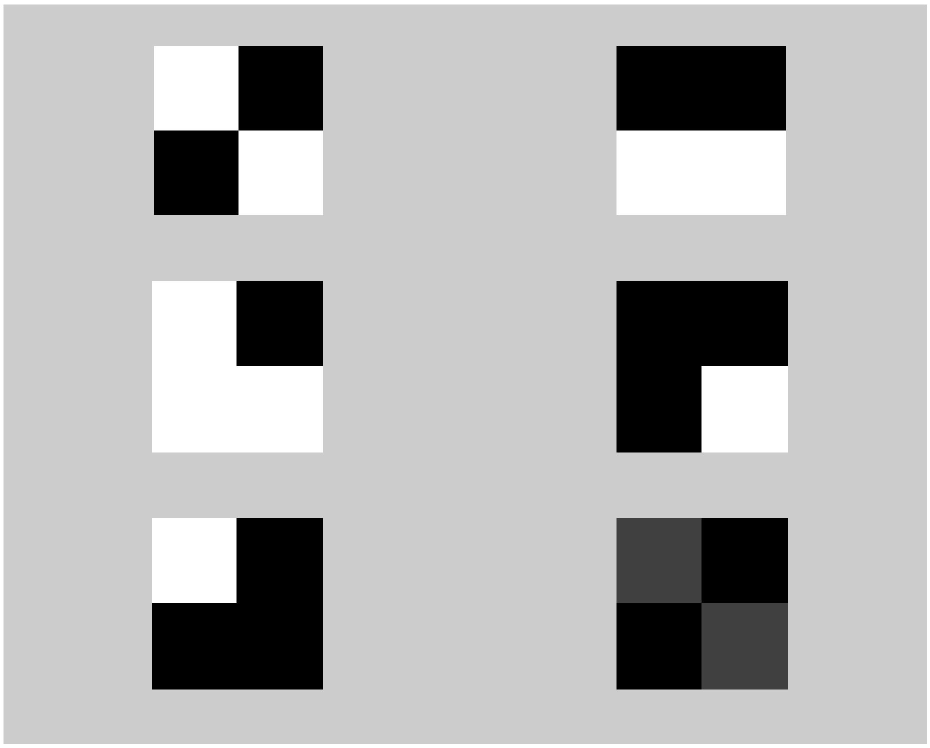
```
%Blue Component
b=i(:,:,3);
subplot(3,2,4); imshow(b); title('Blue Component');
%Color to Gray Image
rg=rgb2gray(i);
subplot(3,2,5); imshow(rg); title('Gray Image');
```

Complement, Converting and Simulation of an Image

```
% Display color Image, find its complement and convert to gray scale
I=imread('cancercell.jpg');
subplot(2,2,1); imshow(I); subimage(I);
title('Color Image');
c=imcomplement(I);
subplot(2,2,2); imshow(c); subimage(c);
title('Complement of color Image');
r=rgb2gray(I);
subplot(2,2,3); imshow(r); subimage(r);
title('Gray scale of color Image');
%Complement of Gray Image
b=imcomplement(r);
subplot(2,2,4); imshow(b); subimage(b);
title('Complement of Gray Image');
%Simulation of an Image (Arithmetic & Logic Operation)
a=ones(40); b=zeros(40); c=[a b;b
a]; d=[b b;a a]; A=10*(c+d);
M=c.*d;
S=c-d; D=c/4; figure;
subplot(3,2,1);  imshow(c);
subplot(3,2,2);  imshow(d);
subplot(3,2,3);  imshow(A);
subplot(3,2,4);  imshow(M);
subplot(3,2,5);  imshow(S);
subplot(3,2,6); imshow(D);
```

OUTPUT:





RESULT:

REVIEW QUESTIONS:

1. What MATLAB function is used to acquire an image from a file?
2. Why might it be necessary to convert a colour image to grayscale before processing it further?
3. How is the negative of a grayscale image calculated, and what is the significance of the value 255 in this context?
4. In binary image processing, how is the negative of a binary image typically obtained?
5. Can you describe a scenario where calculating the negative of an image is useful in image processing?

Ex. No.	IMPLEMENTATION OF RELATIONSHIPS BETWEEN PIXELS	Date

AIM:
The aim is to implement relationships between pixels in a digital image.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Relationships between pixels in an image refer to the interactions and dependencies among neighboring or non-neighboring pixels. These relationships are crucial for tasks such as noise reduction, feature extraction, object detection, and more. MATLAB represents an image as a matrix, with each element corresponding to a pixel. The central pixel's location is typically

denoted by its row and column coordinates in the matrix. MATLAB represents an image as a matrix, with each element corresponding to a pixel. To find a pixel's neighbors, we can specify a neighborhood size, such as a square or circular region, centered on the pixel of interest. MATLAB's array indexing allows for easy extraction of the neighboring pixels.

PROCEDURE:

1. Load the image into MATLAB.
2. Use the `imread` function in MATLAB to read the image.
3. Determine the specific relationship or operation you want to implement.
4. Implement the chosen relationship between pixels. Calculate the new pixel values based on a mapping function that stretches the intensity values.
5. Create a kernel or mask for the specific filtering operation (e.g., Gaussian, Sobel, or custom filters). Use convolution to apply the filter to the image.
6. For segmentation tasks, choose an appropriate method such as thresholding, region-growing, or edge-based segmentation.
7. Implement the chosen method to divide the image into distinct regions or objects.
8. Display the original image alongside the processed image to visualize the impact of the implemented relationship or operation.
9. Testing and Evaluation: Test the implemented relationship or operation on different images and evaluate its effectiveness for the intended image processing task.

PROGRAM:

```
% To find Neighbour of a given Pixel
a=magic(5);
disp(‘a=’); disp(a);
b=input('Enter the row < size of the Matrix');
c=input(' Enter the Column < size of matrix');
disp(‘Element’); disp(a(b,c));
% 4 Point Neighbour
N4=[a(b+1,c), a(b-1,c), a(b,c+1), a(b,c-1)];
disp(‘N4=’); disp(N4);
%8 Point Neighbour
N8=[a(b+1,c), a(b-1,c), a(b,c+1), a(b,c-1), a(b+1,c+1), a(b+1,c-1), a(b-1,c-1), a(b-1,c+1)];
disp(‘N8=’); disp(N8);
%Diagonal Neighbour
ND=[ a(b+1,c+1), a(b+1,c-1), a(b-1,c-1), a(b-1,c+1)];
disp(‘ND=’); disp(ND);
```

OUTPUT:

```
>> pixel
   17   24    1    8   15
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9

Enter the row < size of the Matrix 3
```

```
 Enter the Column < size of matrix 2
N4:
    12     5    13     4

N8:
    12     5    13     4    19    10    23     7

ND:
    19    10    23     7
```

RESULT:

REVIEW QUESTIONS:

1. What does it mean to implement relationships between pixels in image processing?
2. Give an example of an image processing task that relies on defining and using relationships between pixels.
3. How would you implement a pixel-wise contrast stretching operation in MATLAB, and why might this be useful in enhancing images?
4. What is image filtering, and why is it an essential technique in image processing?
5. Describe the process of applying a convolution filter to an image and provide an example of a filter that can be used for a specific purpose.

Ex. No.	ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS	Date

AIM: The aim is to analyze images using different color models, such as RGB, HSL, and CMYK, to gain insights into color-based features and properties within the images.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Color models are mathematical representations of colors in digital images. Common color models include RGB (Red, Green, Blue), HSL (Hue, Saturation, Lightness), and CMYK (Cyan, Magenta, Yellow, Key/Black).
In the RGB model, colors are represented by combining various intensities of red, green, and blue. It is the most common color model used in digital displays and cameras.
The HSL model focuses on human perception of colors. It represents colors based on their hue, saturation, and lightness. This model is valuable for color manipulation and image analysis.
CMYK is used primarily in color printing and subtractive color mixing. It represents colors using cyan, magenta, yellow, and key (black) components. Analyzing images in CMYK can help with color correction and print-related tasks.

PROCEDURE:
1. Load the image in MATLAB.
2. Convert the loaded image to different color models, such as RGB, HSL, and CMYK, using built-in functions or libraries. In MATLAB, you can use functions like `rgb2hsl` or `rgb2cmyk`.

```
 Enter the Column < size of matrix 2
N4:
   12     5    13     4

N8:
   12     5    13     4    19    10    23     7

ND:
   19    10    23     7
```

RESULT:

REVIEW QUESTIONS:

1. What does it mean to implement relationships between pixels in image processing?
2. Give an example of an image processing task that relies on defining and using relationships between pixels.
3. How would you implement a pixel-wise contrast stretching operation in MATLAB, and why might this be useful in enhancing images?
4. What is image filtering, and why is it an essential technique in image processing?
5. Describe the process of applying a convolution filter to an image and provide an example of a filter that can be used for a specific purpose.

Ex. No.	ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS	Date

AIM: The aim is to analyze images using different color models, such as RGB, HSL, and CMYK, to gain insights into color-based features and properties within the images.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Color models are mathematical representations of colors in digital images. Common color models include RGB (Red, Green, Blue), HSL (Hue, Saturation, Lightness), and CMYK (Cyan, Magenta, Yellow, Key/Black).
In the RGB model, colors are represented by combining various intensities of red, green, and blue. It is the most common color model used in digital displays and cameras.
The HSL model focuses on human perception of colors. It represents colors based on their hue, saturation, and lightness. This model is valuable for color manipulation and image analysis.
CMYK is used primarily in color printing and subtractive color mixing. It represents colors using cyan, magenta, yellow, and key (black) components. Analyzing images in CMYK can help with color correction and print-related tasks.

PROCEDURE:
1. Load the image in MATLAB.
2. Convert the loaded image to different color models, such as RGB, HSL, and CMYK, using built-in functions or libraries. In MATLAB, you can use functions like `rgb2hsl` or `rgb2cmyk`.

3. Calculate statistical metrics for each color channel in the chosen color model. These metrics may include mean, standard deviation, or histograms of pixel values.
4. Display the image in each color model to visually compare and understand how the image's appearance changes.
5. Apply color manipulations or corrections within the chosen color model to enhance or modify specific aspects of the image, such as brightness, contrast, or color balance.
6. Extract color-based features from the different color channels, which can be used for further analysis or object recognition.
7. Perform image segmentation using color information from the selected color model to separate objects or regions of interest in the image based on color.
8. Display the results of your analysis, such as histograms of color channels, segmented regions, or color-corrected images.

PROGRAM:

```
% Load an image
originalImage = imread('image.jpg'); % Replace 'image.jpg' with the image filename

% Display the original image
figure;
subplot(2, 2, 1);
imshow(originalImage);
title('Original Image');

% Convert the image to RGB
rgbImage = originalImage;

% Display the RGB image
subplot(2, 2, 2);
imshow(rgbImage);
title('RGB Image');

% Conversion to HSL (Custom function required)
% hslImage = rgb2hsl_custom(rgbImage);

% Display the HSL image
% subplot(2, 2, 3);
% imshow(hsl2rgb_custom(hslImage));
% title('HSL Image');

% Conversion to CMYK (Custom function required)
% cmykImage = rgb2cmyk_custom(rgbImage);

% Display the CMYK image
% subplot(2, 2, 4);
% imshow(cmyk2rgb_custom(cmykImage));
% title('CMYK Image');

% Analyze color-based features
% Dominant colors in RGB
rgbDominantColor = calculateDominantColor(rgbImage);
fprintf('Dominant color in RGB: R=%d, G=%d, B=%d\n', rgbDominantColor(1),
rgbDominantColor(2), rgbDominantColor(3));

% % Dominant colors in HSL (Custom function required)
% hslDominantColor = calculateDominantColor(hsl2rgb_custom(hslImage));
% fprintf('Dominant color in HSL: R=%d, G=%d, B=%d\n', hslDominantColor(1),
hslDominantColor(2), hslDominantColor(3));

% % Dominant colors in CMYK (Custom function required)
% cmykDominantColor = calculateDominantColor(cmyk2rgb_custom(cmykImage));
% fprintf('Dominant color in CMYK: R=%d, G=%d, B=%d\n', cmykDominantColor(1),
cmykDominantColor(2), cmykDominantColor(3));

% Calculate and plot color histograms
rgbHistogram = calculateColorHistogram(rgbImage);
% hslHistogram = calculateColorHistogram(hsl2rgb_custom(hslImage));
% cmykHistogram = calculateColorHistogram(cmyk2rgb_custom(cmykImage));

% Plot RGB histogram
figure;
subplot(3, 1, 1);
bar(rgbHistogram, 'r');
title('RGB Histogram');

% % Plot HSL histogram
% subplot(3, 1, 2);
% bar(hslHistogram, 'g');
% title('HSL Histogram');

% % Plot CMYK histogram
% subplot(3, 1, 3);
% bar(cmykHistogram, 'b');
% title('CMYK Histogram');

% Function to calculate dominant color (implementation required)

% % Custom RGB to HSL conversion function
% function hslImage = rgb2hsl_custom(rgbImage)
%     % Implement RGB to HSL conversion here
% end
```

% % Function to calculate dominant color

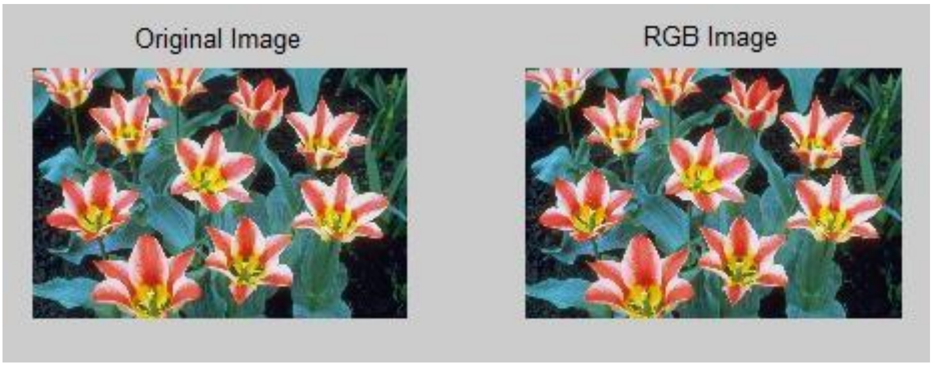
```
function dominantColor = calculateDominantColor(image)
   % Convert the image to double precision
   doubleImage = im2double(image);

   % Reshape the image into a 2D array of RGB values
```

```
    [height, width, ~] = size(doubleImage);
    reshapedImage = reshape(doubleImage, height * width, 3);

    % Perform k-means clustering (change k to the desired number of clusters)
    k = 5; % You can adjust this value
    [clusterIndices, ~] = kmeans(reshapedImage, k);

    % Find the largest cluster (i.e., the dominant color)
    clusterCounts = hist(clusterIndices, 1:k);
    [~, dominantCluster] = max(clusterCounts);

    % Calculate the RGB values of the dominant color
    dominantColor = mean(reshapedImage(clusterIndices == dominantCluster, :), 1);
end
```

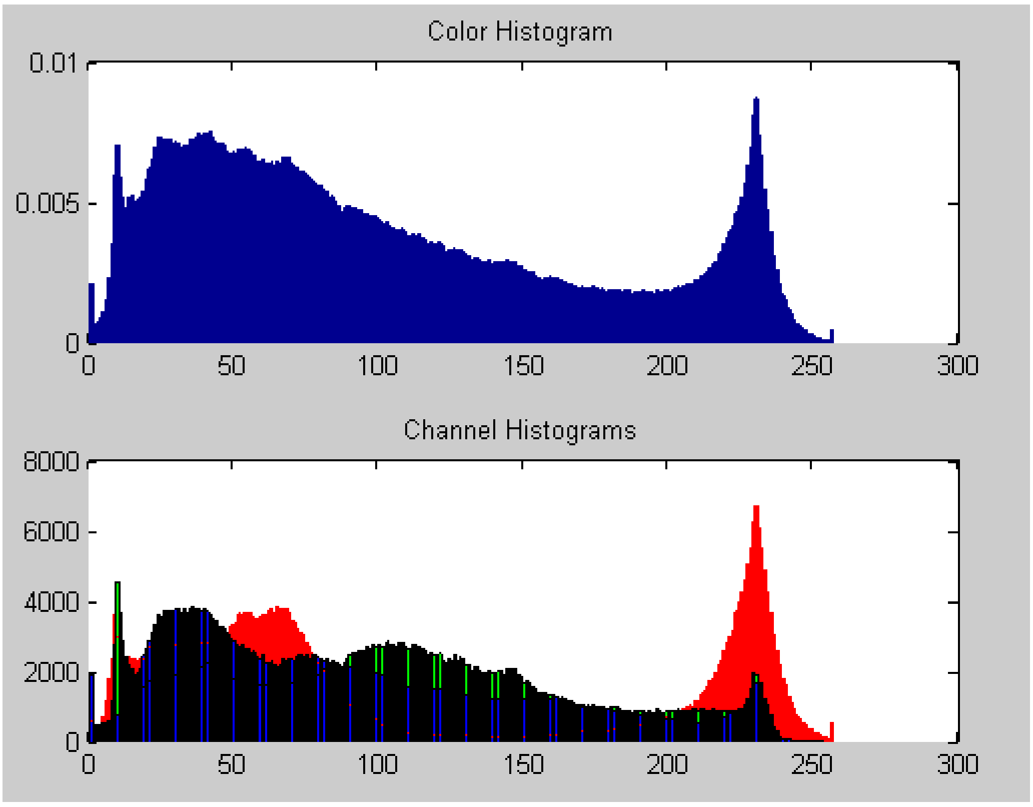
% % Function to calculate color histograms

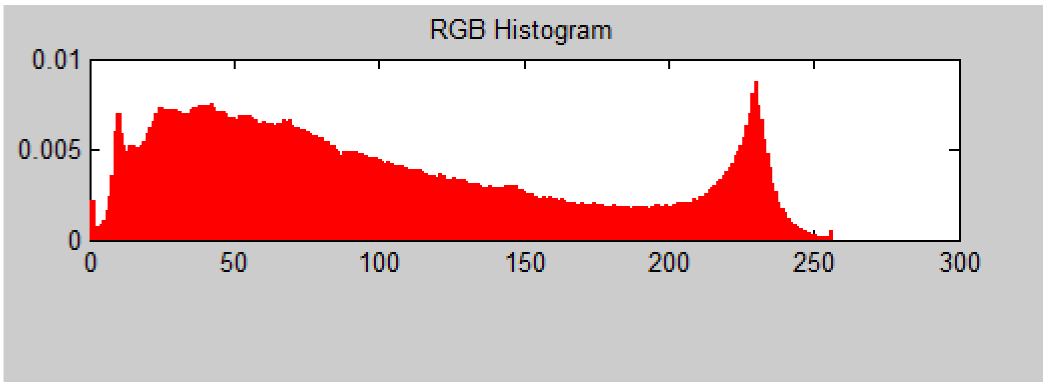
```
function colorHistogram = calculateColorHistogram(image)
    % Convert the image to double precision
    doubleImage = im2double(image);

    % Separate the RGB channels
    redChannel = doubleImage(:, :, 1);
    greenChannel = doubleImage(:, :, 2);
    blueChannel = doubleImage(:, :, 3);

    % Define the number of bins for the histogram
    numBins = 256; % You can adjust this value as needed

    % Calculate histograms for each channel
    redHist = imhist(redChannel, numBins);
    greenHist = imhist(greenChannel, numBins);
    blueHist = imhist(blueChannel, numBins);

    % Combine the individual histograms into a single histogram
    colorHistogram = [redHist, greenHist, blueHist];

    % Normalize the histogram to have values between 0 and 1
    colorHistogram = colorHistogram / sum(colorHistogram);

    % Plot the color histogram (optional)
    figure;
    subplot(2, 1, 1);
    bar(colorHistogram);
    title('Color Histogram');

    % Plot individual channel histograms (optional)
    subplot(2, 1, 2);
    bar(redHist, 'r');
    hold on;
    bar(greenHist, 'g');
    bar(blueHist, 'b');
    title('Channel Histograms');
end
```

OUTPUT:



```
Dominant color in RGB: R=1.290451e-01, G=1.700964e-01, B=1.570514e-01
```





RESULT:

REVIEW QUESTIONS:

1. What are the primary components of the RGB color model, and how are colors represented in this model?
2. Explain the significance of the HSL color model, including its components: Hue, Saturation, and Lightness.
3. How can you extract the hue channel from an image in the HSL color model using MATLAB, and what kind of image analysis tasks might benefit from this?
4. Describe the purpose and usage of the CMYK color model, particularly in the context of color printing.
5. What is the process in MATLAB for converting an image from RGB to CMYK, and how does it contribute to color correction and print-related tasks?

Ex. No.	**IMPLEMENTATION OF TRANSFORMATIONS OF AN IMAGE**	**Date**

AIM: The aim of experiment is to implement transformations on digital images, such as resizing, rotation, and flipping, to alter the appearance and structure of the image while maintaining image quality.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Image transformations are operations that modify the spatial and visual properties of digital images. Common transformations include resizing, rotation, cropping, and flipping. Image resizing involves changing the dimensions of the image, either by scaling it up (enlarging) or down (shrinking). This transformation is useful for adjusting image dimensions or preparing images for different display or printing sizes. Image rotation is the process of changing the image's orientation by a specific angle, often 90, 180, or 270 degrees. It is used to correct image alignment or for artistic effects. Image flipping includes horizontal and vertical mirroring, which reverses the image along the horizontal or vertical axis. Flipping is valuable for creating mirror images or altering the image's perspective.

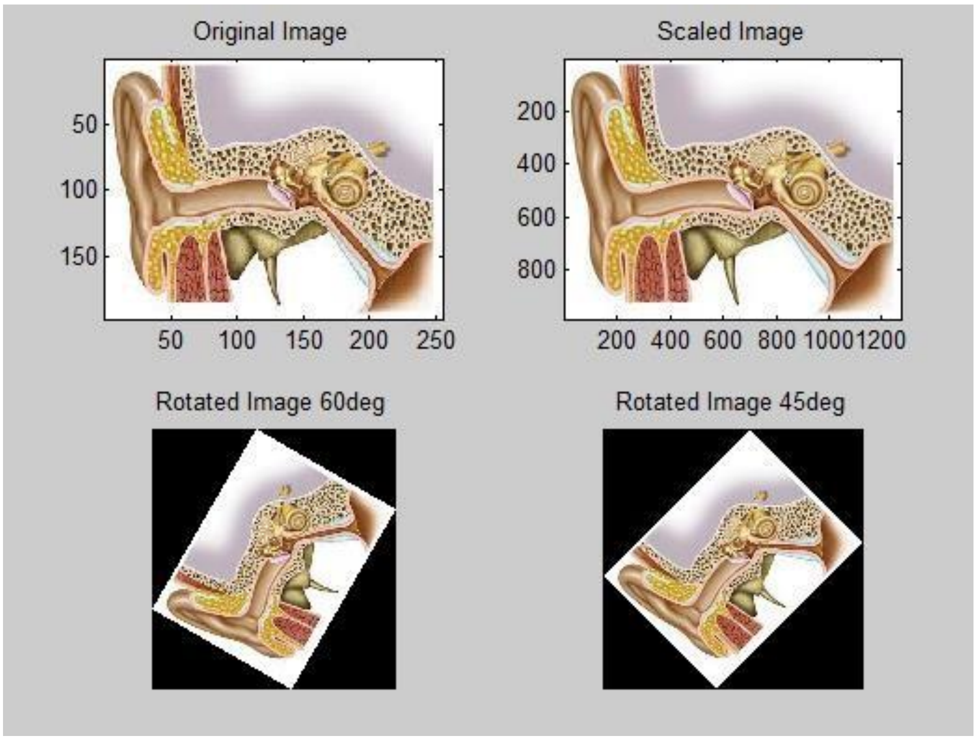
PROCEDURE:

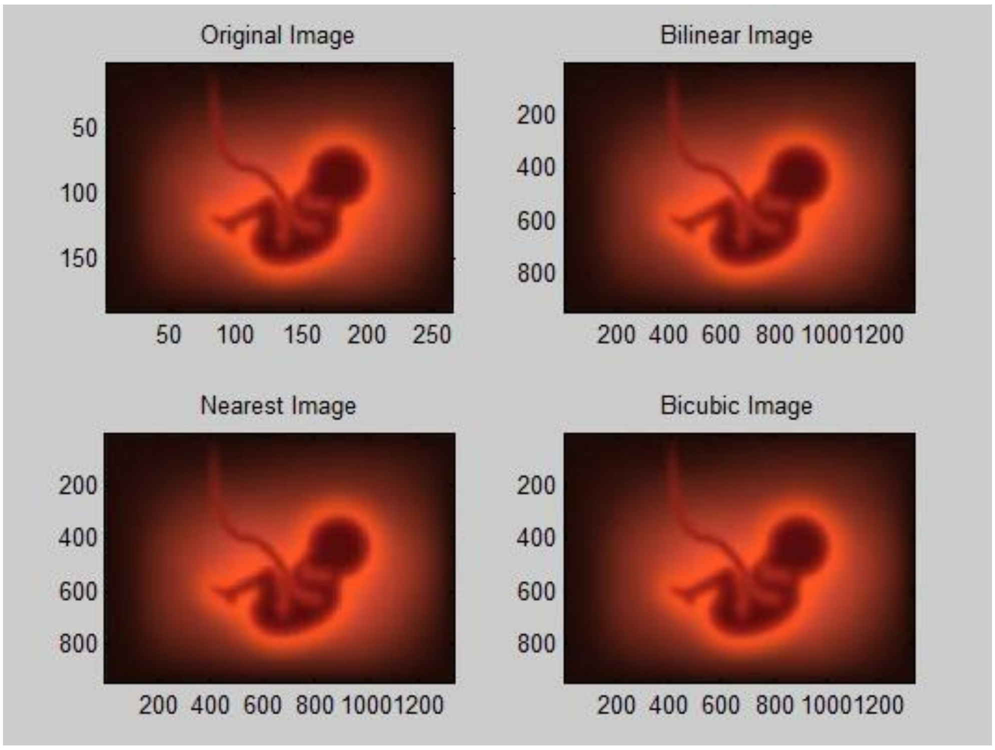
1. Load the digital image you want to transform using MATLAB.
2. Implement resizing by specifying the new dimensions or scaling factor. Interpolation methods, such as nearest-neighbor or bilinear, can be used to fill in pixel values during resizing.
3. Rotate the image by a specified angle, typically using transformation matrices. Be mindful of the interpolation method to maintain image quality during rotation.
4. Perform horizontal or vertical flipping to mirror the image. This involves rearranging pixel values along the specified axis.
5. Display the original image and the transformed image to visually compare the effects of the applied transformations.

PROGRAM:

```
%Scaling & Rotation
 % Scaling (Resize)
 I=imread('earcell.jpg');
 subplot(2,2,1); subimage(I); title('Original Image');
 s=input('Enter Scaling Factor');
 j=imresize(I,s);
 subplot(2,2,2); subimage(j); title('Scaled Image');
 % Rotation
 K=imrotate(j,60);
 subplot(2,2,3); imshow(K); title('Rotated Image 60deg');
 R=imrotate(j,45);
 subplot(2,2,4); imshow(R); title('Rotated Image 45deg');
 %Display the color image and its Resized images by different methods
 %Display the color image
 I=imread('embryo.jpg'); figure,
 subplot(2,2,1);
 subimage(I);
 title('Original Image');
 %Display Resized image by Bilinear method
 B=imresize(I,5);
 subplot(2,2,2); subimage(B);
 title('Bilinear Image');
 %Display Resized image by Nearest method
 C=imresize(I,5,'nearest');
 subplot(2,2,3); subimage(C);
 title('Nearest Image');
 %Display Resized image by Bicubic method
 D=imresize(I,5,'Bicubic');
subplot(2,2,4); subimage(D);
title('Bicubic Image');
```

OUTPUT:





RESULT:

REVIEW QUESTIONS:

1. What is the purpose of image resizing, and how does it affect the image's dimensions?
2. Explain the role of interpolation methods in image resizing, and provide an example of a scenario where resizing is useful.
3. How can you ensure that image quality is maintained when rotating an image in MATLAB?
4. Give an example of an application where image flipping is beneficial.
5. What factors should be considered when assessing the quality of a transformed image?

Ex. No.	**HISTOGRAM PROCESSING AND BASIC THRESHOLDING FUNCTIONS**	**Date**

AIM:

The aim of this project is to perform histogram processing and basic thresholding functions on digital images to enhance or segment regions of interest based on their pixel intensity values.

SOFTWARE REQUIRED:

MATLAB 2013b

THEORY:

A histogram is a graphical representation of the distribution of pixel intensity values in an image. It provides insights into the image's contrast, brightness, and distribution of features. Histogram processing refers to operations that modify the pixel intensity distribution in an image. Common processes include contrast stretching, histogram equalization, and histogram matching. Thresholding is a technique used to segment an image into foreground and background regions based on a specific threshold value. Pixels with intensity values above the threshold are classified as foreground, while those below are considered background.

PROCEDURE:

1. Load the digital image you want to process using MATLAB.
2. Compute and display the histogram of the image, representing the frequency of each intensity level.
3. Implement contrast stretching by specifying a lower and upper bound for the pixel intensity values. This expands the range of pixel values, enhancing image contrast.
4. Perform histogram equalization to redistribute pixel intensities across the entire range, which improves the overall image contrast.
5. If needed, match the image's histogram to a predefined histogram to make it similar to a reference image's histogram.
6. Apply a thresholding operation by specifying a threshold value. Pixels with values above this threshold are set to a predefined value (e.g., 255 for white), while those below are set to another predefined value (e.g., 0 for black).
7. Display the original image, histogram, and the processed image after histogram processing and thresholding.

PROGRAM:

%Creating histogram

```
clc;
clear
all;
close
all;
a = rgb2gray(imread('image.jpg'));
[m, n] = size(a);
no = 0:255;
count = 0;
for z=1:256
for i=1:m
for j=1:n
if  a(i, j) == z-1;
count = count + 1;
end
end
end
t(z) = count;
count = 0;
end
figure;
imshow(a)
title('Original image')
figure;
stem(no, t)
grid on;
xlabel('Intesity levels')
ylabel('No. of pixels')
title('Histogram of the image')
```

%Histogram equalization

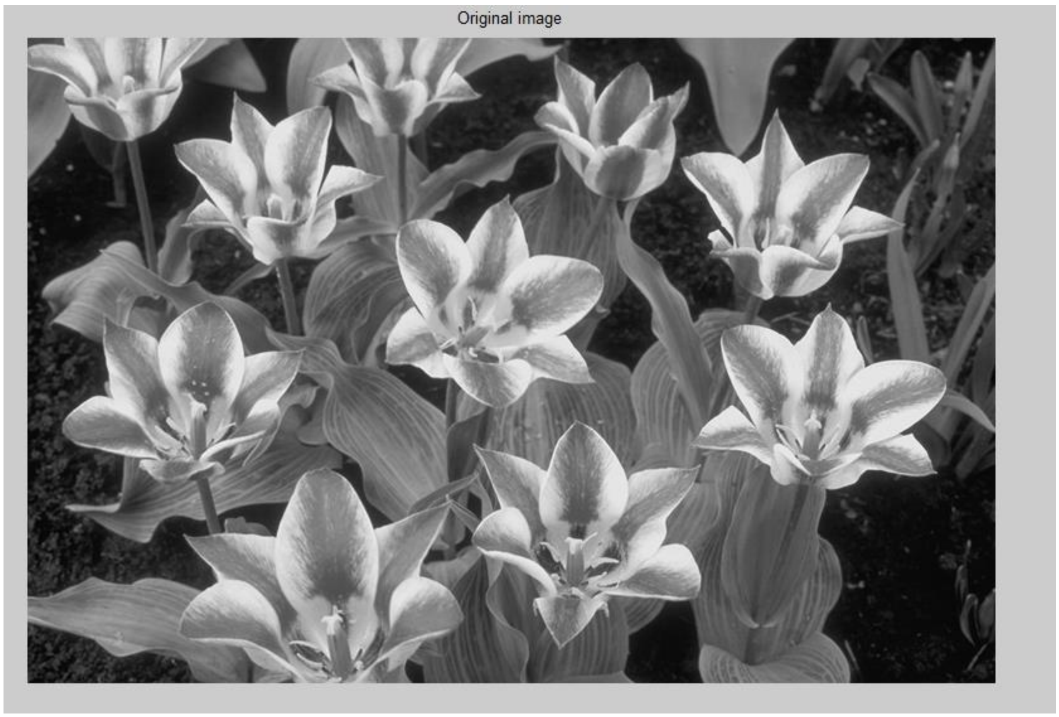
```
clc
clear
all
close
all
a = rgb2gray(imread('image.jpg'));
[m, n] = size(a);
no_of_pixels = m*n;
figure;
imshow(a)
title('Original image')
h_im = uint8(zeros(m, n));
count = zeros(256, 1);
probf = zeros(256, 1);
```

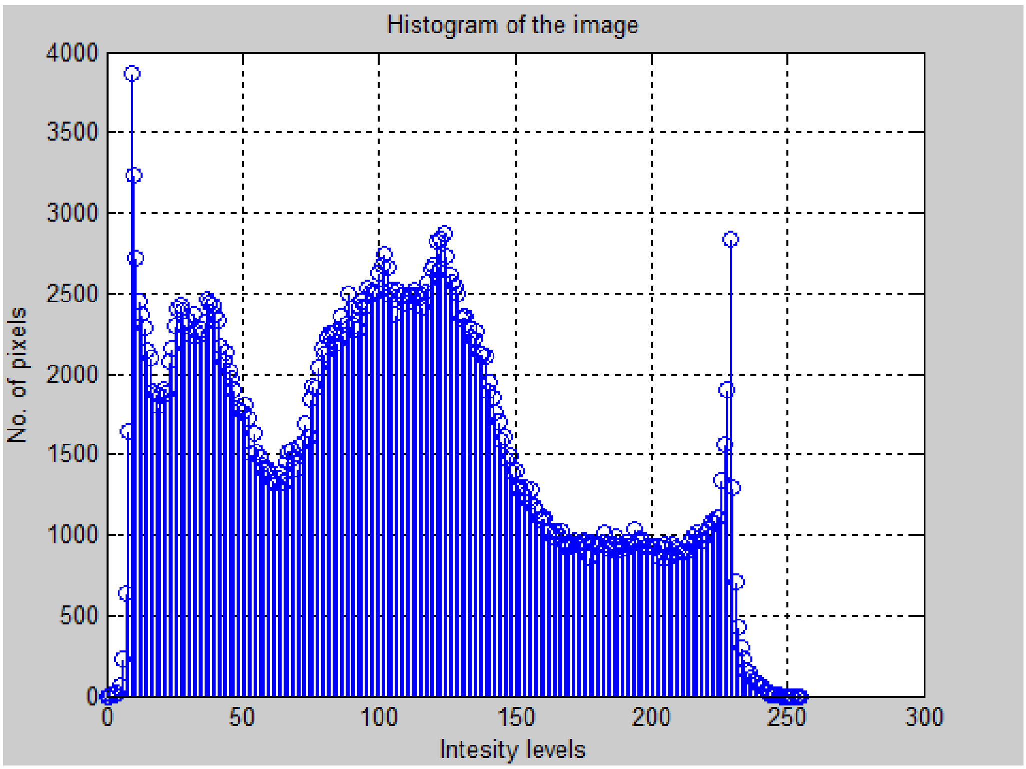
```
probc = zeros(256, 1);
cdf = zeros(256, 1);
output = zeros(1, 256);
no = 1:256;
for i=1:m
for j=1:n
value = a(i, j);
count(value + 1) = count(value + 1) + 1;
probf(value + 1) = count(value +1/no_of_pixels;
end
end
figu
re;
stem(no, probf)
title('Probability distribution function')sum = 0;
b = 255;
for i=1:size(probf)
        sum = sum + count(i);
        cdf(i) = sum;
        probc(i) = cdf(i)/no_of_pixels;
        output(i) = round(probc(i) * b);
end
figu
re;
stem(no, output)
title('Rounded off CDF*(L-1)')
for i=1:m
for j=1:n
  h_im(i, j) = output(a(i, j) + 1);
end
end
figure; imshow(h_im)
title('Histogram equalization')
```

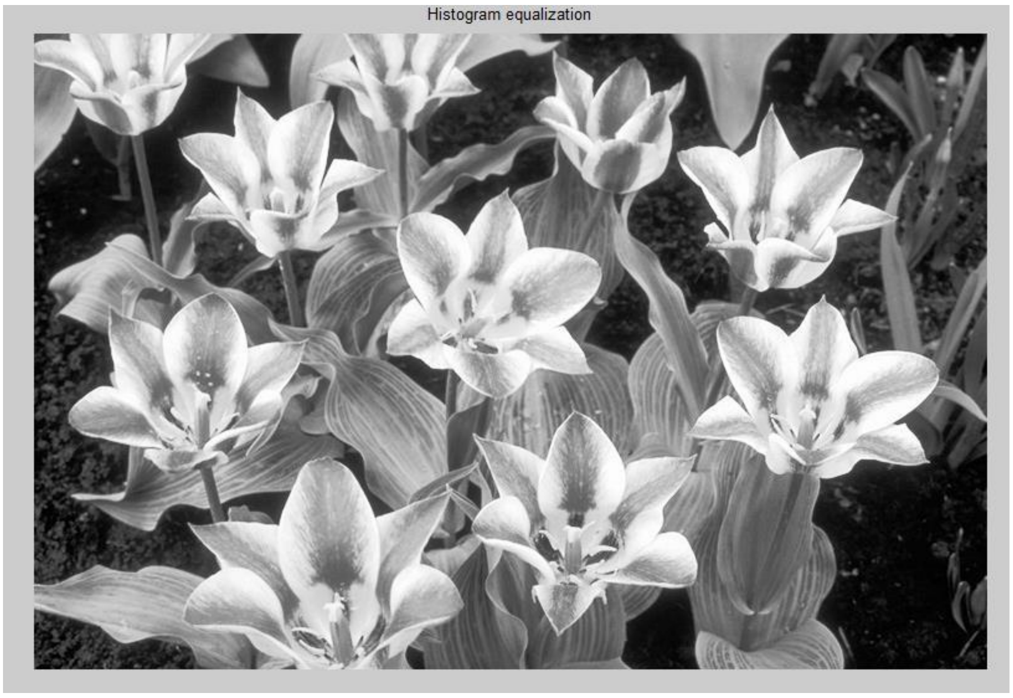
% Thresholding the original image using Otsu's method

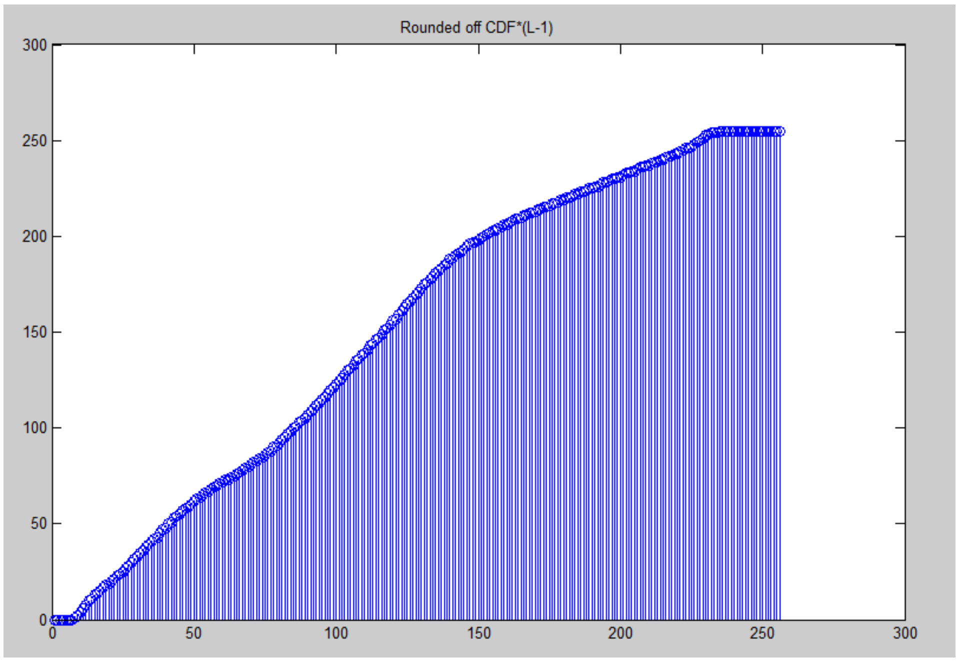
```
a = rgb2gray(imread('image.jpg'));
level = graythresh(a);
B = imbinarize(a, level);
subplot(1, 2, 1) imshow(uint8(a))
subplot(1, 2, 2) imshow(B)
```

OUTPUT:


Original image


Histogram of the image
No. of pixels
Intesity levels
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300





RESULT:

REVIEW QUESTIONS:

1. What is a histogram in the context of digital image processing, and why is it useful for analyzing images?
2. Explain the purpose of contrast stretching and histogram equalization in image enhancement.

3. What is the fundamental concept of thresholding, and how is it used to segment regions in an image?
4. How can you determine an appropriate threshold value for thresholding an image?
5. What are the key factors to consider when assessing the quality of an image after histogram processing and thresholding?

Ex. No.	COMPUTATION OF MEAN, STANDARD DEVIATION, CORRELATION COEFFICIENT OF THE GIVEN IMAGE	Date

AIM:
The aim of this experiment is to compute the Mean, Standard Deviation, and Correlation Coefficient of a given image.

SOFTWARE REQUIRED:
MATLAB 2013b

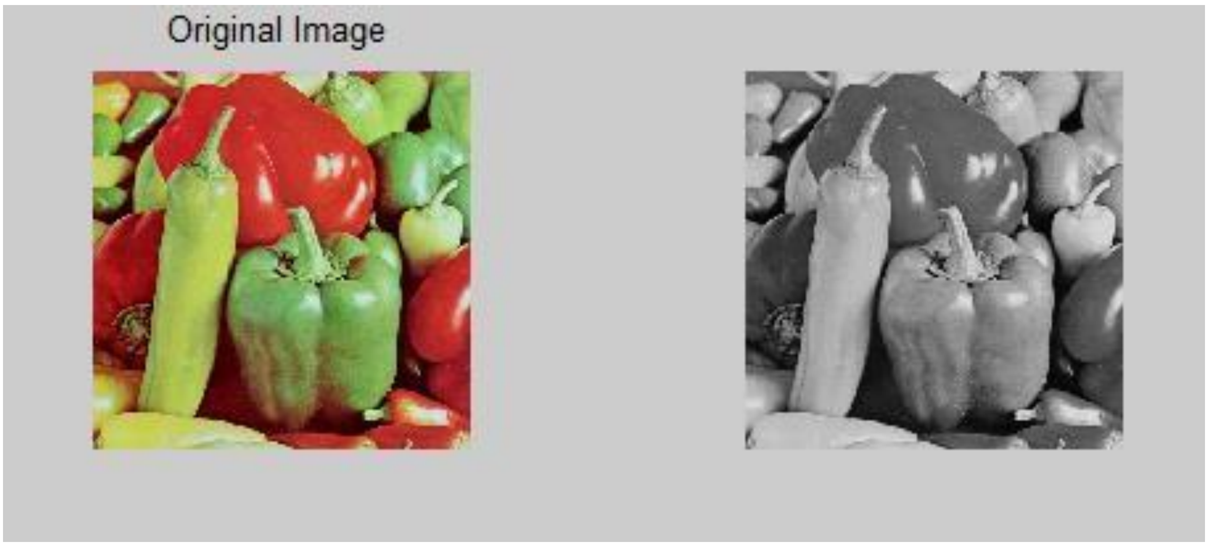
THEORY:
The mean (average) of an image represents the central tendency of pixel intensities in the image. It is calculated by summing up all the pixel values and dividing by the total number of pixels.
Mean (μ) = Σ [I(x, y)] / N
where I(x, y) represents the intensity of the pixel at position (x, y), and N is the total number of pixels in the image. The standard deviation measures the amount of variation or dispersion in pixel intensities. It provides information about the image's contrast and the distribution of pixel values. Standard Deviation (σ) = $\sqrt{}[\Sigma [(I(x, y) - \mu)^2] / N]$
where I(x, y) represents the pixel intensity, μ is the mean, and N is the total number of pixels in the image.
The correlation coefficient measures the degree of linear relationship between two images. It helps to understand how similar or dissimilar two images are. A high correlation coefficient indicates strong positive correlation, while a low coefficient suggests little to no correlation.
Correlation Coefficient (ρ) = $\Sigma [(I_1(x, y) - \mu_1) * (I_2(x, y) - \mu_2)] / [\sqrt{}\Sigma(I_1(x, y) - \mu_1)^2 * \sqrt{}\Sigma(I_2(x, y) - \mu_2)^2]$
where $I_1(x, y)$ and $I_2(x, y)$ are the pixel intensities in two images, μ_1 and μ_2 are the means of the respective images, and the summation is done over all pixels.

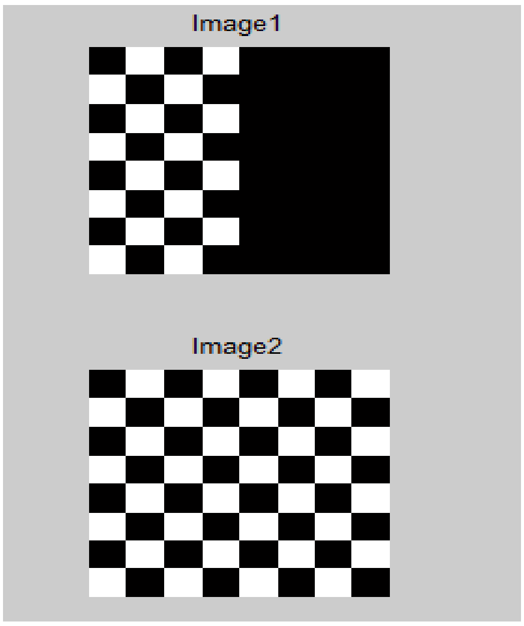
PROCEDURE:
1. Load the given image into your image processing environment MATLAB.
2. Calculate the Mean (μ) of the image using the formula mentioned above. This can be done by iterating through all pixels and accumulating their values.
3. Compute the Standard Deviation (σ) of the image using the formula mentioned above. Again, iterate through all pixels to calculate the necessary values.
4. If you need to compute the Correlation Coefficient between two images, repeat steps 2 and 3 for both images. Ensure that the images are of the same size and resolution.
5. Compute the Correlation Coefficient (ρ) using the formula for correlation mentioned above.

PROGRAM:

```
i=imread('cancercell.jpg');
subplot(2,2,1);
imshow(i);
title('Original Image');
g=rgb2gray(i);
subplot(2,2,2); imshow(g);
title('Gray Image');
c=imcrop(g);
subplot(2,2,3); imshow(c);
title('Cropped Image');
m=mean2(c);disp('m'); disp(m);
s=std2(c); disp('s'); disp(s);
figure, k=(checkerboard>0.8);
subplot(2,1,1); imshow(k);
title('Image1');
k1=(checkerboard>0.5);
subplot(2,1,2); imshow(k1);
title('Image2');
r=corr2(k,k1);
disp('r');disp(r);
```

OUTPUT:





RESULT:

REVIEW QUESTIONS:

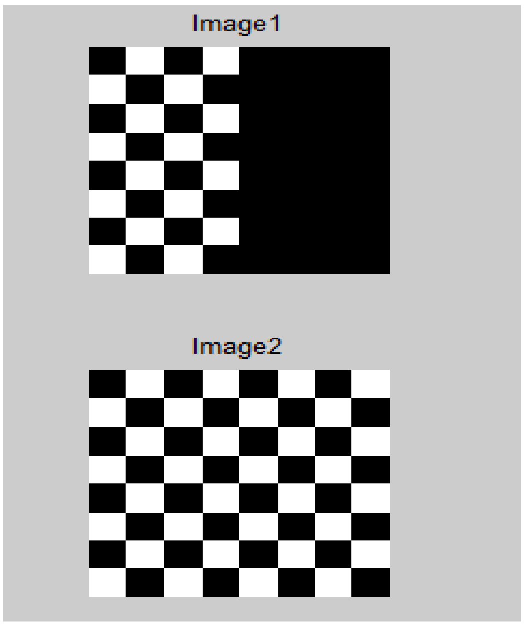
1. How is the mean computed for pixel values in an image and what does it represent?
2. What does the standard deviation reveal about an image's pixel values, and how is it calculated?
3. What is the significance of the correlation coefficient in image analysis, and how is it determined for two images?
4. Describe the concept of correlation coefficient as it pertains to image analysis. How is the correlation coefficient calculated for two images, and what does it indicate about their similarity or dissimilarity?
5. How can mean and standard deviation be used to assess the overall brightness and contrast of an image? Provide a step-by-step explanation with a practical example.

Ex. No.	IMPLEMENTATION OF IMAGE ENHANCEMENT-SPATIAL FILTERING	Date

AIM:
The aim of this project is to implement image enhancement using spatial filtering techniques to improve the visual quality of digital images.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Image enhancement through spatial filtering is a fundamental image processing technique. It involves the use of convolution operations with specific filter kernels to modify pixel values in an image. A basic spatial filter, such as an averaging filter, is used to reduce noise and enhance



RESULT:

REVIEW QUESTIONS:

1. How is the mean computed for pixel values in an image and what does it represent?
2. What does the standard deviation reveal about an image's pixel values, and how is it calculated?
3. What is the significance of the correlation coefficient in image analysis, and how is it determined for two images?
4. Describe the concept of correlation coefficient as it pertains to image analysis. How is the correlation coefficient calculated for two images, and what does it indicate about their similarity or dissimilarity?
5. How can mean and standard deviation be used to assess the overall brightness and contrast of an image? Provide a step-by-step explanation with a practical example.

Ex. No.	IMPLEMENTATION OF IMAGE ENHANCEMENT-SPATIAL FILTERING	Date

AIM:
The aim of this project is to implement image enhancement using spatial filtering techniques to improve the visual quality of digital images.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Image enhancement through spatial filtering is a fundamental image processing technique. It involves the use of convolution operations with specific filter kernels to modify pixel values in an image. A basic spatial filter, such as an averaging filter, is used to reduce noise and enhance

the overall appearance of the image. The process involves convolving the image with the filter kernel, which replaces each pixel's value with a weighted average of its neighboring pixels. This can help to improve image quality, reduce artifacts, and enhance specific features.

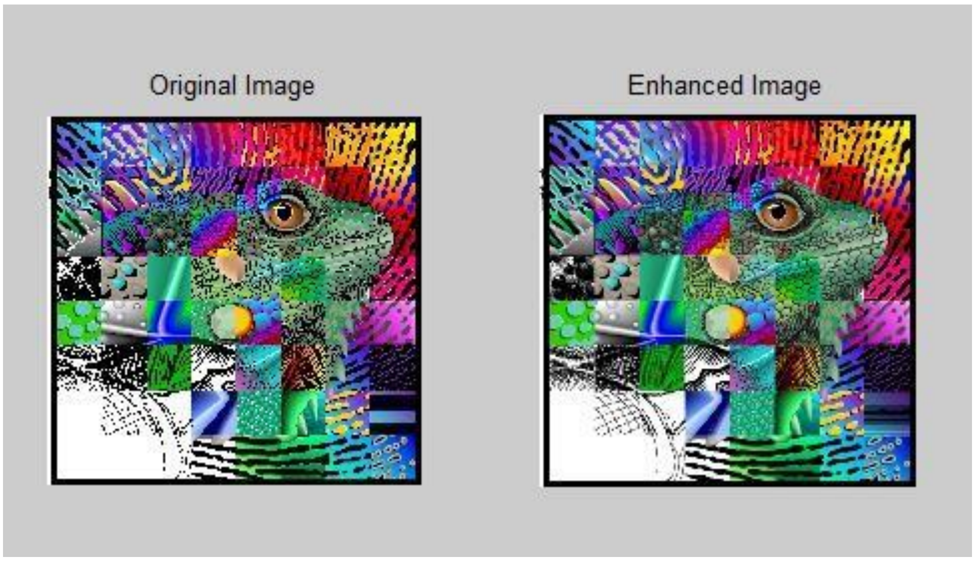
PROCEDURE:

1. Load the input image on which image enhancement will be performed.
2. Choose a suitable filter kernel based on the specific image enhancement goals. Common choices include averaging filters, Gaussian filters, or edge-enhancing filters.
3. Convolve the input image with the selected filter kernel using the `conv2` function in MATLAB.
4. Adjust the filter size and values to control the extent of enhancement.
5. Convert the resulting image to an appropriate data type (e.g., `uint8`) to ensure it falls within the valid pixel value range (0-255).
6. Display both the original and enhanced images using MATLAB's `imshow` function.
7. Save the enhanced image to a file for further analysis or use.

PROGRAM:

```
% Load the image
originalImage = imread('image2.jpg'); % the image filename

% Convert the image to double precision for better processing
originalImage = im2double(originalImage);

% Define the filter kernel (averaging filter)
filterSize = 3; % Size of the filter kernel (e.g., 3x3)
filter = ones(filterSize) / filterSize^2;

% Apply the filter using convolution
enhancedImage = imfilter(originalImage, filter, 'replicate'); % 'replicate' padding to handle
image borders

% Display the original and enhanced images
subplot(1, 2, 1);
imshow(originalImage);
title('Original Image');
subplot(1, 2, 2);
imshow(enhancedImage);
title('Enhanced Image');

% Save the enhanced image if needed
% imwrite(enhancedImage, 'enhanced_image.jpg');
```

OUTPUT:



RESULT:

REVIEW QUESTIONS:
1. What is the primary goal of image enhancement through spatial filtering?
2. Can you explain the concept of a filter kernel and its role in image enhancement using spatial filtering?
3. What is convolution, and how does it apply to image enhancement in this context?
4. What are some common spatial filter types used for image enhancement, and how do they differ in their effects on an image?
5. What considerations should be taken into account when selecting a filter kernel and its size for image enhancement, and how does this choice affect the output?

Ex. No.	IMPLEMENTATION OF IMAGE ENHANCEMENT- FILTERING IN FREQUENCY DOMAIN	Date

AIM:
The aim of this project is to implement image enhancement through filtering in the frequency domain to improve the visual quality and extract specific features from digital images.
SOFTWARE REQUIRED:
MATLAB 2013b
THEORY:
Image enhancement through filtering in the frequency domain involves transforming an image from the spatial domain to the frequency domain using techniques like the Fast Fourier Transform (FFT). Once in the frequency domain, filtering operations are applied to enhance or suppress specific frequency components. The filtered image is then transformed back to the spatial domain using the Inverse Fast Fourier Transform (IFFT). This process can be used to remove noise, sharpen details, and improve image quality.

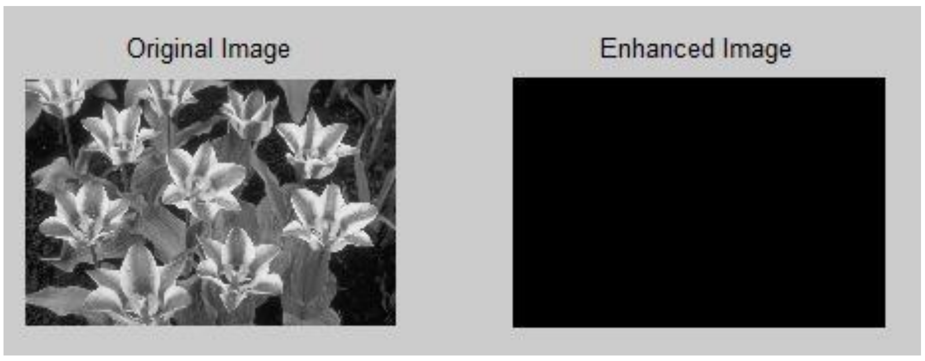
PROCEDURE:

1. Load the input image for image enhancement.
2. Apply the Fast Fourier Transform (FFT) to convert the image from the spatial domain to the frequency domain.
3. Choose an appropriate filter (e.g., high-pass, low-pass, or band-pass filter) to perform the desired enhancement operation.
4. Apply the selected filter to the transformed image in the frequency domain. This involves point-wise multiplication or convolution with the filter.
5. Use the Inverse Fast Fourier Transform (IFFT) to transform the filtered image back to the spatial domain.
6. Adjust the output image to ensure it falls within the valid pixel value range (0-255).
7. Display both the original and enhanced images.

PROGRAM:

```
clc
clear all
close all
a = imresize(imread('image.jpg’), [256, 256]);
m, n] = size(a);
mask = zeros(m, n);
for i=113:143
        for j=113:143
                mask(i, j) = 1;
        end
end
b = fftshift(fft2(a));
d = b .* mask;
e = abs(ifft2(d));
subplot(2, 2, 1)
imshow(a)
title('Orginal image')
subplot(2, 2, 2) imshow(uint8(e))
title('Low-pass filter filtered image')
subplot(2, 2, 3)
imshow(mask)
title('Low-pass filter mask')
```

OUTPUT:



RESULT:

REVIEW QUESTIONS:

1. What is the fundamental difference between image enhancement in the spatial domain and the frequency domain?
2. How does the Fast Fourier Transform (FFT) enable image enhancement in the frequency domain?
3. What are some common filters used for image enhancement in the frequency domain, and how do they affect image features?
4. Can you explain the concept of convolution or point-wise multiplication in the context of frequency domain filtering for image enhancement?
5. What are the advantages of using frequency domain filtering for image enhancement, and in what scenarios is it particularly useful?

Ex. No.	IMAGE SEGMENTATION – EDGE DETECTION, LINE DETECTION AND POINT DETECTION	Date

AIM:

The aim of this project is to perform image segmentation, specifically focusing on edge detection, line detection, and point detection in digital images, enabling the extraction of important features and boundaries.

SOFTWARE REQUIRED:

MATLAB 2013b

THEORY:

Image segmentation is a crucial image processing task that aims to partition an image into meaningful regions. The techniques used in this project involve:

1. Edge Detection: This technique highlights significant transitions or boundaries in an image. It is commonly used to identify object boundaries or regions of interest.
2. Line Detection: Utilizing the Hough Transform, this technique identifies straight lines in an image, which is useful for applications such as object tracking and lane detection.
3. Point Detection: The Harris corner detection method identifies key points or corners in an image, providing critical feature points for various computer vision tasks.

PROCEDURE:

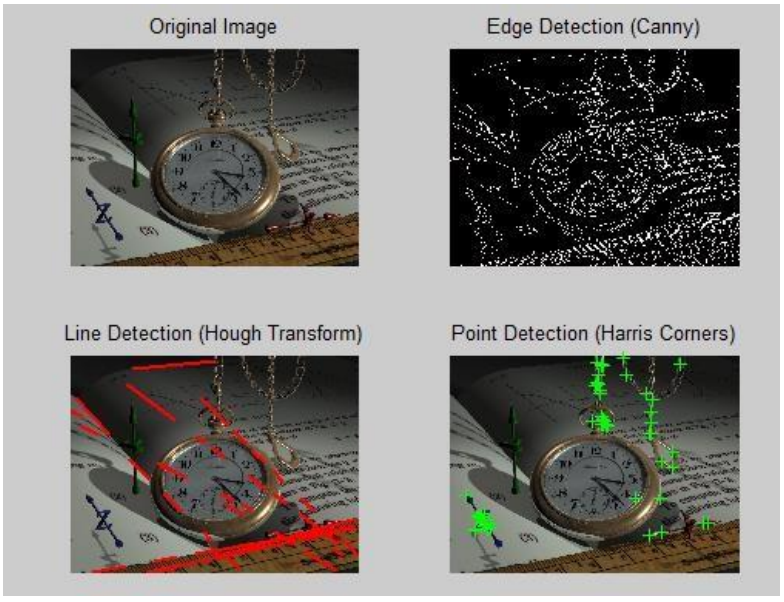
1. Load the input image onto which image segmentation techniques will be applied.
2. Apply edge detection methods, such as the Canny edge detector, to highlight image edges and transitions.
3. Employ the Hough Transform to identify straight lines in the image, displaying their positions and orientations.
4. Utilize the Harris corner detection technique to find key points or corners in the image.
5. Display the original image and the results of edge detection, line detection, and point detection.
6. Save the images resulting from each segmentation technique for further analysis or use.

PROGRAM:

```
% Load the image
inputImage = imread('your_image.jpg');
% Edge Detection using Canny
edgeImage = edge(rgb2gray(inputImage), 'Canny');
% Line Detection using Hough Transform
```

```
[H, theta, rho] = hough(edgeImage);
peaks = houghpeaks(H, 10);  % Adjust the number of peaks as needed
lines = houghlines(edgeImage, theta, rho, peaks);
% Point Detection using the Harris corner detector
corners = detectHarrisFeatures(rgb2gray(inputImage));
% Display the results
figure;
subplot(2, 2, 1);
imshow(inputImage);
title('Original Image');
subplot(2, 2, 2);
imshow(edgeImage);
title('Edge Detection (Canny)');
subplot(2, 2, 3);
imshow(inputImage);
hold on;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1), xy(:,2), 'LineWidth', 2, 'Color', 'red');
end
title('Line Detection (Hough Transform)');
subplot(2, 2, 4);
imshow(inputImage);
hold on;
plot(corners.selectStrongest(50));
title('Point Detection (Harris Corners)');
% Save the images if needed
imwrite(edgeImage, 'edge_detection.jpg');
imwrite(inputImage, 'line_detection.jpg');
imwrite(inputImage, 'point_detection.jpg');
```

OUTPUT:


Original Image
Edge Detection (Canny)
Line Detection (Hough Transform)
Point Detection (Harris Corners)

RESULT:

REVIEW QUESTIONS:

1. What is the significance of edge detection in image segmentation?
2. How does the Hough Transform assist in line detection in image analysis?
3. What are the key objectives of point detection using methods like the Harris corner detector?
4. In what scenarios might image segmentation techniques like edge detection, line detection, and point detection be particularly valuable?
5. Can you discuss potential challenges or limitations associated with these image segmentation techniques?

Ex. No.	IMPLEMENTATION OF REGION BASED SEGMENTATION	Date

AIM:

The aim is to partition an input image into meaningful regions or segments based on color, intensity, or texture characteristics, thus improving image analysis and object recognition.

SOFTWARE REQUIRED:

MATLAB 2013b

THEORY:

Region-based segmentation is a computer vision technique that groups pixels or image regions into meaningful segments based on their similarity in terms of color, intensity, or texture. This segmentation technique involves the following key steps:

1. Preprocessing: The input image is preprocessed to enhance its quality, reduce noise, and normalize intensity values.
2. Feature Extraction: Relevant features, such as color histograms, texture features, or intensity gradients, are computed for each pixel or region in the image.
3. Region Growing or Split-and-Merge: The segmentation algorithm selects seed points and iteratively grows or splits regions based on feature similarity. Region growing starts with seed pixels and expands by including neighboring pixels that meet certain similarity criteria. Split-and-merge, on the other hand, recursively divides large regions into smaller ones if dissimilarity criteria are met.
4. Region Merging: After regions have been created, regions that are too small or similar to their neighbors are merged.
5. Postprocessing: Additional postprocessing steps can be applied to refine the segmentation results, such as removing small noise regions or smoothing region boundaries.

PROCEDURE:

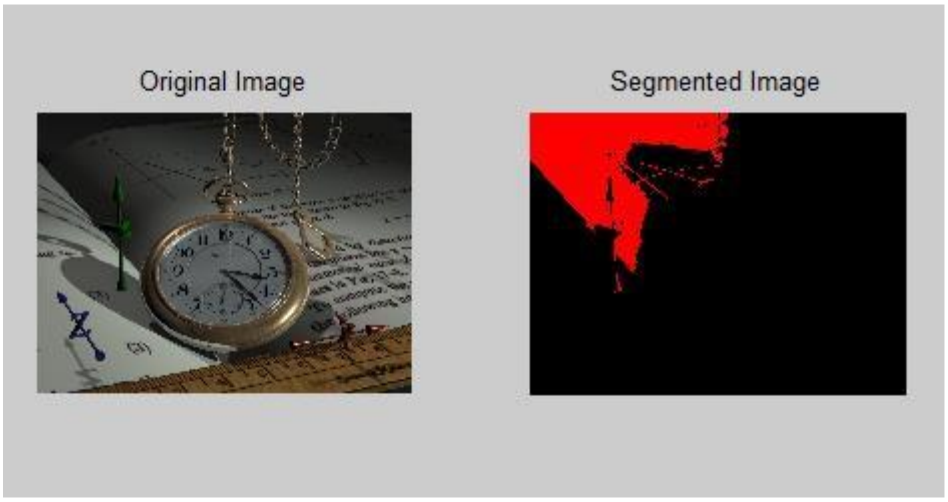
1. Import the input image.
2. Preprocess the image (e.g., noise reduction, intensity normalization).
3. Compute relevant image features (e.g., color histograms, texture features).
4. Initialize seed points or regions.
5. Implement the region growing or split-and-merge algorithm.
6. Apply region merging to refine the segmentation.
7. Perform postprocessing as needed.
8. Visualize and analyze the segmented image.

9. Optimize parameters and iterate for better results.
10. Save or export the segmented image.

PROGRAM:

```
% Read the input image
inputImage = imread('clock.jpg');
% Define a seed point (you can choose this point manually)
seedPoint = [100, 100]; % [row, column]
% Set the intensity similarity threshold
intensityThreshold = 20;
% Initialize the segmented image
segmentedImage = zeros(size(inputImage));
% Create a stack for region growing
stack = [];
stack = [stack; seedPoint];
% Define the region mean and region size
regionMean = double(inputImage(seedPoint(1), seedPoint(2)));
regionSize = 1;
% Define connectivity for 8-connected neighbors
connectivity = [-1, -1; -1, 0; -1, 1; 0, -1; 0, 1; 1, -1; 1, 0; 1, 1];
% Region growing loop
while ~isempty(stack)
   currentPoint = stack(1, :);
   stack(1, :) = [];
for i = 1:8
      neighbor = currentPoint + connectivity(i, :);
      if neighbor(1) > 0 && neighbor(1) <= size(inputImage, 1) && neighbor(2) > 0 &&
neighbor(2) <= size(inputImage, 2)
         if segmentedImage(neighbor(1), neighbor(2)) == 0 &&
abs(double(inputImage(neighbor(1), neighbor(2))) - regionMean) <= intensityThreshold
            regionSize = regionSize + 1;
            regionMean = (regionMean * (regionSize - 1) + double(inputImage(neighbor(1),
neighbor(2)))) / regionSize;
            stack = [stack; neighbor];
            segmentedImage(neighbor(1), neighbor(2)) = 1;
         end
      end
   end
end
% Display the segmented image
figure;
subplot(1, 2, 1);
imshow(inputImage);
title('Original Image');
subplot(1, 2, 2);
imshow(segmentedImage, []);
title('Segmented Image');
% You can save the segmented image if needed
imwrite(segmentedImage, 'segmented_image.jpg');
```

OUTPUT:



RESULT:

REVIEW QUESTIONS:
1. What is the primary objective of region-based image segmentation?
2. Describe the key steps involved in region-based segmentation.
3. What are some preprocessing techniques that can be applied to improve segmentation results?
4. How does region growing differ from split-and-merge in image segmentation?
5. What postprocessing steps can be employed to refine the segmented image?

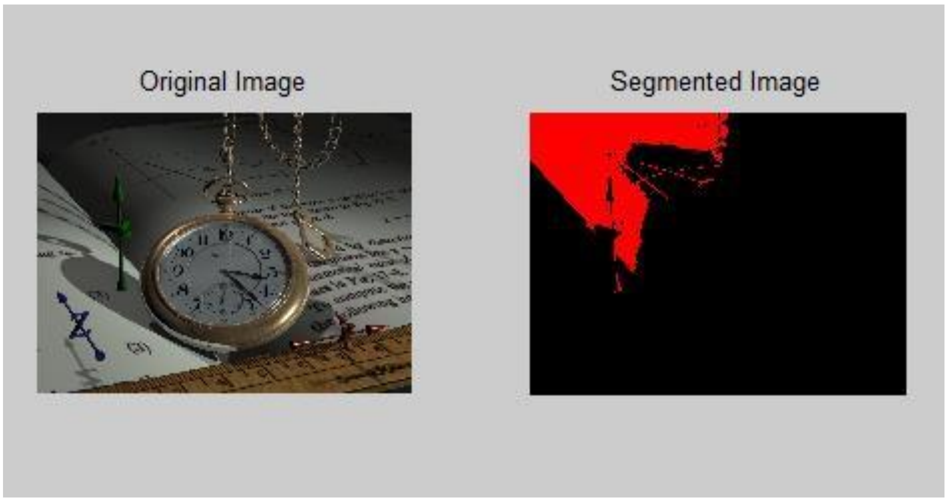
Ex. No.	**BASIC MORPHOLOGICAL OPERATIONS**	**Date**

AIM:
The aim is to understand and implement basic morphological operations, including dilation, erosion, opening, and closing, using digital images.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Morphological operations are image processing techniques based on the shape and structure of objects within an image. They are often applied to binary or grayscale images and rely on a structuring element (also known as a kernel) to modify the pixels in the image.
Dilation is used to enlarge the boundaries of objects in a binary image. It involves sliding the structuring element over the image and replacing the center pixel with the maximum value found in the neighborhood defined by the structuring element.
Erosion is used to shrink the boundaries of objects in a binary image. It involves sliding the structuring element over the image and replacing the center pixel with the minimum value found in the neighborhood defined by the structuring element.
Opening is a combination of erosion followed by dilation. It is used to remove small noise in binary images or separate objects that are close to each other.

OUTPUT:



RESULT:

REVIEW QUESTIONS:

1. What is the primary objective of region-based image segmentation?
2. Describe the key steps involved in region-based segmentation.
3. What are some preprocessing techniques that can be applied to improve segmentation results?
4. How does region growing differ from split-and-merge in image segmentation?
5. What postprocessing steps can be employed to refine the segmented image?

Ex. No.	BASIC MORPHOLOGICAL OPERATIONS	Date

AIM:
The aim is to understand and implement basic morphological operations, including dilation, erosion, opening, and closing, using digital images.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
Morphological operations are image processing techniques based on the shape and structure of objects within an image. They are often applied to binary or grayscale images and rely on a structuring element (also known as a kernel) to modify the pixels in the image.
Dilation is used to enlarge the boundaries of objects in a binary image. It involves sliding the structuring element over the image and replacing the center pixel with the maximum value found in the neighborhood defined by the structuring element.
Erosion is used to shrink the boundaries of objects in a binary image. It involves sliding the structuring element over the image and replacing the center pixel with the minimum value found in the neighborhood defined by the structuring element.
Opening is a combination of erosion followed by dilation. It is used to remove small noise in binary images or separate objects that are close to each other.

Closing is a combination of dilation followed by erosion. It is used to close small gaps in binary images or to connect objects that are almost touching.

PROCEDURE:

1. Import the digital image onto which you want to apply morphological operations.
2. Convert the image to a binary format if it's not already binary. This is often done by thresholding.
3. Define a structuring element (a small matrix) that specifies the neighborhood for the morphological operation. The size and shape of the structuring element depend on the specific operation and the desired effect.
4. Apply dilation to the binary image using the chosen structuring element.
5. Update the pixel values according to the dilation operation.
6. Apply erosion to the binary image using the chosen structuring element.
7. Update the pixel values according to the erosion operation.
8. Apply an opening operation, which consists of an erosion followed by dilation. This is used for noise removal and object separation.
9. Apply a closing operation, which consists of a dilation followed by erosion. This is used for gap-filling and object connection.
10. Visualize or save the processed image.
11. Experiment with different structuring elements and parameters to achieve the desired image enhancement or feature extraction.
12. Summarize the results and the impact of the morphological operations on the image.

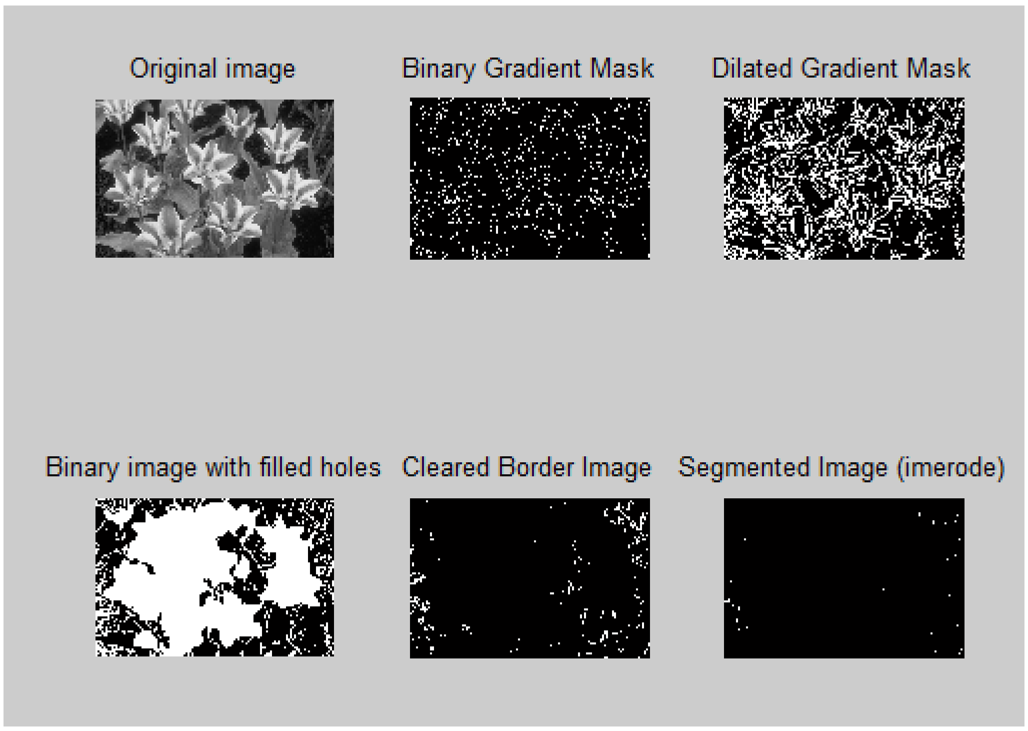
PROGRAM:

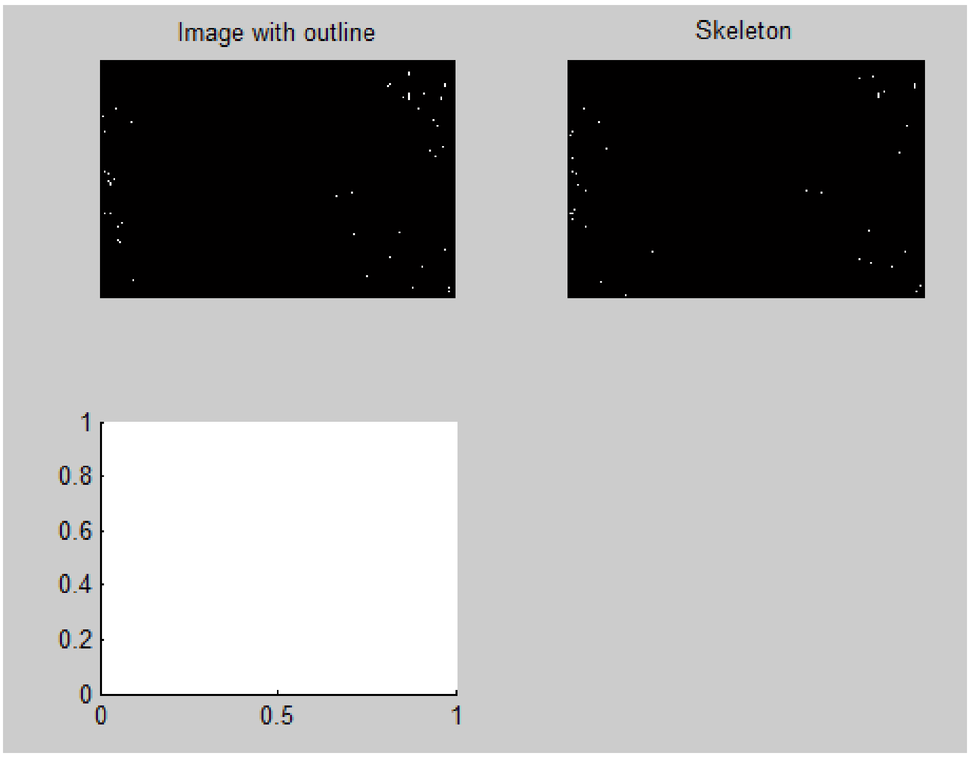
```
clc
clear all
close all
I = rgb2gray(imread('image.jpg'));
subplot(2, 3, 1)
imshow(I)
title('Original image')
%Edge detection and binary image
[~, threshold] = edge(I, 'sobel');
fudgeFactor = 0.5;
BWs = edge(I, 'sobel', threshold *fudgeFactor);
%Display the resulting binary gradient mask
subplot(2, 3, 2)
imshow(BWs)
title('Binary Gradient Mask')
%imdilate
se90 = strel('line', 3, 90);
se0 = strel('line', 3, 0);
BWsdil = imdilate(BWs, [se90, se0]);
subplot(2, 3, 3)
imshow(BWsdil)
title('Dilated Gradient Mask')
```

```
BWdfill = imfill(BWsdil, 'holes');
subplot(2, 3, 4)
imshow(BWdfill)
title('Binary image with filled holes')
BWnobord = imclearborder(BWdfill, 4);
subplot(2, 3, 5)
imshow(BWnobord)
title('Cleared Border Image')
seD = strel('diamond', 1);
BWfinal = imerode(BWnobord, seD);
BWfinal = imerode(BWfinal, seD);
subplot(2, 3, 6)
imshow(BWfinal)
title('Segmented Image (imerode)')
%Remove inner elements
BW2 = bwmorph(BWfinal, 'remove');
figure;
subplot(2, 2, 1)
imshow(BW2)
title('Image with outline')
%Skeleton image
BW3 = bwmorph(BWfinal, 'skel', Inf);
subplot(2, 2, 2)
imshow(BW3)
title('Skeleton')
subplot(2, 2, 3) imshow(labeloverlay(I,
BWfinal)) title('Mask over original image')
BWoutline = bwperim(BWfinal); Segout = I;
Segout(BWoutline) = 255;
subplot(2, 2, 4)
imshow(Segout)
title('Outlined original image')
%Opening and closing
originalBW = I; figure;
subplot(2, 2, 1)
imshow(originalBW)
title('Original')
se = strel('disk', 10);
closeBW = imclose(originalBW, se);
subplot(2, 2, 2)
imshow(closeBW, [])
title('imclose')
original = I;
subplot(2, 2, 3)
imshow(original)
title('Original')
se = strel('disk', 20);
afterOpening = imopen(original, se);
subplot(2, 2, 4)
```

```
imshow(afterOpening, [])
title('imopen')
```

OUTPUT:





RESULT:

REVIEW QUESTIONS:
1. What are morphological operations in image processing, and what is their primary purpose?
2. Explain the fundamental morphological operations: dilation and erosion. How do they work, and what are their typical applications?
3. What is the role of a structuring element in morphological operations? How does its size and shape affect the results?
4. Describe the differences between opening and closing operations. When would you use one over the other?
5. How can morphological operations be used for noise reduction or feature enhancement in images? Provide examples of scenarios where morphological operations are beneficial.

Ex. No.	IMPLEMENTATION OF IMAGE COMPRESSION TECHNIQUES	Date

AIM:
To perform JPEG compression using DCT to an image usingMATLAB.

SOFTWARE REQUIRED:
MATLAB 2013b

THEORY:
JPEG (Joint Photographic Experts Group) compression is a widely used image compression technique that employs the Discrete Cosine Transform (DCT) for reducing the size of digital images while maintaining reasonable image quality. JPEG compression begins with dividing the image into small blocks, typically 8x8 pixels. These blocks are then subjected to the DCT transformation, which converts spatial domain pixel values into frequency domain coefficients. After the DCT, quantization is applied to the frequency domain coefficients. Quantization reduces the precision of the coefficients by dividing them by a quantization matrix. The quantized coefficients are scanned in a zigzag pattern to group the high-frequency components together and make them more compressible. The zigzag-scanned coefficients are then run-length encoded, which replaces sequences of repeated values with a code that represents the value and the number of repetitions. This is a form of lossless compression. The encoded data is further compressed using Huffman encoding, which assigns shorter codes to more frequently occurring sequences and longer codes to less frequent sequences.

PROCEDURE:
1. Load the input image.
2. Divide the image into 8x8 pixel blocks.
3. Apply the DCT transformation to each block.
4. Quantize the DCT coefficients using a quantization matrix.
5. Perform zigzag scanning on the quantized coefficients.
6. Run-length encode the zigzag-scanned coefficients.
7. Apply Huffman encoding to further compress the data.
8. Save the compressed data and encoding tables for later decoding.
9. To decompress, reverse the process using Huffman decoding, run-length decoding, de-zigzag scanning, de-quantization, and inverse DCT transformation.
10. Display or save the decompressed image.

PROGRAM:

```
%nearer values are retained apart  from DC component
clc
clear
all
close
all
a = rgb2gray(imread('image.jpg'));
figure;
subplot(1, 2, 1)
imshow(a)
title('Original image')
[m, n] = size(a);
bs = 8;
Q = 20 * ones(bs);
Q(1, 1) = 10;
Rec_a = zeros(size(a));
for i=1:bs:(m-bs+1)
        for j=1:bs:(n-bs+1)
                block = a(i:i+bs-1, j:j+bs-1);
                block_D = dct2(block);
                block_Q = round(block_D./Q);
                d = block_Q.*Q;
                Rec_a(i:i+bs-1, j:j+bs-1) = idct2(d);
        end
end
subplot(1, 2, 2)
imshow(uint8(Rec_a))
title('Reconstructed image')
MSE = sum(sum((uint8(a)-uint8(Rec_a)).^2))/(m*n)
% set all components to zero except the DC component
clc
clear
all
close
all
a = rgb2gray(imread('D:\College\DIP\Lab\Lab8\image.jpg'));
figure;
subplot(1, 2, 1)
imshow(a)
title('Original image')
[m, n] = size(a);
bs = 8;
Q = 20 * ones(bs);
Q(1, 1) = 10;
Rec_a = zeros(size(a));
for i=1:bs:(m-bs+1)
        for j=1:bs:(n-bs+1)
                block = a(i:i+bs-1, j:j+bs-1);
                block_D = dct2(block);
                d = zeros(size(block_D));
                d(1, 1) = block_D(1, 1);
                Rec_a(i:i+bs-1, j:j+bs-1) = idct2(d);
        end
 end
subplot(1, 2, 2)
imshow(uint8(Rec_a))
title('Reconstructed image')
MSE = sum(sum((uint8(a)-uint8(Rec_a)).^2))/(m*n)
% specify how many elements are to be retained
clc
clear all
close all
a = rgb2gray(imread('D:\College\DIP\Lab\Lab8\image.jpg'));
k = input("Enter the number of values to be retained:");
figure; subplot(1,
2, 1)imshow(a)
title('Original image');
[m, n]=size(a);
bs = 8;
Rec_a = zeros(size(a));for
i=1:bs:(m-bs+1)
        for j=1:bs:(n-bs+1)
                block = a(i:i+bs-1, j:j+bs-1);
                block_D = dct2(block);
                d = zeros(size(block_D));
                d(1:k, 1:k) = block_D(1:k, 1:k);
                Rec_a(i:i+bs-1, j:j+bs-1) = idct2(d);
        end
end
subplot(1, 2, 2)
imshow(uint8(Rec_a))
title('Reconstructed image');
MSE = sum(sum((uint8(a)-uint8(Rec_a)).^2))/(m*n)
```

OUTPUT:

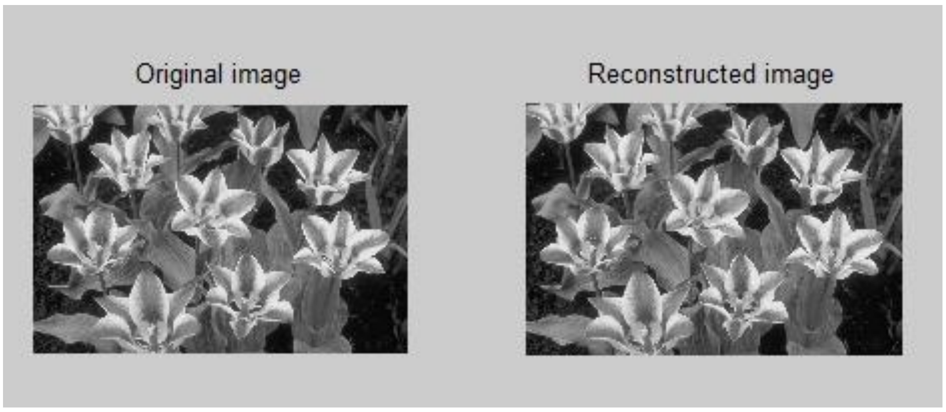


Fig 1.

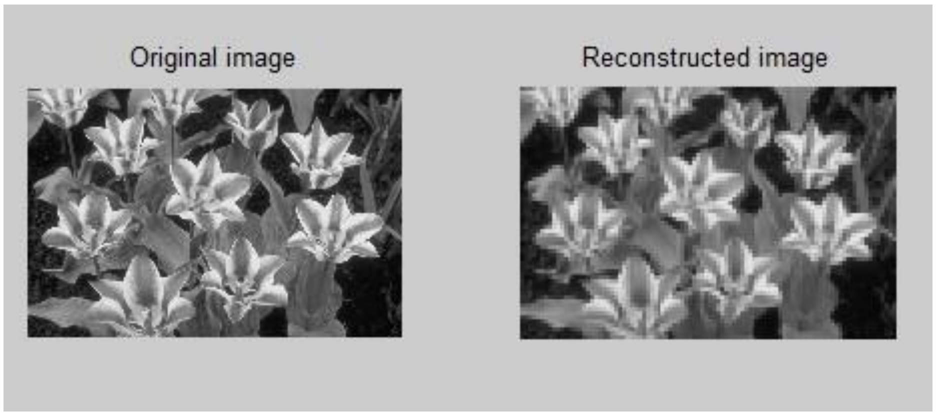


Fig.2.

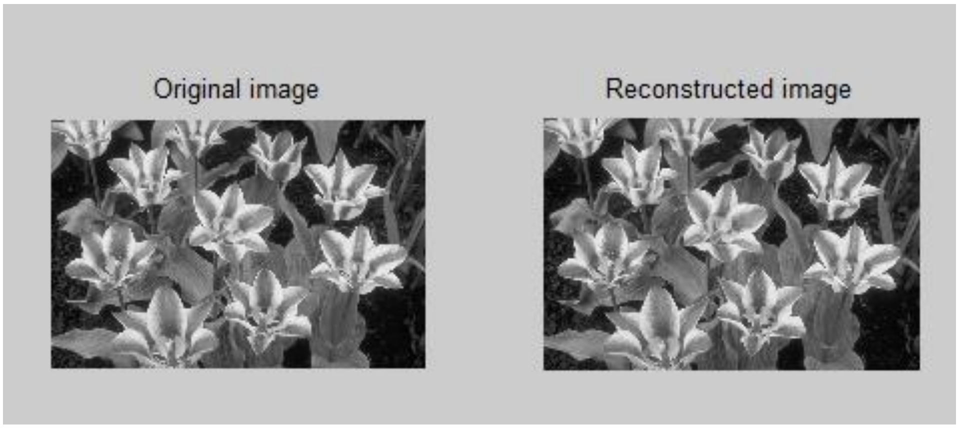


Fig.3.

```
Enter the number of values to be retained: 5
MSE =    4.9197
```

RESULT:

REVIEW QUESTIONS:

1. What is the fundamental purpose of JPEG compression, and how does it achieve this purpose?
2. How does the Discrete Cosine Transform (DCT) contribute to JPEG compression?
3. Explain quantization in JPEG compression. How does the choice of quantization matrix affect image quality?
4. Describe the significance of zigzag scanning in JPEG compression. How does it impact the encoding process?
5. Why are both Run-Length Encoding (RLE) and Huffman Encoding used in JPEG compression?