Detailed Project Report on:

Medical Image Processing Techniques

Showcasing hands-on experience

Tool: MATLAB

February 2026

Table of Contents

1. Grey Scale conversion, Binarization, Negative operation of a medical image.

THEORY:

Greyscale Conversion: Greyscale conversion transforms colour images into shades of grey by eliminating colour information while preserving luminance. In medical imaging, grayscale representation is crucial as most diagnostic images (X-rays, CT scans, MRI) are inherently monochromatic.

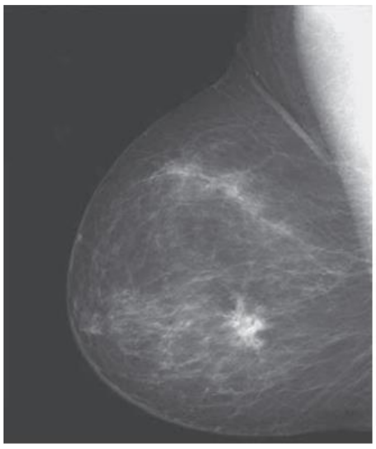
Image Binarization: Binarization converts grayscale images into binary (black and white) format by applying a threshold value. Pixels above the threshold become white (1), while those below become black (0). This technique is essential for segmentation, edge detection, and isolating regions of interest in medical images.

Image Negative: The operation inverts pixel intensities, transforming dark regions to bright & vice versa. For an 8-bit grayscale image, the transformation is: Negative = 255 - Original. In medical imaging, negatives can enhance visibility of certain anatomical structures.

These fundamental operations form the preprocessing foundation for advanced medical image analysis and diagnosis.

Input Image:

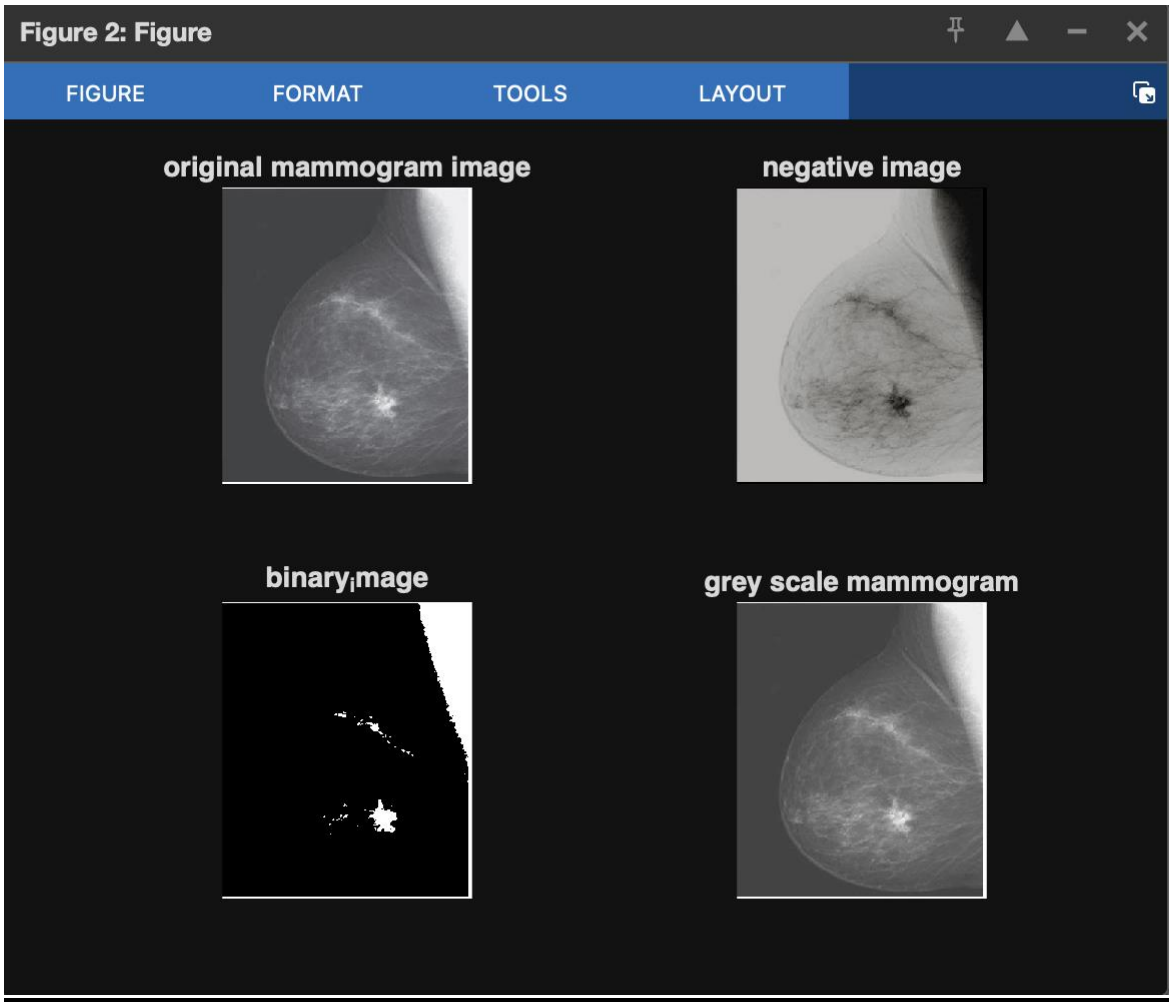
Digital Mammogram



CODE:

```
%reading image
img = imread(Theory/MATLAB Drive/mammogram.png');

%greyscaleconversion
grey_img = rgb2gray(img);

%imagenegative
negative_img = imcomplement(img) %helps to highlight white objects embedded in dark
background

%binarization
binary_image = imbinarize(gray_img)

%displaying all images in one window

figure

subplot(2,2,1)
imshow(img)
title('original mammogram image')

subplot(2,2,2)
imshow(negative_img)
title('negative image')

subplot(2,2,3)
imshow(binary_image)
title('binary_image')

subplot(2,2,4)
imshow(grey_img)
title('grey scale mammogram') %the original image itself seems in greyscale
```

RESULT:



DISCUSSION:

The complement operation successfully inverted the intensity values of the image, making bright regions dark and dark regions bright. This enhanced the visibility of certain features, which can be useful for better interpretation in medical imaging.

CONCLUSION:

The experiment demonstrated how to read an image and apply the complement operation, binarization and grey scale conversion.

2. Neighborhood analysis and the types of connectivity between pixels.

THEORY:

In digital image processing, each pixel has a specific location and can be related to other pixels through its neighborhood. The most common neighborhood types are: 4-neighbour, 8-neighbour, and diagonal neighbor.

Let V be the set of values used to determine connectivity

• 4-connectivity: Two pixels p and q with values from V are 4-connected if q is in the set N4 (p).

• 8-connectivity: Two pixels p and q with values from V are 8-connected if q is in the set N8 (p).

• m-connectivity (mixed): Two pixels p and q with values from V are m-connected if

i) q is in the set N4(p), or

ii) q is in the set ND(p) and the set N4(p) ∩ N4(q) is empty (This is the set of pixels that are 4-neighbors of p and q and whose values are from V)

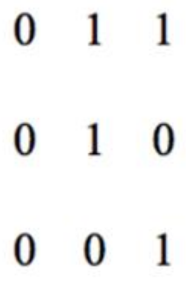


Fig: An arrangement of pixels

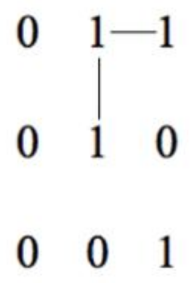


Fig: 4-connectivity of pixels

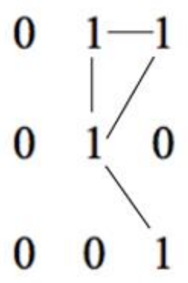


Fig: 8-connectivity of pixels

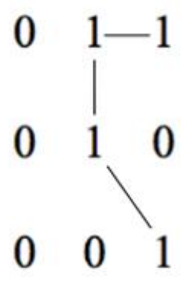
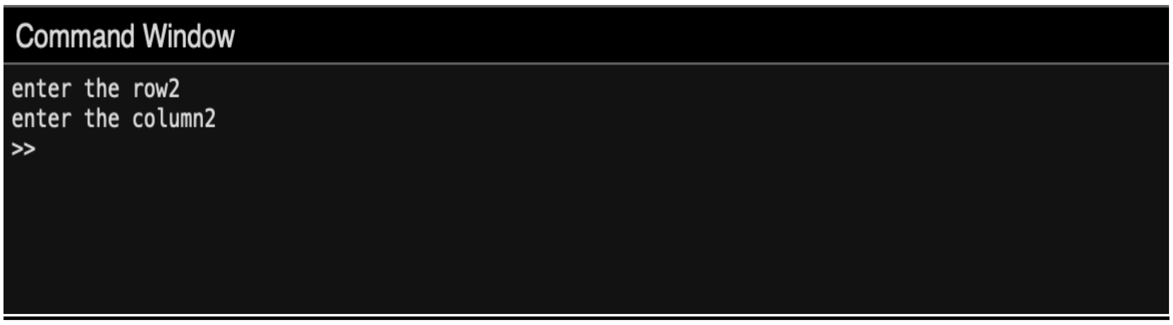


Fig: m-connectivity of pixels

CODE:

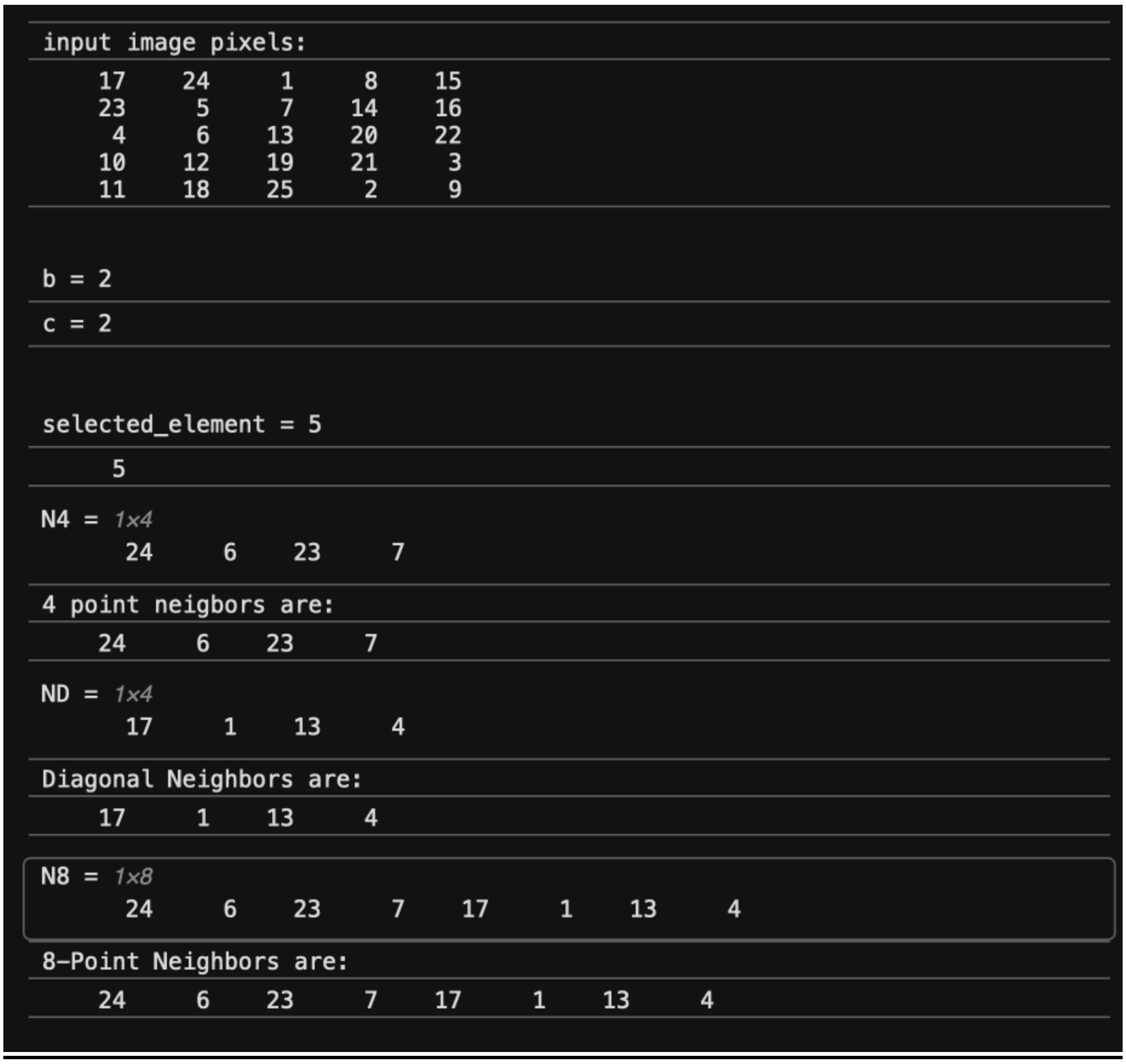
```
% Neighborhood Analysis
clc;

%creating a sample matrix
a=magic(5);
disp(a)

%taking user input for seed pixel
b = input('enter the row')
c = input('enter the column')

%display selected element
selected_element= a(b,c)
disp(selected_element)

%4-point neighbors (up, down, right, left)
N4 = [a(b-1,c), a(b+1,c), a(b,c-1), a(b, c+1)]
disp('4 point neigbors are:Theory)
disp(N4)

%Diagonal neighbors
ND = [a(b-1,c-1), a(b-1,c+1), a(b+1,c+1), a(b+1,c-1)];
disp('Diagonal Neighbors are:Theory);
disp(ND);

%8-point neigbors
N8 = [N4, ND];
disp('8-Point Neighbors are:Theory);
disp(N8);
```

RESULT:



Command Window

```
enter the row2
enter the column2
>>
```



```
input image pixels:
    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


b = 2

c = 2


selected_element = 5

     5

N4 = 1×4
      24     6    23     7

4 point neigbors are:
    24     6    23     7

ND = 1×4
      17     1    13     4

Diagonal Neighbors are:
    17     1    13     4

N8 = 1×8
      24     6    23     7    17     1    13     4

8-Point Neighbors are:
    24     6    23     7    17     1    13     4
```

DISCUSSION:

The experiment showed how different neighborhood definitions affect pixel connectivity. Using 4-neighbour, connectivity is limited to direct horizontal and vertical pixels, while 8-neighbour provides a more complete surrounding relationship. Diagonal neighbors help in identifying corner connections.

CONCLUSION:

The relationship between pixels can be defined in different ways depending on the application. Implementing 4, 8, and diagonal neighbors helps in understanding pixel connectivity and is essential for various image processing operations.

3. Histogram Equalization

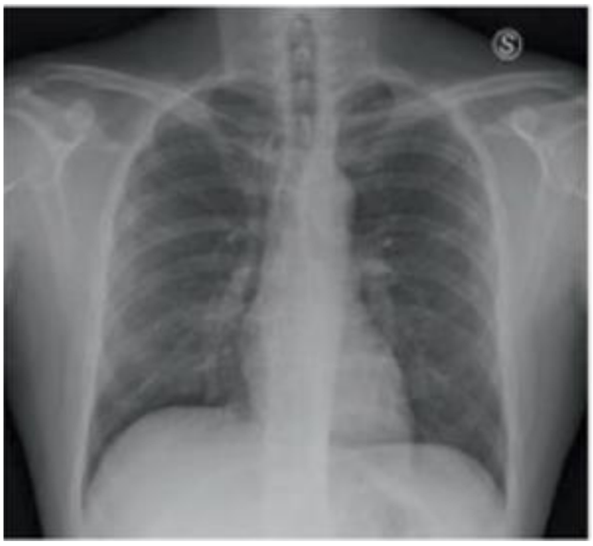
THEORY:

Histogram Equalization is a contrast enhancement technique used in digital image processing to improve the visual quality of an image. It works by redistributing the intensity values of an image so that they span a wider and more uniform range of gray levels. This is especially useful for images that are too dark, too bright, or have poor contrast due to limited dynamic range.

The histogram of an image represents the frequency of occurrence of each gray level. If the histogram is concentrated in a narrow region, the image appears low in contrast. Histogram equalization transforms the original gray levels using a mapping function derived from the cumulative distribution function (CDF) of the image's histogram.

Input Image:

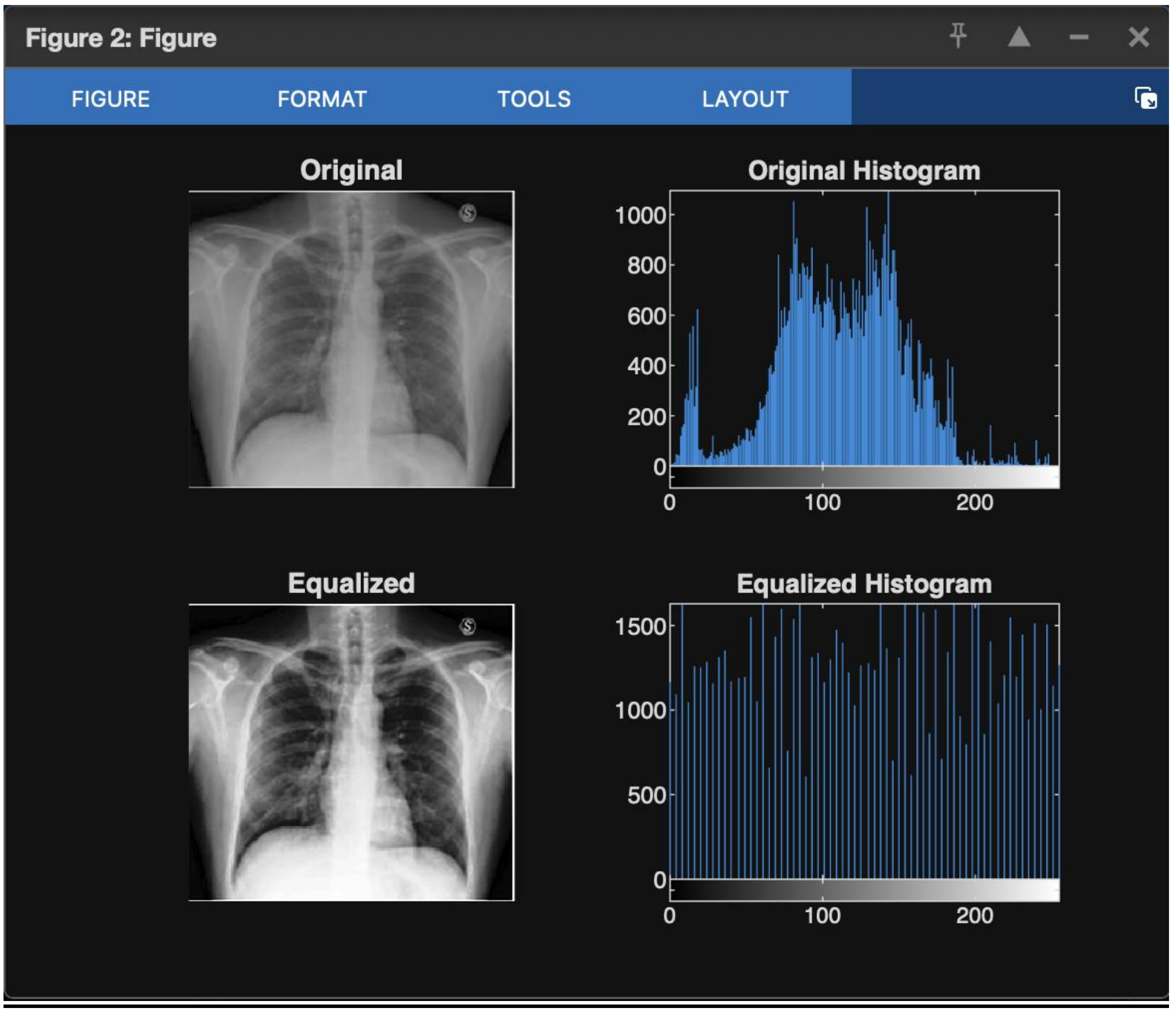
Chest X-ray



CODE:

```
% Read and convert image
I = imread('Theory/MATLAB Drive/xray.png');
I = rgb2gray(I);

% Histogram equalization
I_eq = histeq(I);

% Display
figure;
subplot(2,2,1), imshow(I), title('Original');
subplot(2,2,2), imhist(I), title('Original Histogram');
subplot(2,2,3), imshow(I_eq), title('Equalized');
subplot(2,2,4), imhist(I_eq), title('Equalized Histogram');
```

RESULT:



DISCUSSION:

In this experiment, histogram equalization was applied to enhance the contrast of a grayscale Xray image. The original image had intensity values concentrated in a narrow range, which resulted in low visual contrast and poor detail visibility. After applying histogram equalization, the pixel intensity values were redistributed more uniformly across the available gray levels, leading to a significant improvement in contrast.

CONCLUSION:

Histogram equalization effectively enhances image contrast by spreading intensity values across the full gray-level range. This improves detail visibility in low-contrast images, making it a useful and simple image enhancement technique.

4. TO DISPLAY THE BITPLANES OF AN IMAGE

THEORY:

Bit-plane slicing is a technique in digital image processing that analyzes the contribution of each bit in the binary representation of pixel intensities. In an 8-bit grayscale image, each pixel value ranges from 0 to 255 and can be represented using 8 bits:

b7, b6, b5, b4, b3, b2, b1, b0,

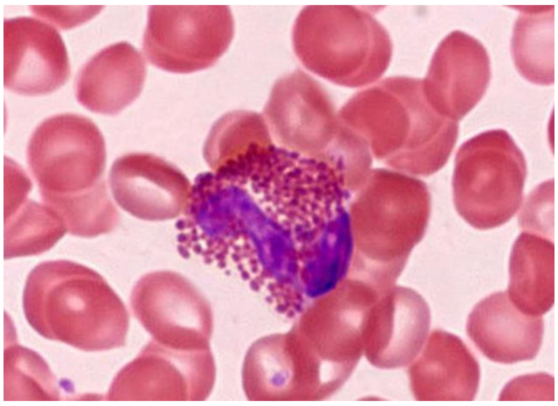
where b7 is the most significant bit (MSB) and b0 is the least significant bit (LSB). By displaying the bit-planes, we can observe how different bits contribute to the overall image appearance:

- Higher-order bit-planes (MSBs) represent the main shape and features.
- Lower-order bit-planes (LSBs) contain subtle details and noise.

Bit-plane slicing is useful in image compression, watermarking, steganography, and feature analysis because it helps separate visually significant information from less important data.

Input Image:

Microscopic view of eosinophil immune cell

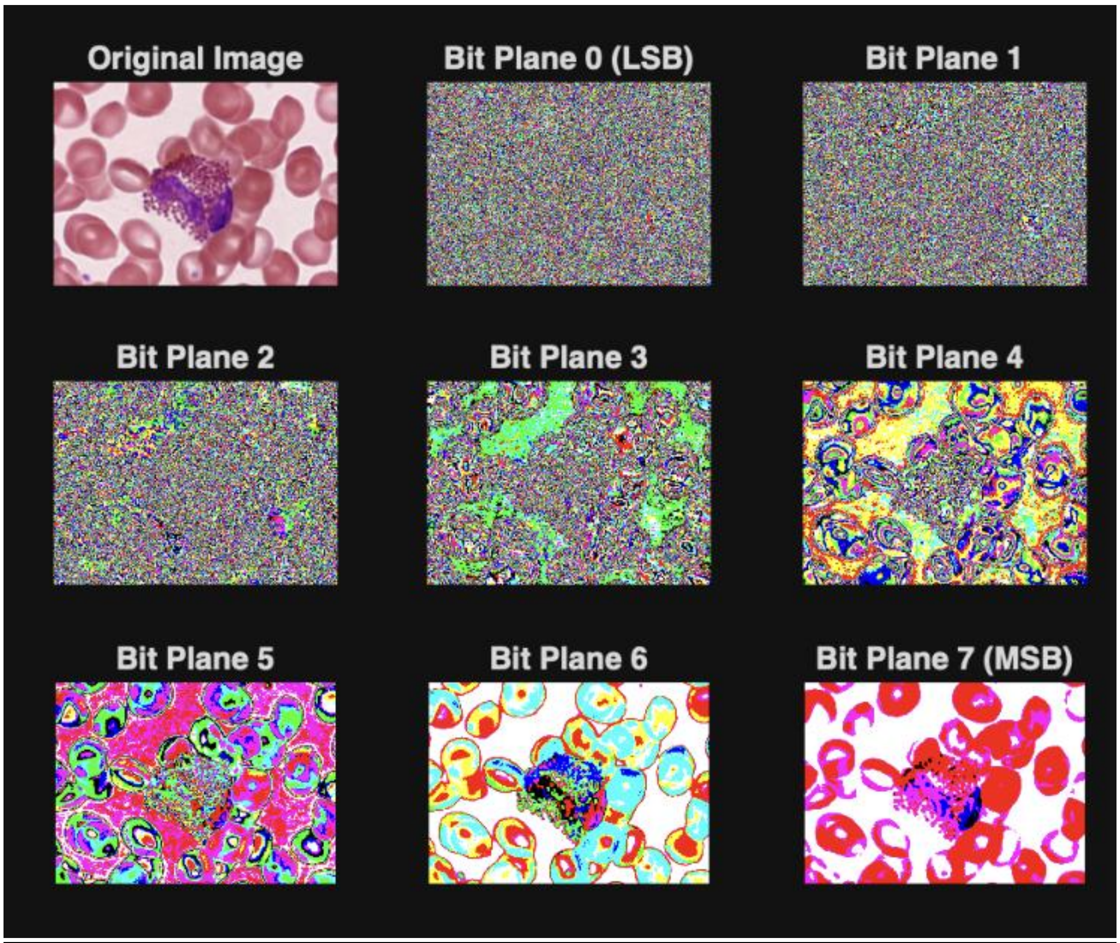


CODE:

```
% Bit Plane Slicing

% Read the image
I = imread(Theory/MATLAB Drive/cell.png');

%extracting bit planes
b0 = bitget(I, 1) * 255;  % Multiply by 255 to make visible
b1 = bitget(I, 2) * 255;
b2 = bitget(I, 3) * 255;
b3 = bitget(I, 4) * 255;
b4 = bitget(I, 5) * 255;
b5 = bitget(I, 6) * 255;
b6 = bitget(I, 7) * 255;
b7 = bitget(I, 8) * 255;


% Original image
subplot(3, 3, 1);
imshow(I);
title('Original Image');

% Bit plane 0 (LSB)
subplot(3, 3, 2);
imshow(b0);
title('Bit Plane 0 (LSB)Theory);

% Bit plane 1
subplot(3, 3, 3);
imshow(b1);
title('Bit Plane 1');

% Bit plane 2
subplot(3, 3, 4);
imshow(b2);
title('Bit Plane 2');

% Bit plane 3
subplot(3, 3, 5);
imshow(b3);
title('Bit Plane 3');

% Bit plane 4
subplot(3, 3, 6);
imshow(b4);
title('Bit Plane 4');

% Bit plane 5
subplot(3, 3, 7);
```

```
imshow(b5);
title('Bit Plane 5');

% Bit plane 6
subplot(3, 3, 8);
imshow(b6);
title('Bit Plane 6');

% Bit plane 7 (MSB)
subplot(3, 3, 9);
imshow(b7);
title('Bit Plane 7 (MSB)Theory);
```

RESULT:



DISCUSSION:

In this experiment, the image was decomposed into its individual bit-planes to analyze the contribution of each bit to the overall image. The higher-order bit-planes (especially the MSB planes) contained most of the visual and structural information, while the lower-order planes mainly represented fine details and noise. This shows that not all bits contribute equally to image perception.

CONCLUSION:

The experiment confirms that the most significant bit-planes preserve the essential features of the image, while the least significant planes contribute little visual information.

<u>5. Implementation of image smoothening filters</u>

THEROY:

Smoothing filters are used in digital image processing to reduce noise and small intensity variations while preserving the overall structure of an image. They operate by modifying each pixel value based on the values of its neighboring pixels, thereby producing a more visually uniform image.

<u>Mean Filter (Averaging Filter)</u>

The mean filter is a linear smoothing filter that replaces each pixel value with the average of the intensity values in its neighborhood. A small window (e.g., 3×3, 5×5) is moved across the image, and the center pixel is replaced by the mean of all the pixels inside that window.

Mathematically, for a window THEORY of size m x n:

$$g\ (x, \text{Theory}) = \frac{1}{mn} \sum_{(s,t)\, \in\, Sxy} g(s, t)$$

where;

- g (s, t) = original image
- g (x, Theory) = filtered image

The median filter is a non-linear smoothing filter. Instead of averaging, it replaces each pixel value with the median of the intensity values in its neighborhood window.

Mathematically, for a window THEORY of size m x n:

$$g\ (x, \text{Theory}) = \text{median}\ (g\ (s, t);\ \text{where}\ (s, t) \in Sxy)$$

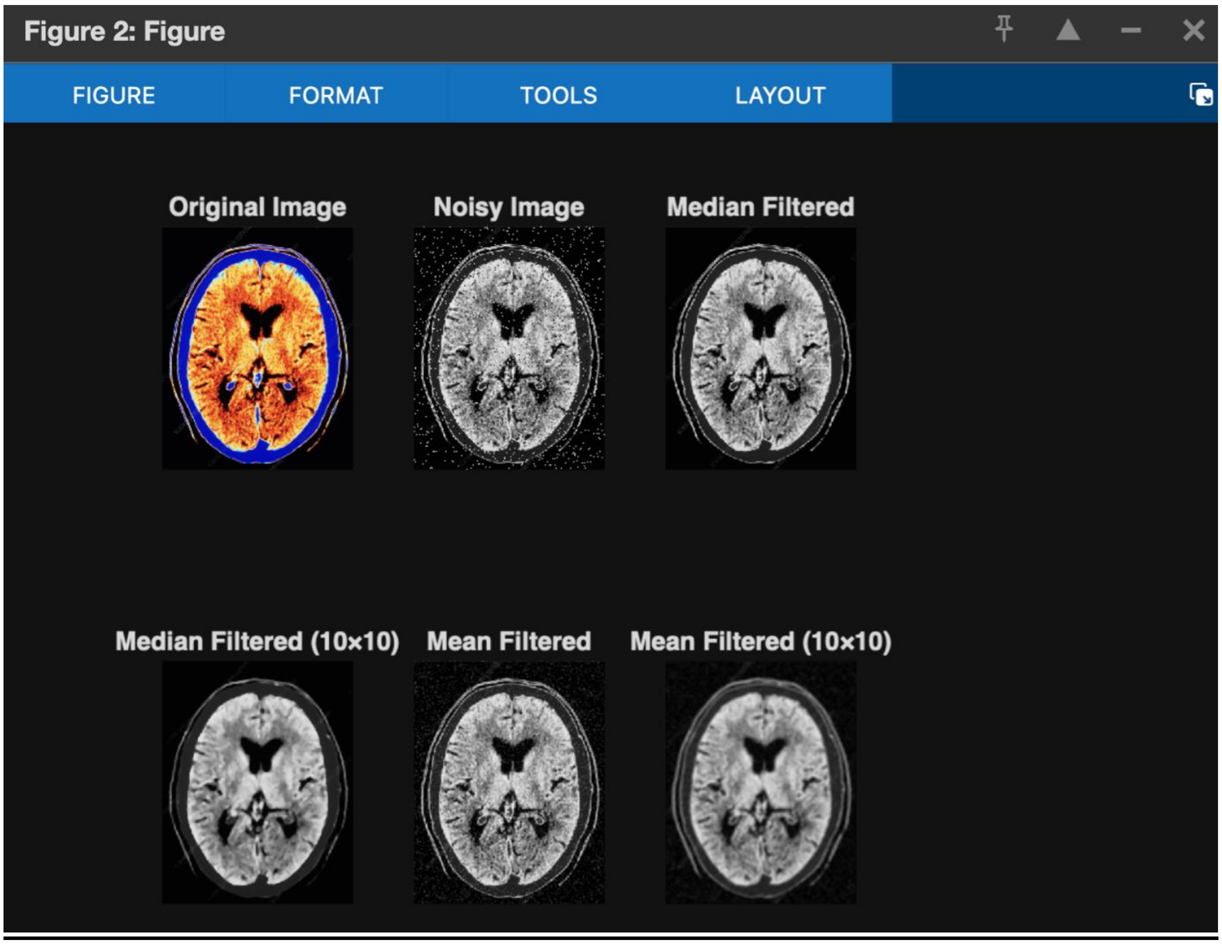
where;

- g (s, t) = original image
- g (x, Theory) = filtered image

CODE:

```
% Image Smoothing: Median and Mean Filters

% Read and convert image
I = imread(Theory/MATLAB Drive/color_CT-image.png');
I_grey = rgb2gray(I);

% Add noise for median filter demo
I_noisy = imnoise(I_grey, 'salt & pepper', 0.05);

% MEDIAN FILTERING
I_median3 = medfilt2(I_noisy, [3 3]);
I_median10 = medfilt2(I_noisy, [10,10]);

% MEAN FILTERING
filter_mean3 = fspecial('average', [3 3]);
I_mean3 = imfilter(I_noisy, filter_mean3);
filter_mean10 = fspecial('average', [10 10]);
I_mean10 = imfilter(I_noisy, filter_mean10);

% Display comparison

subplot(2, 4, 1);
imshow(I);
title('Original Image');

subplot(2, 4, 2);
imshow(I_noisy);
title('Noisy Image');

subplot(2, 4, 3);
imshow(I_median3);
title('Median Filtered');

subplot(2, 4, 5);
imshow(I_median10);
title('Median Filtered (10×10)Theory);

subplot(2, 4, 6);
imshow(I_mean3);
title('Mean Filtered');


subplot(2, 4, 7);
imshow(I_mean10);
title('Mean Filtered (10×10)Theory);
```

RESULT:



DISCUSSION:

In this experiment, both mean and median filters were applied to reduce noise in the image. The mean filter effectively smoothed random intensity variations but also caused noticeable blurring of edges and fine details. In contrast, the median filter preserved edges better while removing impulse (salt-and-pepper) noise. This highlights the importance of selecting the appropriate filter based on the type of noise present in the image.

CONCLUSION:

Hence, mean and median filters were used to reduce noise from the image.

6. Image Sharpening using Laplacian Filter

THEORY:

Image sharpening is a technique used in digital image processing to enhance edges and fine details in an image. It works by emphasizing rapid intensity changes, which usually correspond to object boundaries and important structural features. One of the most commonly used methods for sharpening is based on the Laplacian operator.

The Laplacian filter is a second-order derivative operator that highlights regions of rapid intensity change. Unlike first-order operators (such as gradient filters), the Laplacian responds strongly to edges regardless of their orientation. It is defined as the sum of second partial derivatives of the image:

$$\nabla^2 f\,(x, \text{Theory}) = \frac{\partial f}{\partial x2} + \frac{\partial f}{\partial y2}$$

In digital image processing, this operator is implemented using a convolution mask. Common Laplacian masks include:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

When an image is convolved with a Laplacian mask, the result contains strong responses at edges and fine details but little information in smooth regions. The sharpened image is then obtained by:

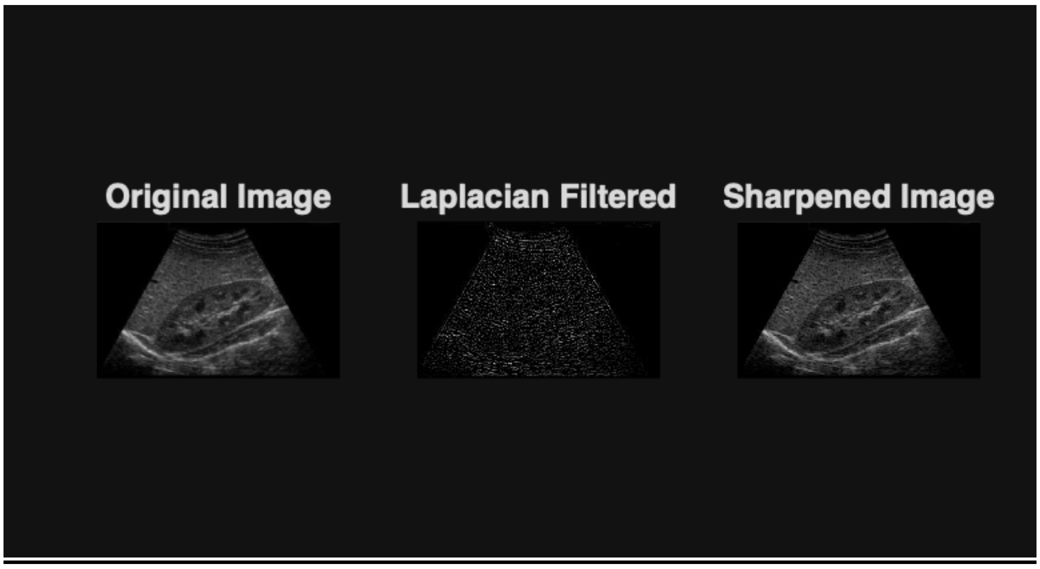
$g\,(x, \text{Theory}) = f\,(x, \text{Theory}) - \nabla^2 f\,(x, \text{Theory})$

- $f\,(x, \text{Theory})$ is the original image,
- $\nabla^2 f\,(x, \text{Theory})$ is the Laplacian of the image,

CODE:

```
% Read image
img = imread(Theory/MATLAB Drive/ultrasound.png');  % Replace with your image path

img_gray = rgb2gray(img);

% Create Laplacian filter
lap_filter = fspecial('laplacian', 0.2);

% Apply Laplacian filter
laplacian_img = imfilter(img_gray, lap_filter, 'replicate');

% Sharpen image = Original - Laplacian
sharpened_img = imsubtract(img_gray, laplacian_img);

% Display results

subplot(1, 3, 1);
imshow(img_gray);
title('Original Grayscale Image');

subplot(1, 3, 2);
imshow(laplacian_img, []);
title('Laplacian Filtered');

subplot(1, 3, 3);
imshow(sharpened_img);
title('Sharpened Image');
```

RESULT:



DISCUSSION

In this experiment, the Laplacian filter was used to enhance the sharpness of the image by emphasizing regions with rapid intensity changes. The filtered result highlighted edges and fine details that were less visible in the original image. Adding the Laplacian output back to the original image produced a noticeably sharper image with improved boundary definition. However, it was also observed that the Laplacian filter is sensitive to noise and can amplify unwanted variations, especially in noisy images. This shows that applying a smoothing filter before sharpening can improve the overall result.

CONCLUSION:

The Laplacian filter is an effective method for image sharpening because it enhances edges and fine details regardless of their orientation. The experiment confirms that Laplacian-based sharpening improves visual clarity, but care must be taken to control noise amplification.

7. Edge detection using derivative filters

THEORY:

Edge detection is a fundamental operation in image processing used to locate significant intensity transitions in an image. These transitions often correspond to object boundaries, surface markings, or changes in illumination. Gradient-based edge detection methods identify edges by computing the first derivative of image intensity and finding locations where the rate of change is high.

Let an image be represented as a 2D intensity function f (x, Theory).

$$\nabla f\,(x, \text{Theory}) = [\,\frac{\partial f}{\partial x}\,, \frac{\partial f}{\partial \text{Theory}}\,]$$

$$|\,\nabla f\,| = (G^2_x + G^2_y)^{1/2}$$

Where, Gx and Gy are the gradients in x and Theory directions respectively

a) Robert cross filter

$$H_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}, \quad H_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

b) Prewitt filter

$$H_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \quad H_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

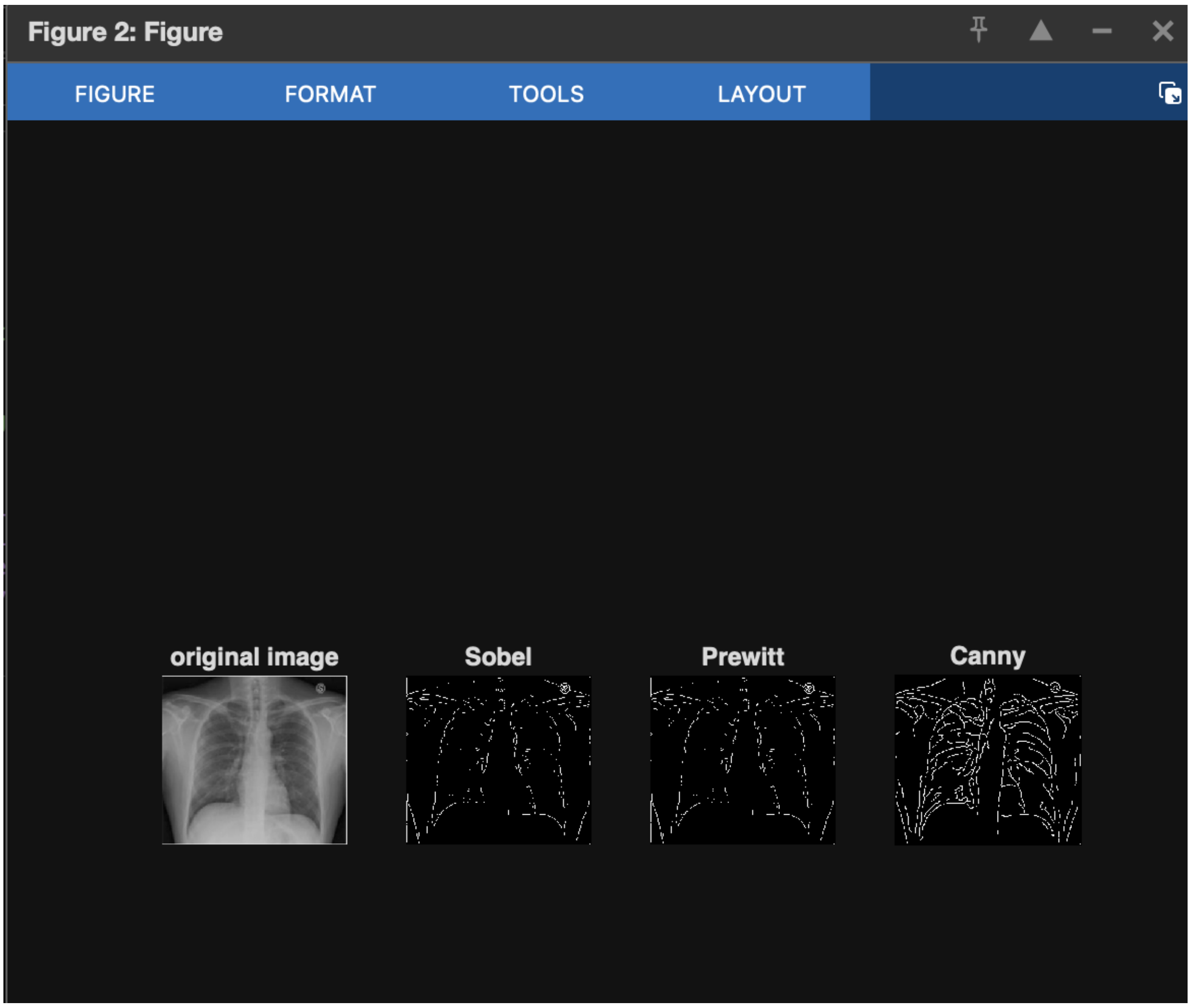
c) Sobel filter

$$H_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad H_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

CODE:

```
%reading image
img = imread(Theory/MATLAB Drive/xray.png');

%greyscale conversion
grey_img = rgb2gray(img);

% Applying edge detection filters with adjusted thresholds
sobel_edges = edge(grey_img, 'sobel', 0.03);      % Lower threshold
prewitt_edges = edge(grey_img, 'prewitt', 0.03);  % Lower threshold
canny_edges = edge(grey_img, 'canny'); % [low high] thresholds

%Display results
subplot(1,4,1), imshow(grey_img), title('original image');
subplot(1,4,2), imshow(sobel_edges), title('Sobel');
subplot(1,4,3), imshow(prewitt_edges), title('Prewitt');
subplot(1,4,4), imshow(canny_edges), title('Canny');
```

RESULT:



DISCUSSION:

Gradient-based edge detection effectively highlights object boundaries by responding to rapid changes in intensity. Operators like Sobel and Prewitt provide a good balance between edge localization and noise suppression, especially when preceded by smoothing. However, their sensitivity to noise and tendency to produce thick or fragmented edges can limit performance in low-contrast or noisy images. The choice of operator and threshold value strongly influences the quality of detected edges.

CONCLUSION:

Gradient-based filters are simple, fast, and widely used for detecting edges in digital images. By estimating first-order derivatives, they identify important structural features in an image.

8. Haar Transformation

THEORY:

The Haar Transform is the simplest form of the Discrete Wavelet Transform (DWT) and is widely used in image processing for data compression, feature extraction, and multi-resolution analysis. It represents an image in terms of average (low-frequency) and difference (high-frequency) components, allowing both spatial and frequency information to be analyzed simultaneously. Example of the transformation matrix of order 4.

4x4 transformation matrix is:

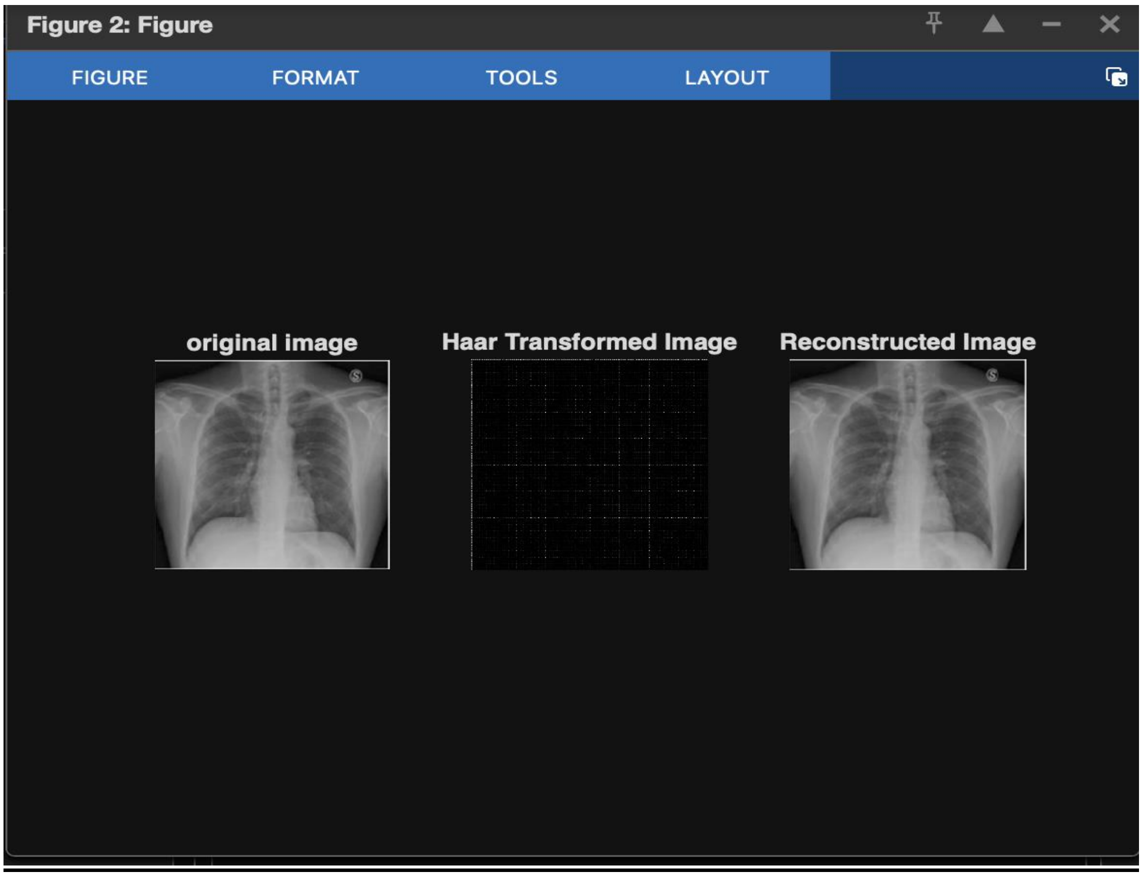
$$\mathbf{H}_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

CODE:

```
%% Read image
I = imread(Theory/MATLAB Drive/xray.png');

I = rgb2gray(I);

I = im2double(I);

%% Resize to power of 2
I = imresize(I,[256 256]);

%% Generate Haar matrix
N = 256;
THEORY = haar_matrix(N);

%% Apply Haar transform
F = THEORY * I * THEORY;

%original image
subplot(1,3,1)
imshow(I)
title('original image')
```

```
%% Show transform result
subplot(1,3,2)
imshow(log(abs(F)+1));
title('Haar Transformed Image');

%% Reconstruct image
I_rec = THEORY * F * THEORY;

%display reconstructed image
subplot(1,3,3)
imshow(I_rec);
title('Reconstructed Image');

%% ===== Haar Matrix Function =====
function THEORY = haar_matrix(N)

THEORY = 1;
Theory = [1 1; 1 -1] / sqrt(2);

while size(THEORY,1) < N
    THEORY = kron(THEORY,Theory);
end

end
```

RESULT:



DISCUSSION:

In this experiment, the 2D Haar transform was applied to a grayscale image using a Haar basis matrix.

CONCLUSION:

The original image was accurately reconstructed using the inverse transform, confirming the Haar transform's efficiency and lossless nature. This shows its usefulness in image compression and analysis.

9. Image Restoration Technique

THEORY:

Image restoration is a fundamental task in digital image processing that aims to recover an original (ideal) image from a degraded observation. Unlike enhancement (which is subjective and heuristic), restoration is model-based and relies on mathematical descriptions of the degradation process.

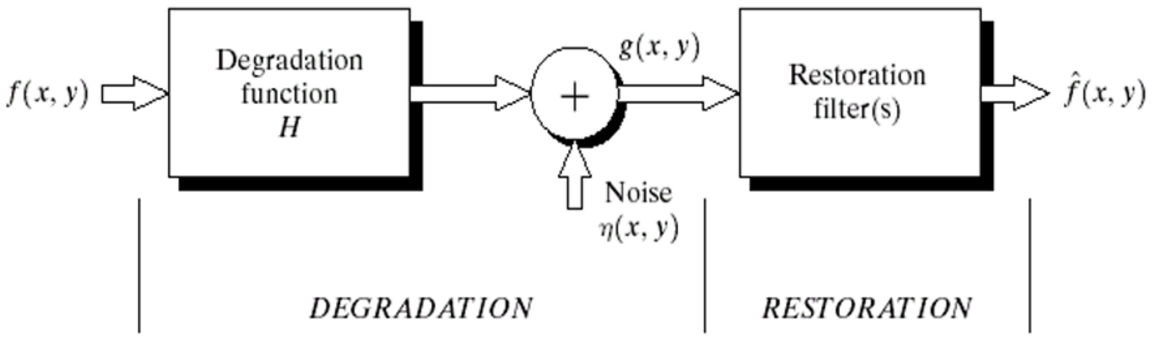


Figure: Image degradation and restoration

An observed image g(x, Theory) is commonly modeled as:

g (x, Theory) = Theory (x, Theory) ∗ f (x, Theory) + η (x, Theory)

where f (x, Theory) is the original image, Theory(x,Theory) is the degradation function in spatial domain, n (x, Theory) is the noise, and * is convolution.

Then, the degraded image in frequency domain is

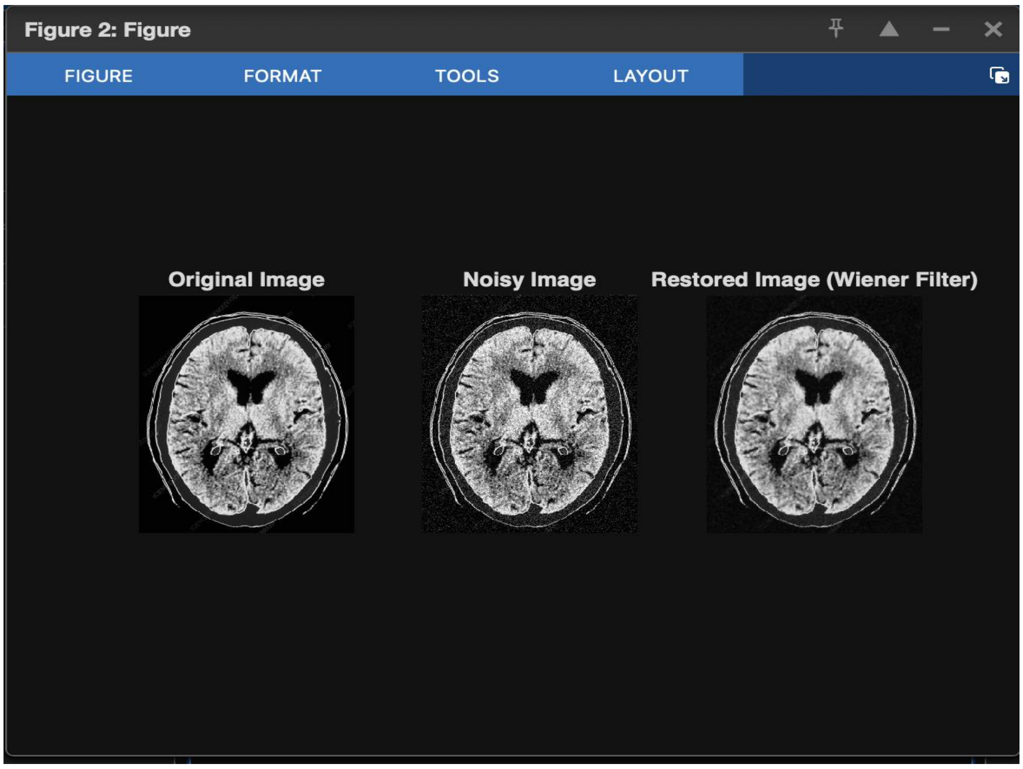
G (x, Theory) = F (x, Theory). THEORY (x, Theory) + N(x, Theory)

Where the capital letters are the Fourier transform of the corresponding terms in the spatial domain. The main objective of the restoration is to estimate restored image to original image. The more we learn about THEORY and n, the more restored image will get closer to the original image.

CODE:

```
% Read image
img = imread(Theory/MATLAB Drive/color_CT-image.png');  % Replace with your
image path

% Convert to grayscale
grayImg = rgb2gray(img);

% Add Gaussian noise to image
noisyImg = imnoise(grayImg, 'gaussian', 0, 0.01);

% Restore image using Wiener filter
restoredImg = wiener2(noisyImg, [5 5]);

% Display results

subplot(1, 3, 1);
imshow(grayImg);
title('Original Image');

subplot(1, 3, 2);
imshow(noisyImg);
title('Noisy Image');

subplot(1, 3, 3);
imshow(restoredImg);
title('Restored Image (Wiener Filter)Theory);
```

RESULT:



DISCUSSION:

The results of image restoration show that modeling the degradation process is essential for effective recovery of the original image. Methods such as inverse filtering can improve sharpness when noise is minimal, but they are highly sensitive to noise and often amplify unwanted artifacts. Wiener and regularized filters provide a better balance between deblurring and noise suppression, leading to visually smoother and more stable results. Spatial-domain denoising methods like median filtering are particularly effective for impulse noise, while adaptive and model-based techniques preserve important edges and details.

CONCLUSION:

Image restoration is a powerful, model-based approach to recovering degraded images by reducing blur and noise. By using appropriate filters and constraints, it is possible to reconstruct images that closely approximate the original scene.

10. IMAGE SEGMENTATION

THEORY:

Image segmentation is a core task in digital image processing and computer vision that involves partitioning an image into meaningful and non-overlapping regions. Each region corresponds to objects, parts of objects, or areas with similar properties such as intensity, color, or texture. The goal is to simplify the representation of an image so that it becomes easier to analyze and interpret.

Thresholding

The simplest method. Pixels are classified based on intensity values.

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) \geq T \\ 0, & \text{if } f(x,y) < T \end{cases}$$

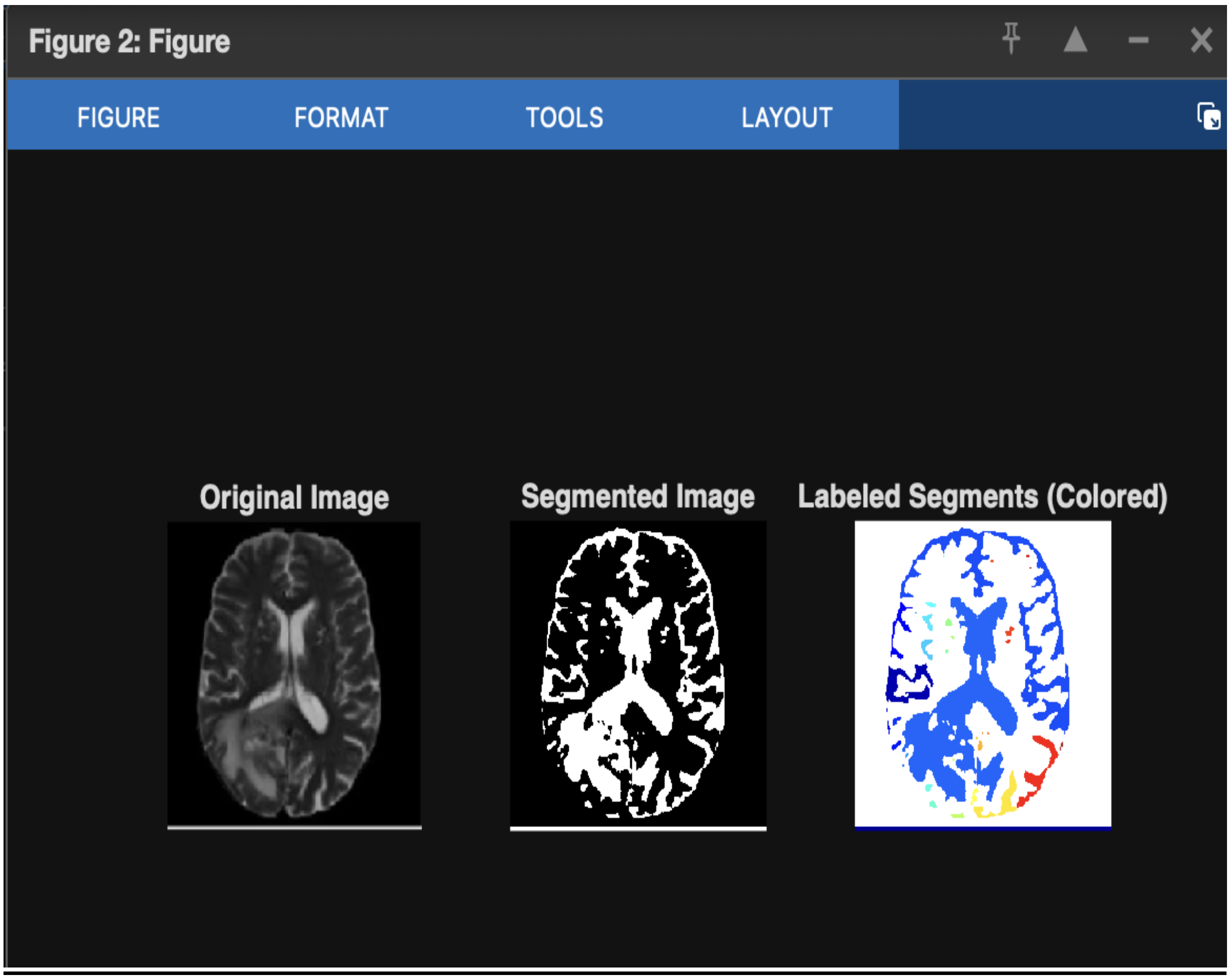
- Global thresholding: single threshold for whole image
- Adaptive thresholding: threshold varies according to the spatial locations
- Otsu's method: automatically selects the global optimal threshold

CODE:

```
% Image Segmentation using Thresholding

% Read image
img = imread(Theory/MATLAB Drive/brain.png');

grayImg = rgb2gray(img);


% Perform segmentation using Otsu's thresholding
level = graythresh(grayImg);  % Calculate threshold using Otsu's method
segmentedImg = imbinarize(grayImg, level);

% Label connected components
labeledImg = bwlabel(segmentedImg);
coloredSegments = label2rgb(labeledImg);

% Display results

subplot(1, 3, 1);
imshow(grayImg);
```

```
title('Original Image');

subplot(1, 3, 2);
imshow(segmentedImg);
title('Segmented Image');

subplot(1, 3, 3);
imshow(coloredSegments);
title('Labeled Segments (Colored)Theory);
```

RESULT:



DISCUSSION:

Simple thresholding is fast and effective when there is good contrast between objects and background, but it fails in images with uneven illumination or noise. Edge-based methods highlight object boundaries well, yet they often produce fragmented edges that require further processing. Region-based and clustering approaches provide more coherent segments by grouping similar pixels, making them useful for complex scenes and medical images. However, these methods can be computationally more expensive and sensitive to parameter selection such as the number of clusters or seed points.

CONCLUSION:

Image segmentation is a crucial step in image analysis, as it converts raw pixel data into meaningful regions for further processing and interpretation. By applying appropriate segmentation techniques based on image characteristics, accurate extraction of objects and structures is possible.