CSE-812 Lab Report Utilizing MATLAB R2023

Palash Hossen

October 2, 2024

1 Image Conversion: Indexed Image to Grayscale, RGB, and HSI

The goal is to convert an indexed image **"canoe.tif"** into different color formats: **Grayscale, RGB, and HSI** (approximated using HSV), and display the results.

1.1 Indexed to Grayscale

In the first task, the indexed image is converted into a grayscale image using the colormap associated with the indexed image. The implementation is given below:

```
close all;
clear all;
clc;

[indexedImage, map] = imread("D:\Academic\8th Semester\DIP_LAB\canoe.tif");
I = ind2gray(indexedImage,map);

subplot(1,2,1);
imshow(indexedImage, map); title('Indexed Image');
subplot(1,2,2);
imshow(I); title('Grayscale Image');
```

1.1.1 Output

The `ind2gray()` converts the indexed image to a grayscale image with colormap. The left image is the original indexed image, and the right image is the grayscale version.

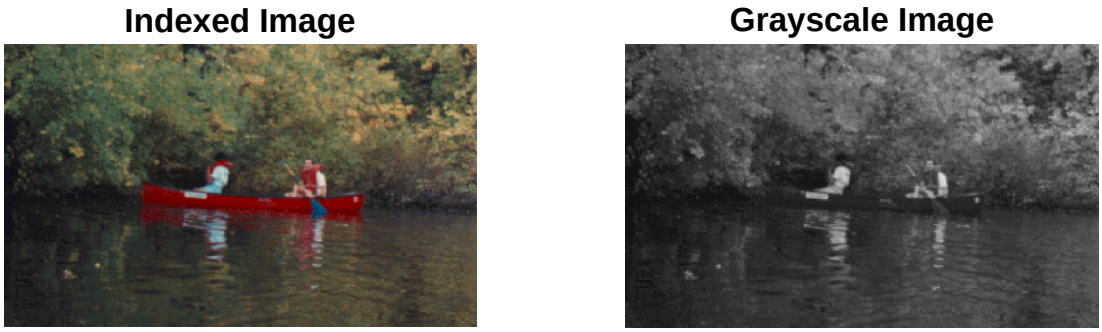

Figure 1: Indexed to grayscale conversion

1.2 Indexed to RGB

In the second task, the indexed image is converted into an RGB image using the colormap associated with the indexed image. The implementation is given below:

```
close all;
clear all;
clc;

[indexedImage, map] = imread("D:\Academic\8th Semester\DIP_LAB\canoe.tif");
I = ind2rgb(indexedImage,map);

subplot(1,2,1);
imshow(indexedImage, map); title('Indexed Image');
subplot(1,2,2);
imshow(I); title('RGB Image');
```

1.2.1 Output

The MATLAB code reads an indexed image and its colormap, converting it to RGB using `ind2rgb`. This transformation enables accurate color representation, essential for further processing. The left image is the original indexed image, and the right image is the RGB version.

Figure 2: Indexed to RGB conversion

1.3 Indexed to HSI

The MATLAB code reads an indexed image and its colormap, converting it to RGB using `ind2rgb`. This transformation enables accurate color representation, essential for further processing. Next, the RGB image is converted to HSV using `rgb2hsv`, approximating the HSI color space, which separates hue, saturation, and intensity. Finally, the original indexed image and the HSI approximation are displayed side by side using subplots, allowing for a clear visual comparison of the image formats and the effects of the conversion process. The left image is the original indexed image, and the right image is the HSI version.

```
close all;
clear all;
clc;

[indexedImage, map] = imread("D:\Academic\8th Semester\DIP_LAB\canoe.tif");
rgbImage = ind2rgb(indexedImage, map);
hsvImage = rgb2hsv(rgbImage);

subplot(1,2,1);
imshow(indexedImage, map); title('Indexed Image');
subplot(1,2,2);
imshow(hsvImage); title('HSI (Approximation using HSV)');
```

1.3.1 Output

The `rgb2hsv()` converts the indexed image to an HSV (Hue, Saturation, and Value) image, which is an approximation of the HSI model. The left image is the original indexed image, and the right image is the HSI (approximated by HSV).

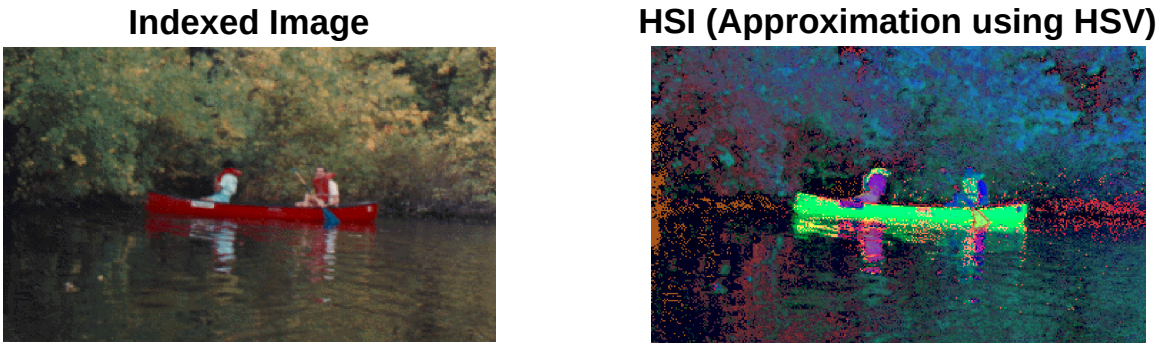

Figure 3: Indexed to HSI conversion

2 Bit Plane Slicing: Display the 8-bit Planes of an Image

Bit Plane Slicing (BPS) is a method of expressing an image in which each pixel is represented by one or more bits of the byte. To incorporate hidden data in any slice of eight slices, the BPS approach requires a bit-slicing algorithm. Each pixel is represented by 8 bits in this approach.

2.1 Displaying from LSB to MSB bit planes separately

The code reads a grayscale image using `imread()`, converts it to a double precision array, and utilizes `de2bi()` to extract bit planes. Each bit plane is reshaped using `reshape()` and displayed with `imshow()`, illustrating binary representations from LSB to MSB.

```
close all; clear all;  clc;
image = imread("D:\Academic\8th Semester\DIP_LAB\cameraman.tif");
[m,n] = size(image);
b = double(image);
c = de2bi(b);

r1 = reshape(c(:,1),m,n);
r2 = reshape(c(:,2),m,n);
r3 = reshape(c(:,3),m,n);
r4 = reshape(c(:,4),m,n);
r5 = reshape(c(:,5),m,n);
r6 = reshape(c(:,6),m,n);
r7 = reshape(c(:,7),m,n);
r8 = reshape(c(:,8),m,n);

subplot(2,4,1); imshow(r1); title('LSB Bit Plane');
subplot(2,4,2); imshow(r2); title('2nd Bit Plane');
subplot(2,4,3); imshow(r3); title('3rd Bit Plane');
subplot(2,4,4); imshow(r4); title('4th Bit Plane');
subplot(2,4,5); imshow(r5); title('5th Bit Plane');
subplot(2,4,6); imshow(r6); title('6th Bit Plane');
subplot(2,4,7); imshow(r7); title('7th Bit Plane');
subplot(2,4,8); imshow(r8); title('MSB Bit Plane');
```

2.1.1 Output

The output images are shown as the 8-bit planes in increasing order, starting from LSB, and ending at MSB.

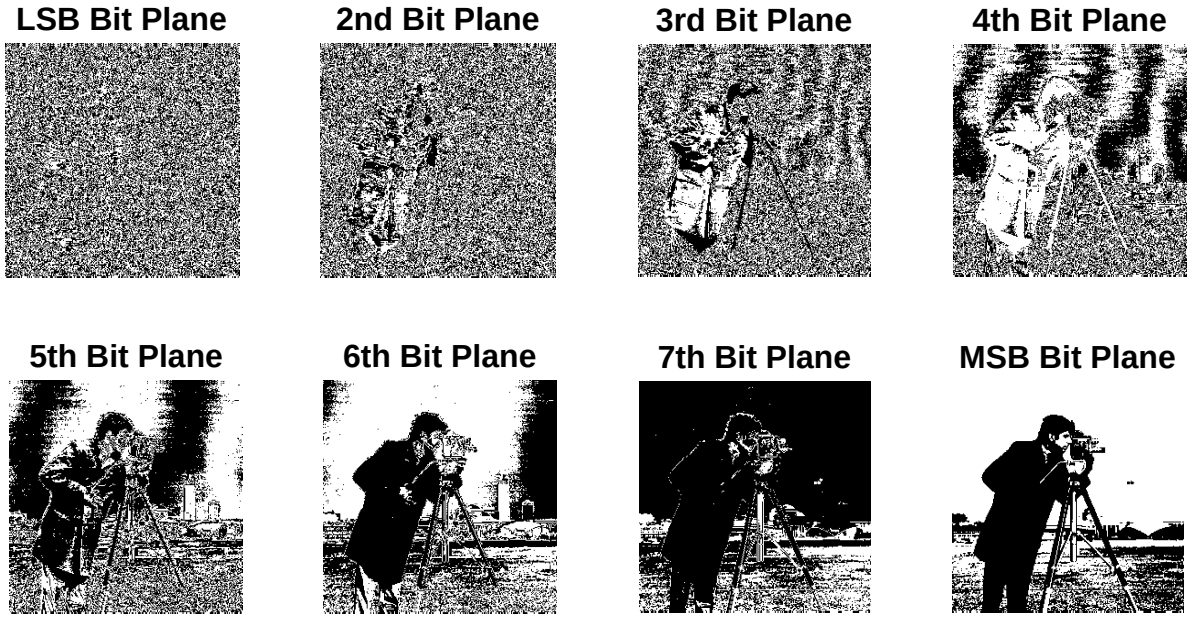

Figure 4: Bit Slices from LSB to MSB

2.2 Setting the LSB to zero

In this code, the **LSB** of each pixel is set to zero using `c(:,1) = 0`. The binary data is then converted back to a decimal using `bi2de()` and the image is reshaped with `reshape()` to its original dimensions. The modified image is displayed alongside the original for comparison.

```
close all; clear all; clc;
image = imread("D:\Academic\8th Semester\DIP_LAB\cameraman.tif");
[m,n] = size(image);
b = double(image);
c = de2bi(b);

% Set LSB (1st bit plane) to 0
c(:,1) = 0;
modified_LSB = bi2de(c);
modified_LSB_image = reshape(modified_LSB, m, n);

% Display the original and modified images
subplot(1,2,1);
imshow(image, []);title('Original Image');
subplot(1,2,2);
imshow(modified_LSB_image, []);title('LSB Set to 0');
```

2.2.1 Output

The left figure shows the original image and the right shows the image after setting the LSB to zero. There is **no significant change** in the quality.

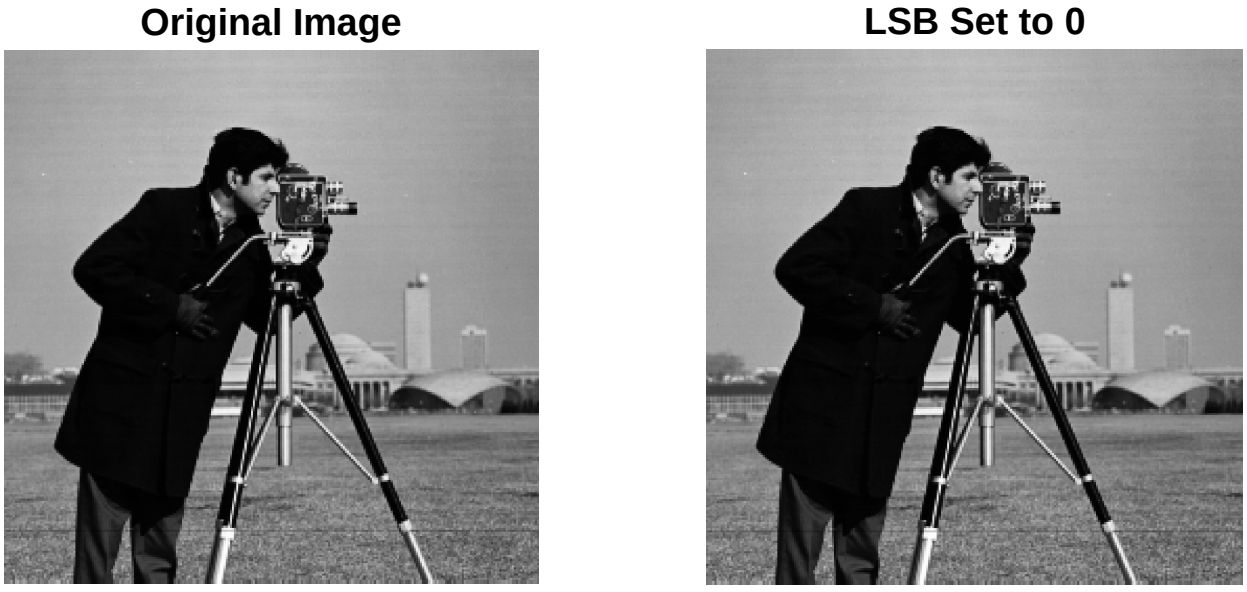

Figure 5: Setting the LSB to zero

2.3 Setting the MSB to zero

In this code, the **MSB** of each pixel is set to zero using `c(:,8) = 0`. The binary data is then converted back to a decimal using `bi2de()` and the image is reshaped with `reshape()` to its original dimensions. The modified image is displayed alongside the original for comparison.

```
close all; clear all; clc;

image = imread("D:\Academic\8th Semester\DIP_LAB\cameraman.tif");
[m,n] = size(image);
b = double(image);
c = de2bi(b);

% Set MSB (8th bit plane) to 0
c(:,8) = 0;
modified_MSB = bi2de(c);
modified_MSB_image = reshape(modified_MSB, m, n);

% Display the original and modified images
subplot(1,2,1);
imshow(image, []);title('Original Image');
subplot(1,2,2);
imshow(modified_MSB_image, []);title('MSB Set to 0');
```

2.3.1 Output

The left figure shows the original image and the right shows the image after setting the MSB to zero. There is a **significant loss** in the quality.

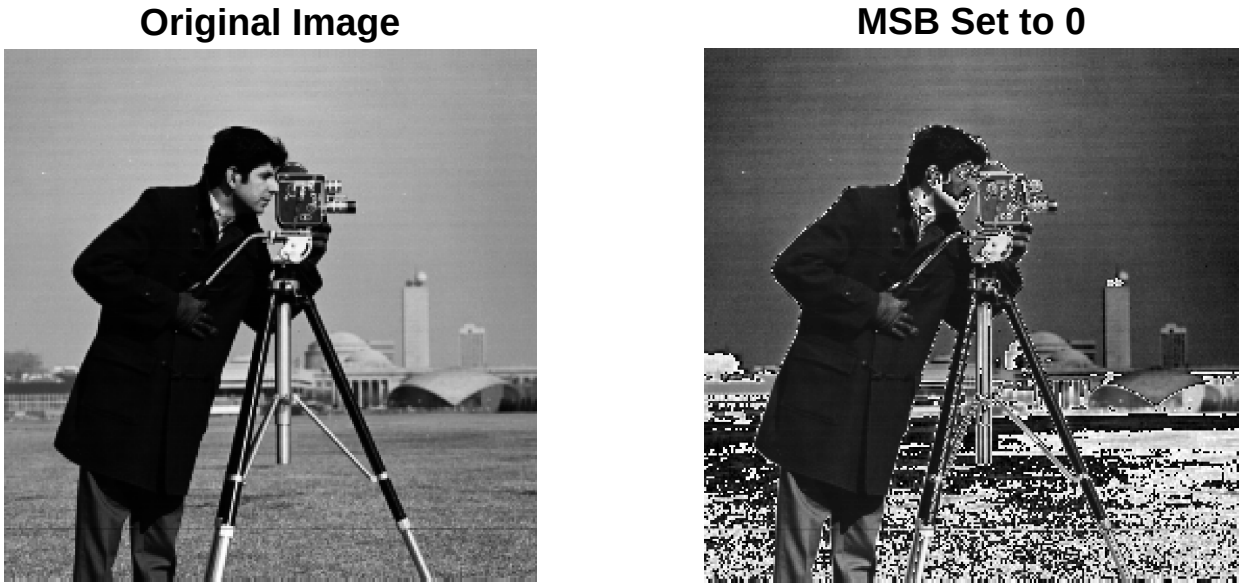

Figure 6: Setting the MSB to zero

3 Flip an Image That Looks Like a Mirror View of the Original Image

In the function `flip(image, 2)`, the number **2** specifies that the image is to be flipped along the **horizontal axis**. This means the operation is performed from left to right, resulting in a mirror image of the original.

```
close all; clear all; clc;

image = imread("D:\Academic\8th Semester\DIP_LAB\cameraman.tif");
% Flip the image horizontally (mirror image)
flippedImage = flip(image, 2);

subplot(1,2,1);
imshow(image); title('Original Image');
subplot(1,2,2);
imshow(flippedImage); title('Mirror Image');
```

3.1 Output

The left figure shows the original image and the right shows the image after flipping. The flipped image is the mirror view of the original image.

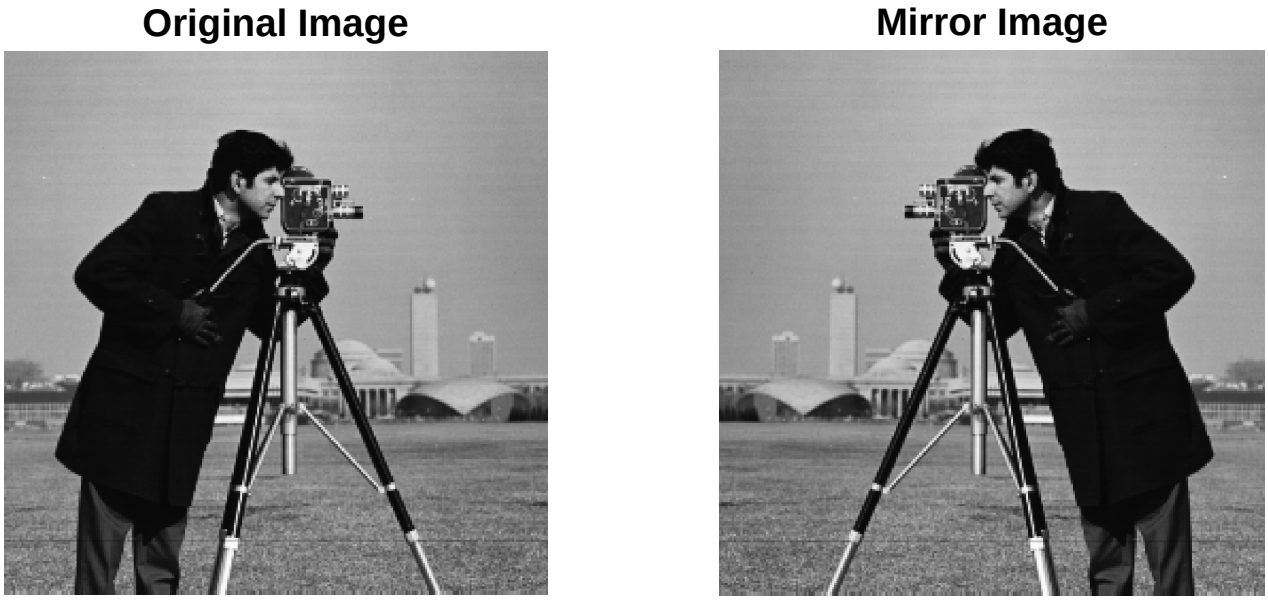

Figure 7: Mirror view of an image

4 Applying the Histogram Specification Method

This code performs histogram matching using several key functions: `imread` to read images, `rgb2gray` for grayscale conversion, `imhist` to compute histograms, and `cumsum` for cumulative distribution functions

(CDFs). It generates a histogram-matched image and displays the original, reference, and matched images along with their histograms.

```
close all;

A = imread("D:\Academic\8th Semester\DIP_LAB\CSECU.jpg");
Ref = imread("D:\Academic\8th Semester\DIP_LAB\cameraman.tif");

% Convert the RGB image to grayscale if necessary
if size(A, 3) == 3
    A_gray = rgb2gray(A);
else
    A_gray = A;
end

if size(Ref, 3) == 3
    Ref_gray = rgb2gray(Ref);
else
    Ref_gray = Ref;
end

[h_A, bins_A] = imhist(A_gray);
[h_Ref, bins_Ref] = imhist(Ref_gray);
cdf_A = cumsum(h_A) / numel(A_gray);
cdf_Ref = cumsum(h_Ref) / numel(Ref_gray);

% Create mapping from source to reference using CDF
mapping = zeros(256, 1);
for i = 1:256
    [~, idx] = min(abs(cdf_A(i) - cdf_Ref));
    mapping(i) = idx - 1;
end

matchedImage = uint8(mapping(double(A_gray) + 1));
figure('Position', [100, 100, 1200, 800]);

subplot(2, 3, 1); imshow(A_gray); title('Original Image (Grayscale)');
subplot(2, 3, 2); imshow(Ref_gray); title('Reference Image (Grayscale)');
subplot(2, 3, 3); imshow(matchedImage); title('Histogram Matched Image');

% Display histograms
subplot(2, 3, 4); imhist(A_gray); title('Histogram of Original Image');
subplot(2, 3, 5); imhist(Ref_gray); title('Histogram of Reference Image');
subplot(2, 3, 6); imhist(matchedImage); title('Histogram of Matched Image');
```

4.1 Output

The original image (upper left and comparatively lighter), the referenced image (upper middle), and histogram matched image (upper right and comparatively darker), and their respective histograms are shown below:

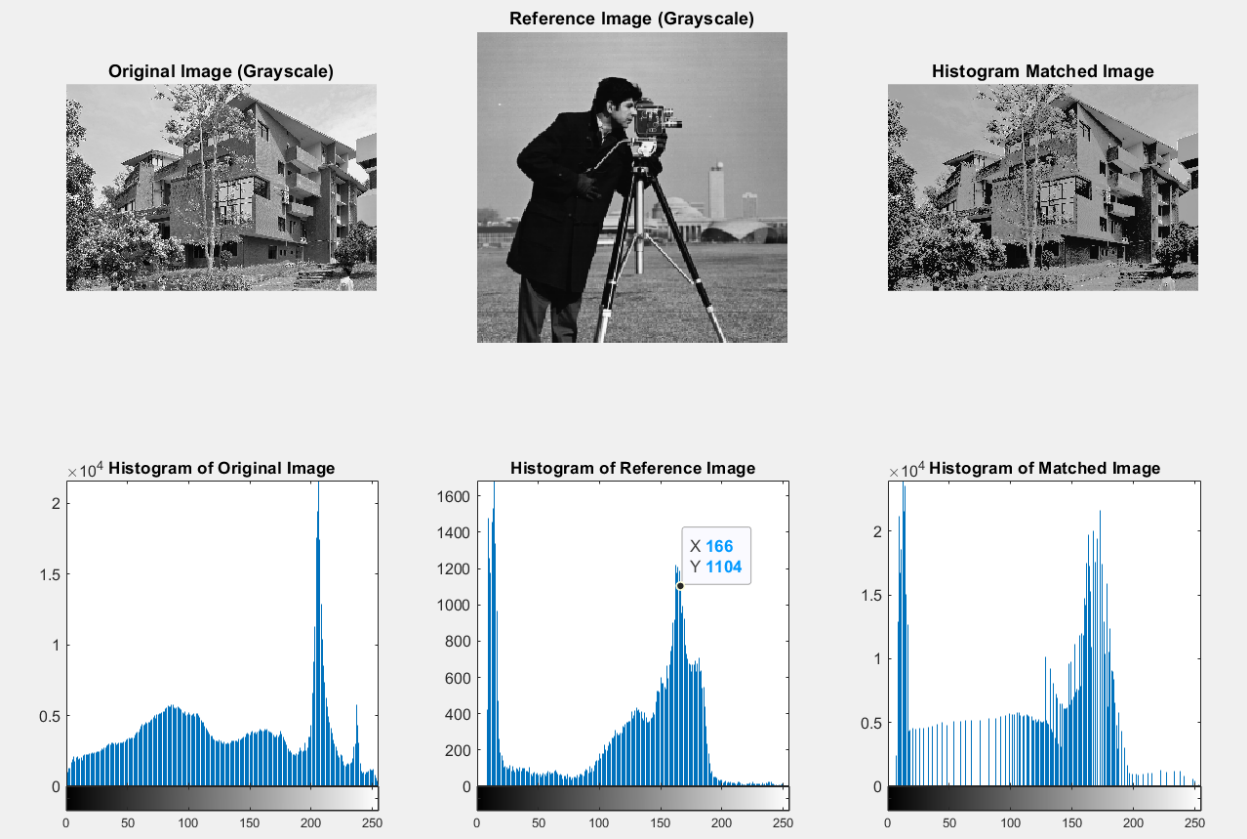

Figure 8: The histogram specification method

5 First Fourier Transform (FFT) of an Image

This code computes the Fast Fourier Transform (FFT) of the "cameraman.tif" image using key functions: `imread` to load the image, `rgb2gray` for grayscale conversion, `fft2` to compute the 2D FFT, and `fftshift` to center the zero-frequency component.

```
close all;
img = imread('D:\Academic\8th Semester\DIP_LAB\cameraman.tif');
% Convert to grayscale if needed (in case it's an RGB image)
if size(img, 3) == 3
    img = rgb2gray(img);
end

fft_img = fft2(double(img));
fft_img_shifted = fftshift(fft_img);
magnitude_spectrum = log(1 + abs(fft_img_shifted));

figure;
subplot(1, 2, 1); imshow(img, []); title('Original Image (Cameraman)');
subplot(1, 2, 2); imshow(magnitude_spectrum, []); title('Magnitude Spectrum of FFT');
```

5.1 Output

The left shows the original image and the right shows the magnitude spectrum of the FFT of the original image.

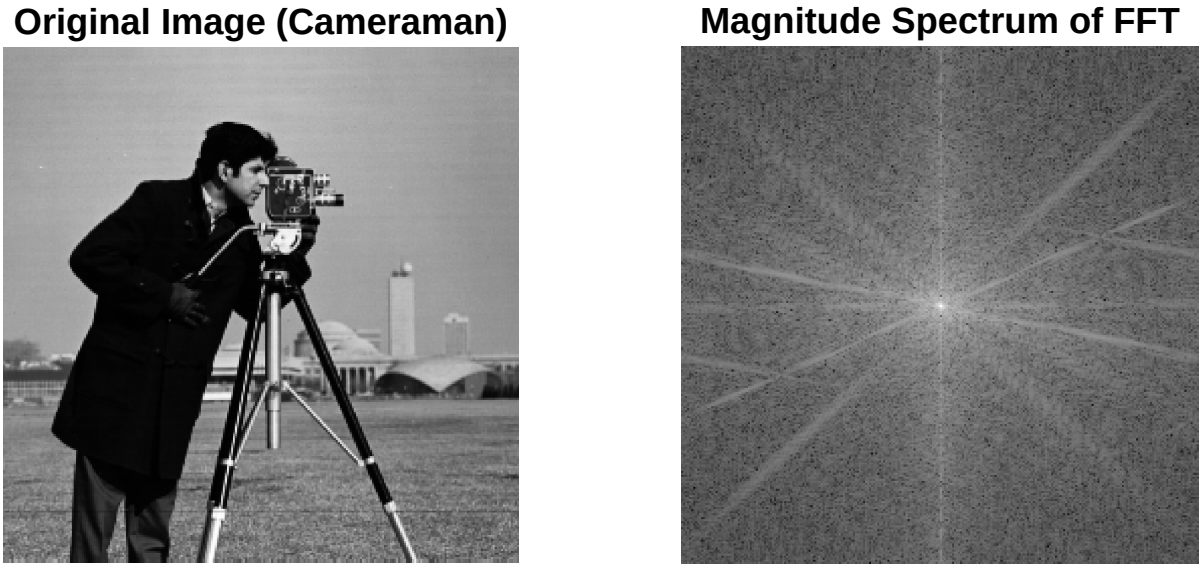

Figure 9: The magnitude spectrum of the FFT of a grayscale image

6 Implementing Morphological Operations Like Erosion, Dilation, Opening, and C"originatingScript":"m2","payload":"guid":"9b6a56e3-baf6-49e1-8ec2-0602493f67f99b0384","muid":"56e79c93-36a9-4658-9ca5-3d953e9068d77f8a12","sid":"a7e2d56c-6a97-4d76-925a-042e14fb5547

The MATLAB code demonstrates the use of key functions such as `imerode` for erosion, `imdilate` for dilation, `imbinarize` for binarization, and morphological operations like `imopen` for opening and `imclose` for closing. These are applied to the input image "casablanca.tif".

6.1 Output

The upper row shows the original, eroded, and dilated images respectively. On the other hand, the lower row shows the binary, opened, and closed images respectively.

```
close all;
input = imread("D:\Academic\8th Semester\DIP_LAB\casablanca.tif");

kernel = ones(5, 5);
img_erosion = imerode(input, kernel);
img_dilation = imdilate(input, kernel);
binary_image = imbinarize(input, 128/255);
opening = imopen(binary_image, kernel);
closing = imclose(binary_image, kernel);

figure; subplot(2, 3, 1); imshow(input); title('Original Image');
subplot(2, 3, 2); imshow(img_erosion); title('Erosion');
subplot(2, 3, 3); imshow(img_dilation); title('Dilation');
subplot(2, 3, 4); imshow(binary_image); title('Binary Image');
subplot(2, 3, 5); imshow(opening); title('Opening');
subplot(2, 3, 6); imshow(closing); title('Closing');
```

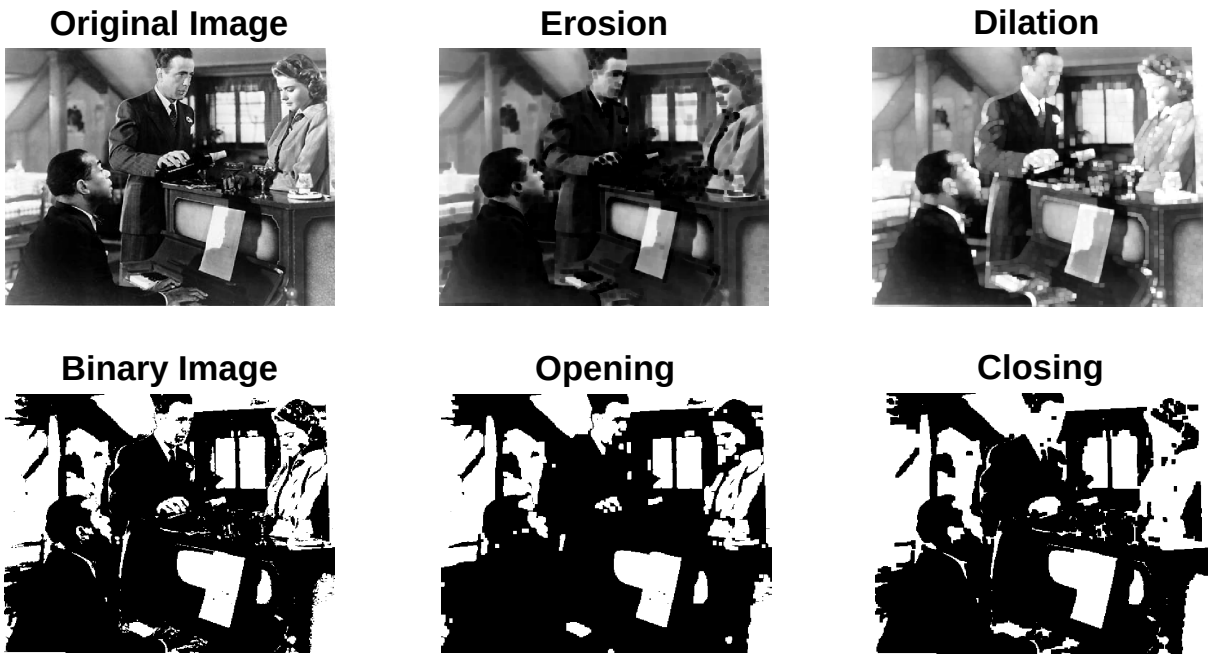

Figure 10: Bit Slices from LSB to MSB