Lab Report: 07

Title: Morphological Operations and Fourier Transform of an Image

Course title: Digital Image Processing Laboratory

Course code: CSE-406

4th Year 1st Semester Examination 2023

Date of Submission: 20/10/2024

Submitted to-

Dr. Md. Golam Moazzam
Professor
Department of Computer Science and Engineering
Jahangirnagar University
&
Dr. Morium Akter
Professor
Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Class Roll	Exam Roll	Name
353	202165	Shanjida Alam

Experiment No: 01

Experiment Name: Dilation of an Image Using Morphological Operations

Objectives:

1. The objective of this experiment is to apply the dilation operation, which enlarges the boundaries of objects in an image.
2. Dilation is useful for emphasizing specific features, such as connecting broken parts of an object.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
input_image = cv2.imread('nature.jpeg', cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5, 5), np.uint8)
dilated_image = cv2.dilate(input_image, kernel, iterations=1)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(input_image, cmap='gray')
plt.title('Original Grayscale Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(dilated_image, cmap='gray')
plt.title('Dilated Grayscale Image')
plt.axis('off')
plt.show()
```

Output:

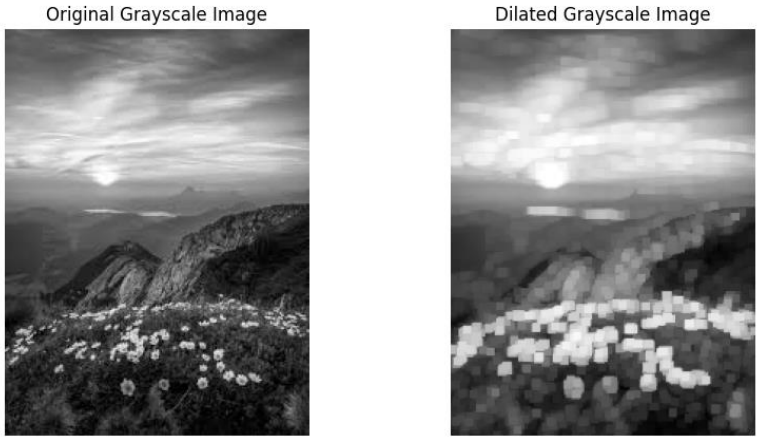

Figure 1.1: Dilation of an image using morphological operations in Python

Explanation:

1. This code reads an image in grayscale and applies a dilation operation using a 5x5 kernel.
2. The dilation process expands the white regions in the image, which emphasizes the boundaries of the objects.
3. The images before and after dilation are displayed side by side.

Code-02: MATLAB

```
inputImage = imread('nature.jpeg');
if size(inputImage, 3) == 3
   inputImage = rgb2gray(inputImage);
end
se = strel('square', 3);
dilatedImage = imdilate(inputImage, se);
figure;
subplot(1, 2, 1);
imshow(inputImage);
title('Original Image');
subplot(1, 2, 2);
imshow(dilatedImage);
title('Dilated Image');
```

Output:

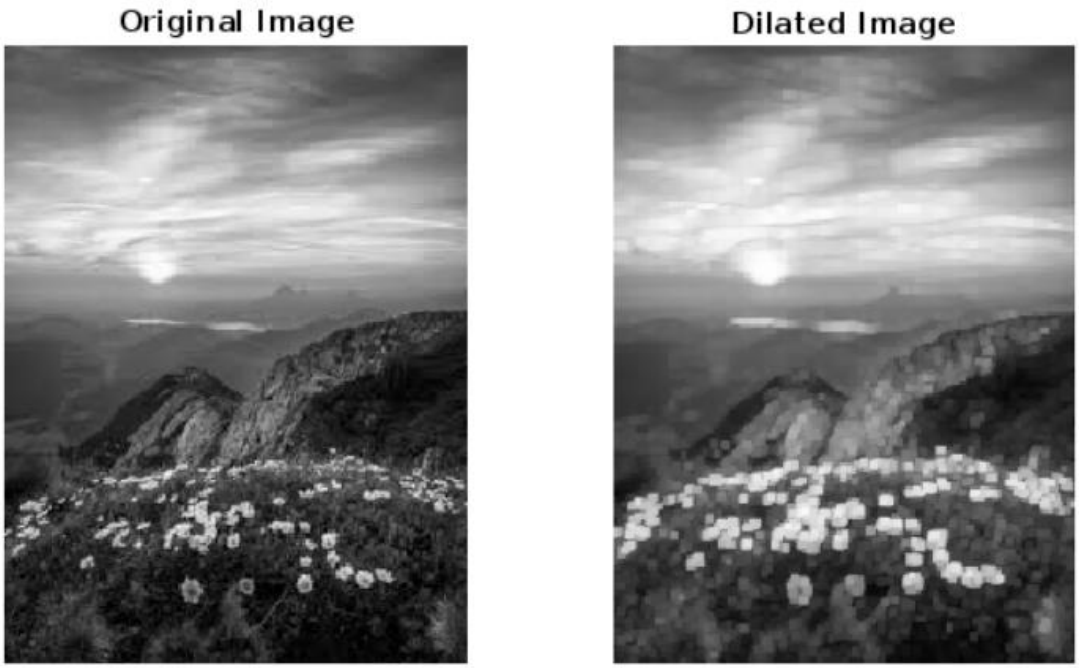

Figure 1.2: Dilation of an image using morphological operations in MATLAB

Explanation:

1. The code reads an input image and creates a 3x3 square structuring element (strel).
2. The imdilate() function is used to apply dilation, which expands object boundaries and fills gaps in the image.
3. The original and dilated images are displayed side by side using subplot for comparison.

Experiment No: 02

Experiment Name: Erosion Operation of an Image Using Morphological Transformations
Objectives:

1. The objective of this experiment is to apply the erosion operation to an image.
2. Erosion reduces the size of objects by eroding away the boundaries, which helps in eliminating small noise and separating objects that are close to each other.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
input_image = cv2.imread('nature.jpeg', cv2.IMREAD_GRAYSCALE)
kernel = np.ones((3, 3), np.uint8)
eroded_image = cv2.erode(input_image, kernel, iterations=1)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(input_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(eroded_image, cmap='gray')
plt.title('Eroded Image')
plt.axis('off')
plt.show()
```

Output:

Figure 2.1: Erosion operation of an image in python

Explanation:

1. This code begins by loading an image in grayscale, followed by the creation of a structuring element (5x5 matrix) used for the erosion operation.
2. The cv2.erode() function is applied to shrink the object boundaries in the image. Erosion is effective in removing small artifacts and noise, and also separates objects that are connected by thin structures.
3. The original and eroded images are displayed side by side for comparison.

Code-02: MATLAB

```
inputImage = imread('nature.jpeg');
if size(inputImage, 3) == 3
    inputImage = rgb2gray(inputImage);
end
se = strel('square', 3);
erodedImage = imerode(inputImage, se);
figure;
subplot(1, 2, 1);
imshow(inputImage);
title('Original Image');
subplot(1, 2, 2);
imshow(erodedImage);
title('Eroded Image');
```

Output:

Figure 2.2: Erosion operation of an image in MATLAB

Explanation:

1. The image is read and a 3x3 square structuring element is created.
2. The imerode() function applies erosion to shrink the object boundaries in the image.
3. This operation is commonly used to eliminate small noise or detach objects that are connected.
4. The original and eroded images are displayed for visual comparison.

Experiment No: 03

Experiment Name: Opening and Closing Operations in Image Processing

Objectives:

1. This experiment aims to apply both opening and closing operations on an image.
2. Opening is used to remove small objects or noise from the image, while closing is used to fill small holes or gaps inside objects.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
input_image = cv2.imread('nature.jpeg', cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5, 5), np.uint8)
opened_image = cv2.morphologyEx(input_image, cv2.MORPH_OPEN, kernel)
closed_image = cv2.morphologyEx(input_image, cv2.MORPH_CLOSE, kernel)
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(input_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.imshow(opened_image, cmap='gray')
plt.title('Opened Image')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.imshow(closed_image, cmap='gray')
plt.title('Closed Image')
plt.axis('off')
plt.show()
```

Output:

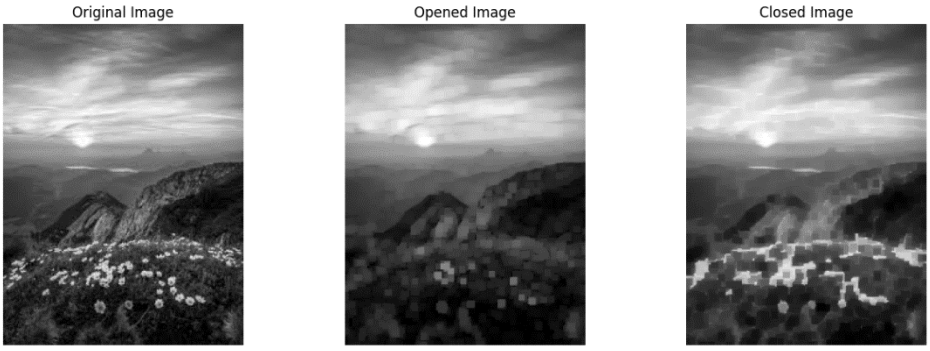

Figure 3.1: Opening and closing operations of an image using Python

Explanation:

1. This code uses cv2.morphologyEx() to perform both opening and closing operations on the image.
2. Opening involves erosion followed by dilation to remove small noise, while closing involves dilation followed by erosion to fill small gaps inside objects.
3. The input image is processed through both operations, and the results are displayed side by side with the original image for comparison.
4. The same structuring element is used for both opening and closing.

Code-02: MATLAB

```
inputImage = imread('nature.jpeg');
if size(inputImage, 3) == 3
    inputImage = rgb2gray(inputImage);
end
se = strel('square', 3);
openedImage = imopen(inputImage, se);
closedImage = imclose(inputImage, se);
figure;
subplot(1, 3, 1);
imshow(inputImage);
title('Original Image');
subplot(1, 3, 2);
imshow(openedImage);
title('Opened Image');
subplot(1, 3, 3);
imshow(closedImage);
title('Closed Image');
```

Output:

Figure 3.2: Opening and closing operations of an image using MATLAB

Explanation:

1. The input image is processed using both opening and closing operations.
2. Opening (via imopen()) helps remove small objects and noise, while closing (via imclose()) fills small holes and gaps in objects.
3. The structuring element for both operations is a 3x3 square.
4. The results are shown alongside the original image for a clear comparison of their effects.

Experiment No: 04

Experiment Name: Fast Fourier Transform (FFT) for Frequency Domain Analysis of an Image

Objectives:

1. The goal of this experiment is to apply the Fast Fourier Transform (FFT) on an image to convert it from the spatial domain to the frequency domain.
2. This technique helps to analyze the frequency components of the image, allowing for frequency-based processing.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
input_image = cv2.imread('nature.jpeg', cv2.IMREAD_GRAYSCALE)
fft_image = np.fft.fft2(input_image)
fft_shifted = np.fft.fftshift(fft_image)
magnitude_spectrum = 20 * np.log(np.abs(fft_shifted) + 1)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(input_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum (FFT)')
plt.axis('off')
plt.show()
```

Output:

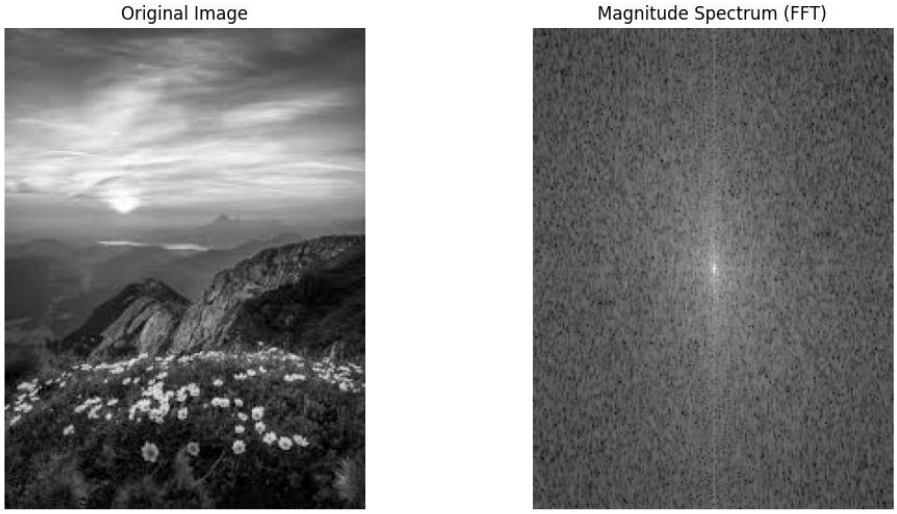

Figure 4.1: FFT of an image using python code

Explanation:

1. The code begins by reading an image in grayscale.
2. The np.fft.fft2() function computes the 2D Fast Fourier Transform (FFT) of the image, converting it to the frequency domain.
3. The np.fft.fftshift() function centers the low-frequency components. The magnitude spectrum of the FFT is then computed and displayed using a logarithmic scale to enhance visibility.
4. This technique allows for visualizing the frequency components of the image, highlighting areas with low and high-frequency content.

Code-02: MATLAB

```
inputImage = imread('nature.jpeg');
if size(inputImage, 3) == 3
    inputImage = rgb2gray(inputImage);
end
fftImage = fft2(double(inputImage));
fftImageShifted = fftshift(fftImage);
magnitudeFFT = abs(fftImageShifted);
magnitudeFFTLog = log(1 + magnitudeFFT);
figure;
subplot(1, 2, 1);
imshow(inputImage);
```

```
title('Original Image');
subplot(1, 2, 2);
imshow(magnitudeFFTLog, []);
title('Magnitude Spectrum (FFT)');
```

Output:

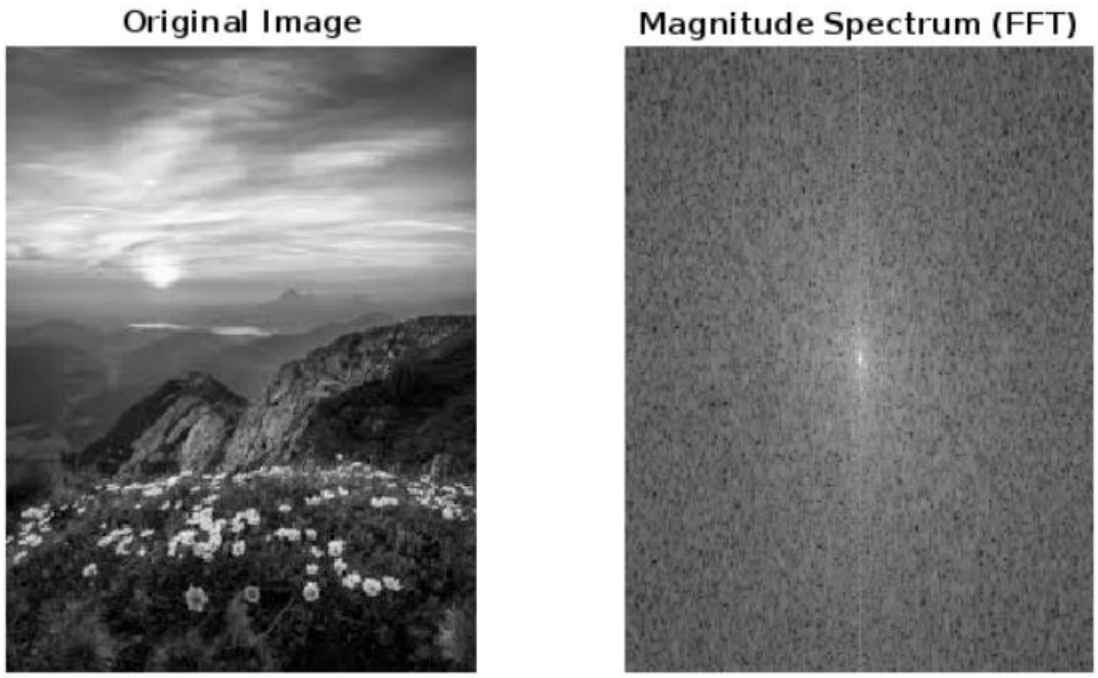

Figure 5.2: FFT of an image using MATLAB

Explanation:

1. The image is converted to double precision before applying the 2D Fast Fourier Transform using fft2().
2. The zero-frequency component is shifted to the center using fftshift().
3. The magnitude spectrum is computed using the logarithmic scale for better visualization.
4. The original image and its frequency representation are displayed side by side for comparison.

Experiment No: 05

Experiment Name: Homomorphic Transformation of an Image

Objectives:

1. The objective is to apply homomorphic filtering to enhance the contrast of an image by separating the illumination and reflectance components.
2. This method helps in improving the dynamic range of the image by enhancing details and reducing shadows.

Code-01: Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def homomorphic_filter(input_image, low=0.5, high=2.0, D0=30):
    input_image = np.float32(input_image) / 255.0
    log_image = np.log1p(input_image)
    fft_image = np.fft.fft2(log_image)
    fft_shifted = np.fft.fftshift(fft_image)
    rows, cols = input_image.shape
    crow, ccol = rows // 2 , cols // 2
    H = np.zeros((rows, cols), np.float32)
    for u in range(rows):
        for v in range(cols):
            D = np.sqrt((u - crow) ** 2 + (v - ccol) ** 2)
            H[u, v] = (high - low) * (1 - np.exp((-D ** 2) / (2 * (D0 ** 2)))) + low
    filtered_fft = fft_shifted * H
    ifft_shifted = np.fft.ifftshift(filtered_fft)
    inverse_fft = np.fft.ifft2(ifft_shifted)
    filtered_image = np.real(inverse_fft)
    homomorphic_image = np.expm1(filtered_image)
    homomorphic_image = cv2.normalize(homomorphic_image, None, 0, 1,
cv2.NORM_MINMAX)
    return homomorphic_image
input_image = cv2.imread('nature.jpeg', cv2.IMREAD_GRAYSCALE)
filtered_image = homomorphic_filter(input_image)
plt.figure(figsize=(12, 6))
```

```
plt.subplot(1, 2, 1)
plt.imshow(input_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(filtered_image, cmap='gray')
plt.title('Homomorphic Filtered Image')
plt.axis('off')
plt.show()
```

Output:

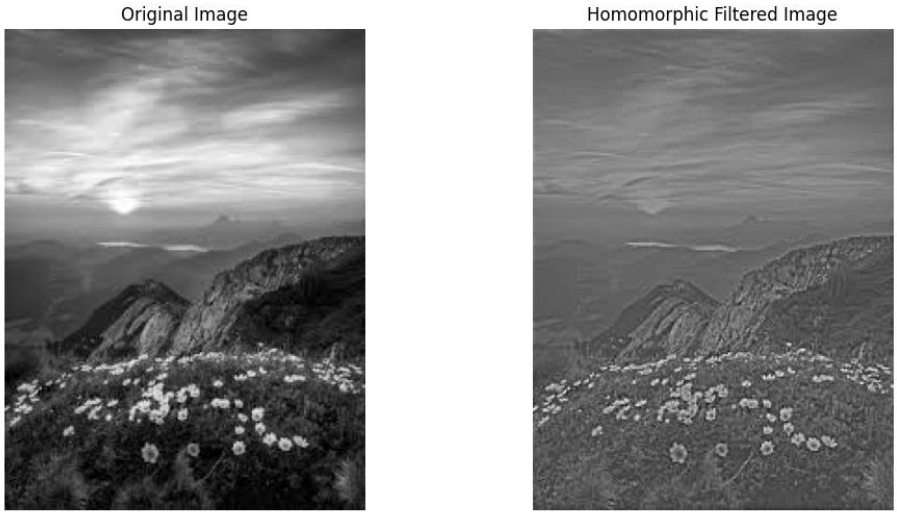

Figure 5.1: Homomorphic transform of an image using python code

Explanation:

1. The code first applies a logarithmic transformation to the image, which allows separating illumination and reflectance components.
2. Then, a high-pass filter is created and applied in the frequency domain using FFT.
3. The inverse FFT is used to transform the filtered image back into the spatial domain, and the exponential function is applied to reverse the log transformation.
4. The result is a contrast-enhanced image where shadowed regions are brightened, and overall dynamic range is improved.

Code-02: MATLAB

```
inputImage = imread('nature.jpeg');
if size(inputImage, 3) == 3
    inputImage = rgb2gray(inputImage);
end
inputImage = im2double(inputImage);
logImage = log(1 + inputImage);
fftImage = fft2(logImage);
fftImageShifted = fftshift(fftImage);
[M, N] = size(inputImage);
D0 = 30;
n = 2;
[X, Y] = meshgrid(1:N, 1:M);
centerX = ceil(N/2);
centerY = ceil(M/2);
D = sqrt((X - centerX).^2 + (Y - centerY).^2);
H = 1 - exp(-(D.^2 / (2 * D0^2)));
filteredFFT = fftImageShifted .* H;
filteredFFTShiftedBack = ifftshift(filteredFFT);
inverseFFT = ifft2(filteredFFTShiftedBack);
filteredImage = real(inverseFFT);
finalImage = exp(filteredImage) - 1;
figure;
subplot(1, 2, 1);
imshow(inputImage, []);
title('Original Image');
subplot(1, 2, 2);
imshow(finalImage, []);
title('Homomorphic Filtered Image');
```

Output:

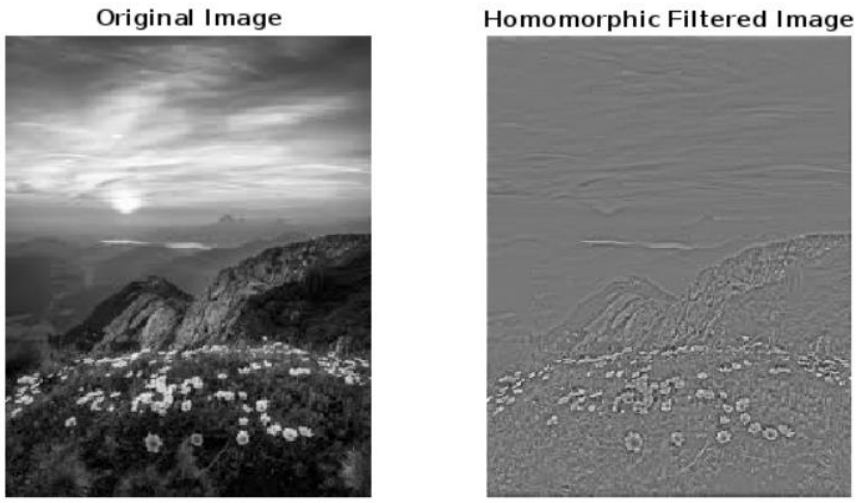

Figure 5.2: Homomorphic transform of an image using MATLAB

Explanation:

1. The image is first converted to double precision and a logarithmic transformation is applied to separate illumination and reflectance.
2. After performing FFT, a high-pass Gaussian filter is applied to suppress low-frequency components (illumination).
3. The filtered image is then converted back to the spatial domain using inverse FFT. The final result is an enhanced image with improved contrast, which is compared to the original image.