Assignment 1

Spring 2025

CSE-408 Digital Image Processing

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

Submitted to:

Engr. Mehran Ahmad

Date:

21st May 2025

Department of Computer Systems Engineering

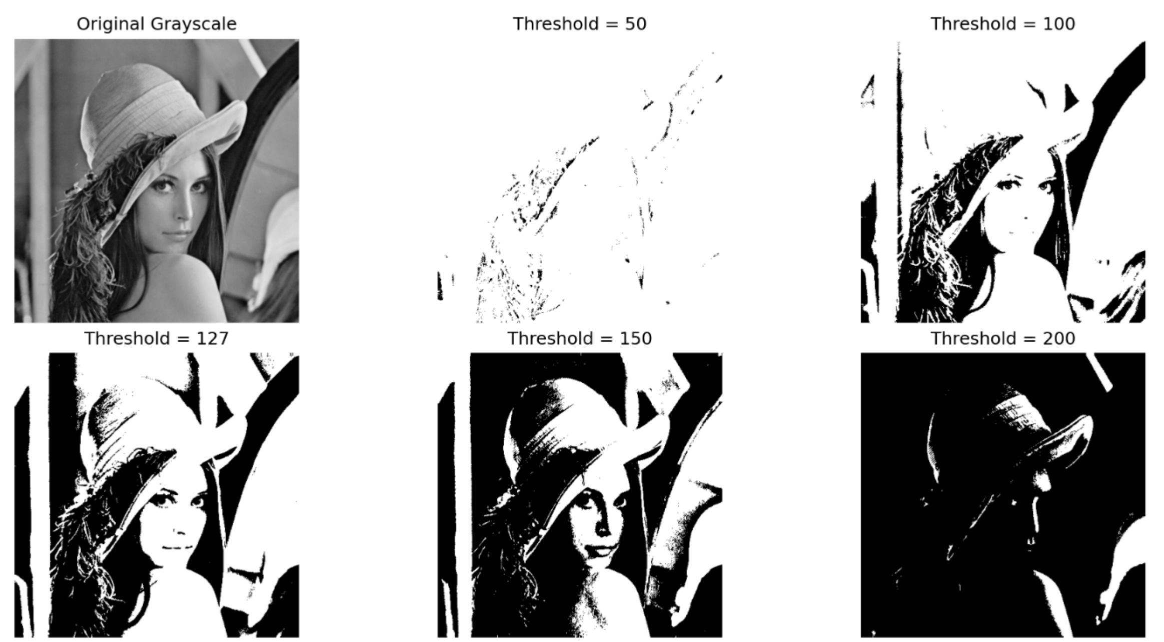
University of Engineering and Technology, Peshawar

Activity 1: Implement thresholding using MATLAB or Python. **Show the code and the output result** (original image and thresholded image).

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)

threshold_values = [50, 100, 127, 150, 200]
max_value = 255

plt.figure(figsize=(15, 8))
plt.subplot(2, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Grayscale')
plt.axis('off')

for i, t in enumerate(threshold_values):
    thresholded = np.where(image > t, max_value, 0).astype(np.uint8)
    plt.subplot(2, 3, i+2)
    plt.imshow(thresholded, c (variable) t: int
    plt.title(f'Threshold = {t}')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:



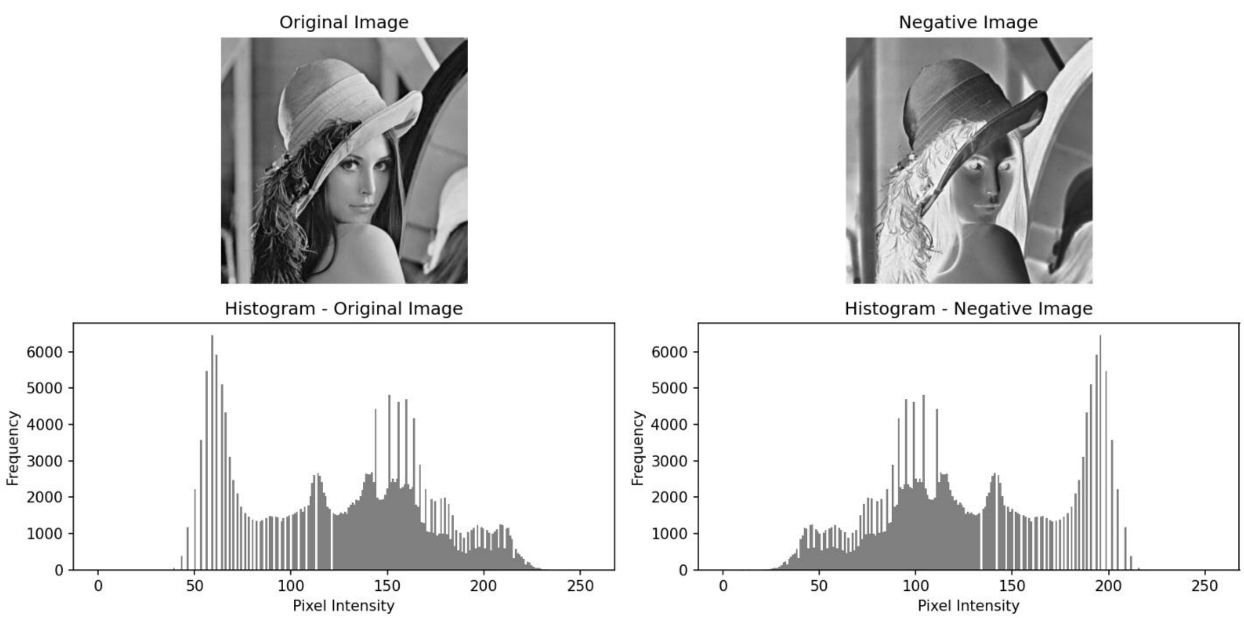
Analysis: Lower thresholds (e.g., 50) result in brighter images with poor edge contrast, while higher thresholds (e.g., 200) retain only the brightest regions. An optimal mid-value (e.g., 127) balances detail and contrast effectively for segmentation.

Activity 2: Write a MATLAB/Python script to perform negative transformation of an input image. **Show the code and the output result** (original and negative image, along with histograms).

Code:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)
L = 256  # For 8-bit images
negative = (L - 1) - image
plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(2, 2, 2)
plt.imshow(negative, cmap='gray')
plt.title('Negative Image')
plt.axis('off')
plt.subplot(2, 2, 3)
plt.hist(image.ravel(), bins=256, range=(0, 255), color='gray')
plt.title('Histogram - Original Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.subplot(2, 2, 4)
plt.hist(negative.ravel(), bins=256, range=(0, 255), color='gray')
plt.title('Histogram - Negative Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

Output:



Analysis: The negative image inverts pixel intensities, making dark areas bright and enhancing hidden details. Its histogram is a mirror of the original, confirming that each intensity III is transformed to 255−I255 - I255−I. This technique is effective for analyzing features in dark regions.

Activity 3: Implement a logarithmic transformation in MATLAB/Python. **Show the code and the output result** (original image and log-transformed image).

Code:

activity3.py > ...

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)
image_normalized = image / 255.0
c_values = [0.5, 1, 2, 5]
plt.figure(figsize=(15, 6))
plt.subplot(1, len(c_values)+1, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

for i, c in enumerate(c_values):
    log_transformed = c * np.log1p(image_normalized)
    log_transformed = cv2.normalize(log_transformed, None, 0, 255, cv2.NORM_MINMAX)
    log_transformed = log_transformed.astype(np.uint8)

    plt.subplot(1, len(c_values)+1, i+2)
    plt.imshow(log_transformed, cmap='gray')
    plt.title(f'c = {c}')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:

Analysis: Logarithmic transformation enhances low-intensity (dark) regions by compressing the dynamic range of pixel values. Varying the constant c controls the degree of enhancement — higher c values make dark areas brighter and more detailed. This is especially useful for images with shadowed or low-contrast features.

Activity 4:

Write MATLAB code to apply a power-law transformation on an image using different values of γ. **Show the code and the output result** (original image and transformed images for various γ values).

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('Lenna_(test_image).png', cv2.IMREAD_GRAYSCALE)
image = image / 255.0  # Normalize to [0, 1]

gamma_values = [0.4, 1.0, 2.0]
c = 1

plt.figure(figsize=(15, 5))
plt.subplot(1, len(gamma_values) + 1, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

for i, gamma in enumerate(gamma_values):
    gamma_corrected = c * np.power(image, gamma)
    plt.subplot(1, len(gamma_values) + 1, i + 2)
    plt.imshow(gamma_corrected, cmap='gray')
    plt.title(f'Gamma = {gamma}')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:

Analysis: Gamma correction is a nonlinear technique used to adjust image brightness based on a power-law relationship. When **gamma < 1** (e.g., 0.4), the image becomes brighter, enhancing details in dark regions. When **gamma > 1** (e.g., 2.0), the image becomes darker, which helps tone down overly bright areas. This method is especially useful in display systems and image preprocessing to match human visual perception.