Digital Image Processing Lab Report 2
CSE 4733

Abdullah Al Jubaer Gem
ID: 210041226
Section: 2B

December 8, 2025

1 Task 1: Nonlinear Transformation Functions

1.1 Problem Statement

Implement nonlinear transformation functions: Log Transformation and Power-law Transformation.

1.2 Solution Approach

Log transformation is used to make dark areas of an image brighter, helping to reveal details in shadows. It works by applying a logarithmic function to each pixel value, which stretches the lower intensity values more than the higher ones.

Power-law transformation, also called gamma correction, adjusts the overall brightness and contrast of an image. It raises each pixel value to a power (gamma), where gamma less than 1 makes the image brighter, and gamma greater than 1 makes it darker. This is useful for correcting display issues or enhancing specific parts of the image.

Intensity scaling is applied to ensure the transformed values fit within the 0-255 range for proper display, preventing clipping or loss of information.

1.3 Implementation

```python
def log_transform(gray_image):
    gray_image = np.asarray(gray_image)
    gray_image = gray_image.astype(np.float32)
    log_image = np.log(gray_image+1)
    min_val = np.min(log_image)
    max_val = np.max(log_image)
    log_image = (log_image-min_val)/(max_val-min_val)*255
    log_image = log_image.astype(np.uint8)
    log_image = Image.fromarray(log_image)
    return log_image

def power_transform(gray_image, gamma):
    gray_image = np.asarray(gray_image)
    gray_image = gray_image.astype(np.float32)
    power_image = np.power(gray_image, gamma)
    min_val = np.min(power_image)
    max_val = np.max(power_image)
    power_image = (power_image-min_val)/(max_val-min_val)*255
    power_image = power_image.astype(np.uint8)
    power_image = Image.fromarray(power_image)
    return power_image
```

1.4 Output

The following figures show the original image and the transformed images.

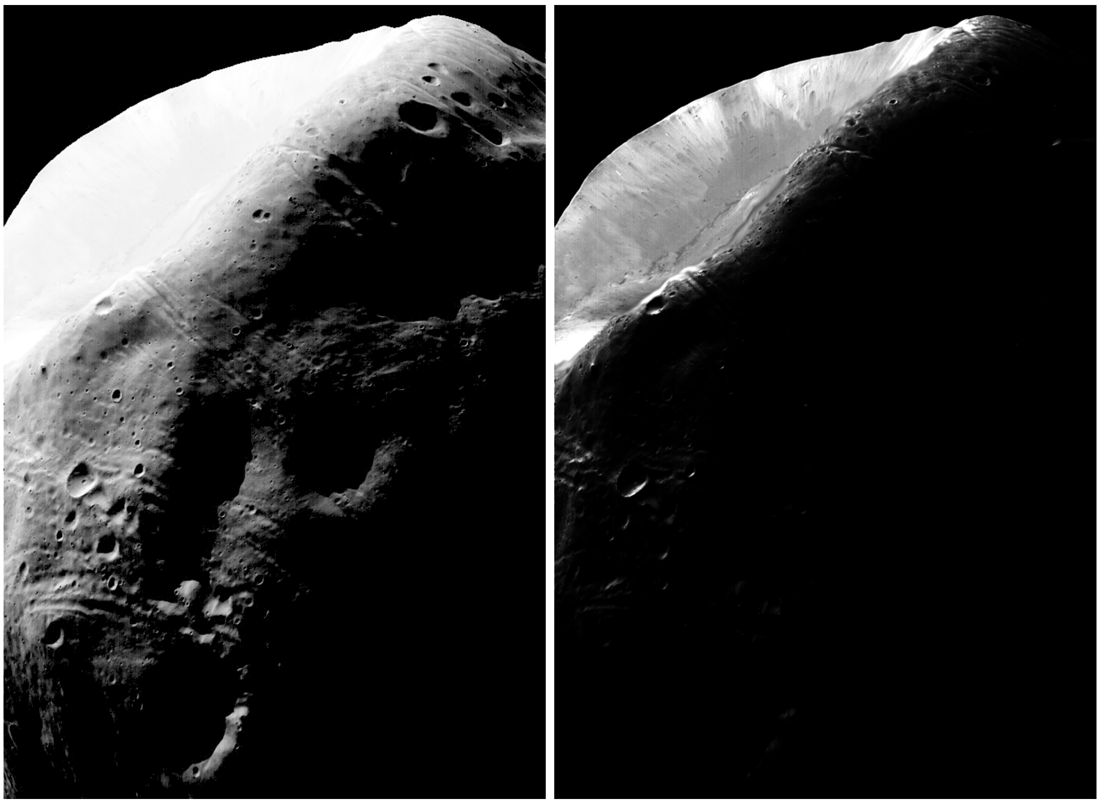

Figure 1: Log Transformed (left) and Power Transformed with $\gamma = 2$ (right)

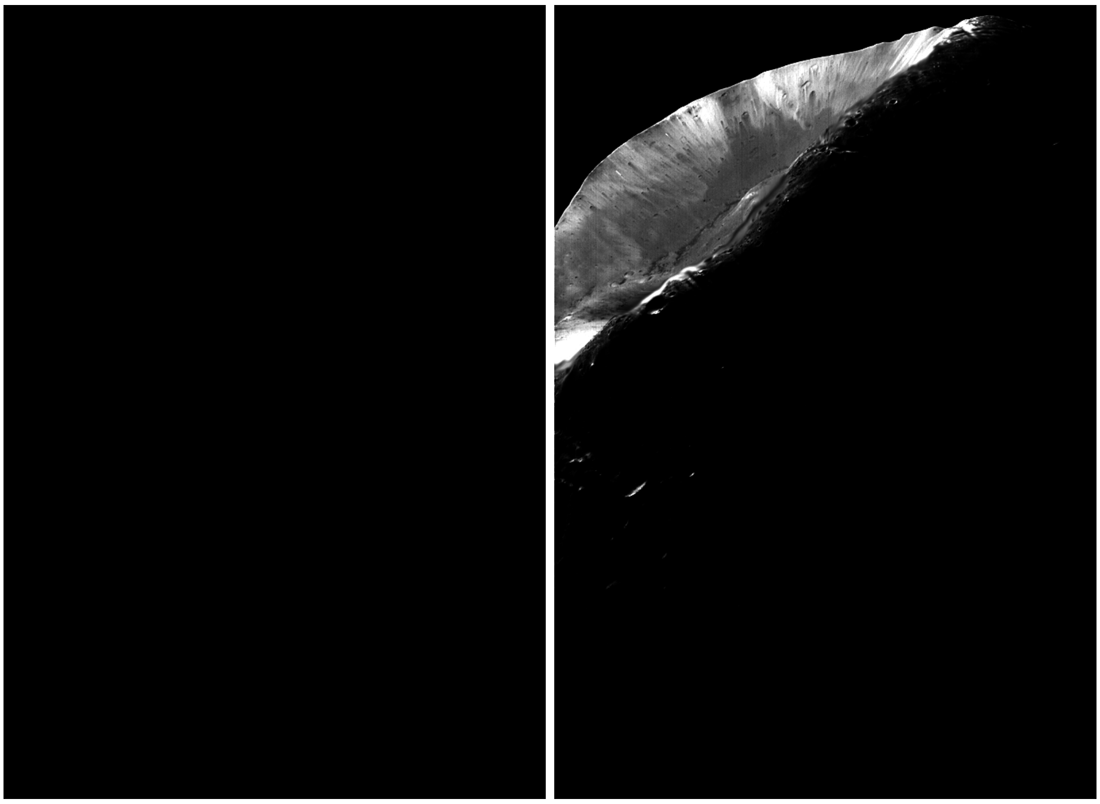

Figure 2: Power Transformed with $\gamma = -0.5$ (left) and $\gamma = 5$ (right)

1.5 Observation

The log-transformed image appears brighter in dark areas, making hidden details visible, which is useful for images like X-rays or astronomical photos where shadows contain important information. The power-law transformed images show adjusted brightness; with gamma <1, the image looks washed out with enhanced dark details, useful for low-light correction, while gamma >1 makes it darker with highlighted bright areas, helpful for high-dynamic-range images.

Log transformation is suitable for images with wide intensity ranges to compress bright values and expand dark ones. Power-law is ideal for gamma correction in displays or enhancing contrast in specific brightness levels. Intensity scaling ensures the results fit the display range, preventing data loss. An inverse log (using exponential) would expand bright values and compress dark ones, useful for images needing highlight enhancement.

2 Task 2: Global Histogram Equalization

2.1 Problem Statement

Implement Global Histogram Equalization on grayscale images.

2.2 Solution Approach

Global histogram equalization spreads out the pixel intensities of an image to use the full range of possible values, making the image more balanced. It works by calculating how often each intensity level appears (the histogram), then adjusting the pixel values so that the resulting image has a more uniform distribution of intensities. This is done using the cumulative distribution of the histogram to map old values to new ones, enhancing contrast across the entire image.

2.3 Implementation

```python
def global_HistEqual(image):
    image = np.asarray(image).astype(np.uint8)
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    hist, bins = np.histogram(image.flatten(), 256, [0, 256])
    image_size = image.shape[0] * image.shape[1]
    pdf = hist / image_size
    cdf = np.cumsum(pdf)
    cdf_upscaled = np.round(cdf * 255).astype(np.uint8)
    new_image = np.zeros_like(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            old_val = image[i][j]
            new_val = cdf_upscaled[old_val]
            new_image[i][j] = new_val
    plt.figure(figsize=(18, 12))
    plt.subplot(2, 3, 1)
    plt.title("Histogram")
    plt.xlabel("Pixel Value")
    plt.ylabel("Frequency")
    plt.bar(range(256), hist, width=1, color='black')
    plt.subplot(2, 3, 2)
    plt.title("PDF")
    plt.xlabel("Pixel Value")
    plt.ylabel("Probability")
    plt.bar(range(256), pdf, width=1, color='black')
    plt.subplot(2, 3, 3)
    plt.title("Original Image")
    plt.imshow(image, cmap='gray')
    plt.axis('off')
    plt.subplot(2, 3, 4)
    plt.title("CDF")
    plt.xlabel("Pixel Value")
    plt.ylabel("Cumulative Probability")
    plt.plot(cdf)
    plt.subplot(2, 3, 5)
    plt.title("Equalized Image")
    plt.imshow(new_image, cmap='gray')
    plt.axis('off')
    plt.tight_layout()
    plt.show()
    return hist, bins, pdf, cdf, new_image
```

2.4 Output

The histogram, PDF, CDF, original image, and equalized image are shown below.

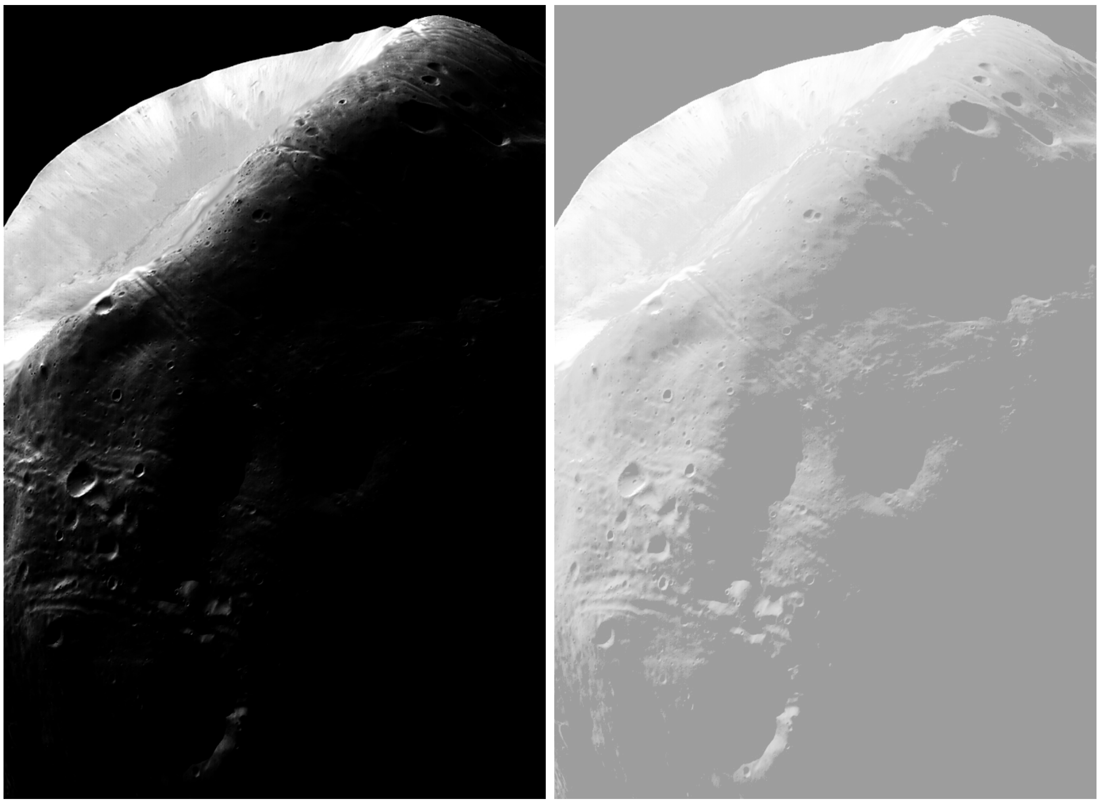

Figure 3: Original Image and Equalized Image

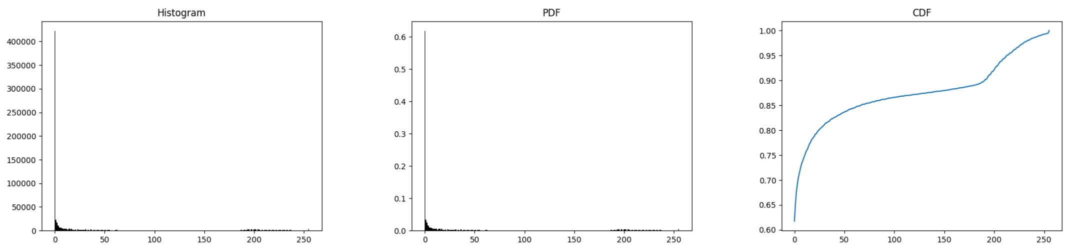

Figure 4: Histogram, PDF, and CDF

2.5 Observation

The equalized image looks more contrasted and balanced, with details in both dark and bright areas becoming clearer, as shown by the flattened histogram, PDF, and CDF. This happens because pixel intensities are redistributed to cover the full range, making the image appear sharper and more detailed.

Repeated equalization doesn't change the image further because it already has a uniform histogram, preventing over-enhancement. Global equalization is highly beneficial for low-contrast images like foggy photos, improving visibility, but detrimental for images with desired intensity ranges, like medical scans, where it might amplify noise or distort features. It's useful for general image enhancement in photography and computer vision.

3 Task 3: Local Histogram Equalization

3.1 Problem Statement

Implement Local Histogram Equalization (LHE) on grayscale images.

3.2 Solution Approach

Local histogram equalization improves contrast in different parts of an image by applying equalization to small, overlapping windows instead of the whole image. For each window, it calculates a local histogram and adjusts the pixel values within that window to enhance details. The windows slide across the image with a certain stride, and the results are averaged where they overlap. This method is better for images with varying lighting conditions, as it adapts to local changes rather than treating the entire image the same way.

3.3 Implementation

```python
def local_HistEqual(gray_image, size, stride):
    gray_image_np = np.asarray(gray_image)
    if len(gray_image_np.shape) == 3:
        gray_image = cv2.cvtColor(gray_image_np, cv2.COLOR_RGB2GRAY)
    else:
        gray_image = gray_image_np
    gray_image = gray_image.astype(np.uint8)
    height, width = gray_image.shape
    local_equalized_image = np.zeros_like(gray_image, dtype=np.float32)
    count_matrix = np.zeros_like(gray_image, dtype=np.float32)
    half = size // 2
    for i in range(0, height - size + 1, stride):
        for j in range(0, width - size + 1, stride):
            window = gray_image[i:i+size, j:j+size]
            hist, _ = np.histogram(window.flatten(), 256, [0, 256])
            pdf = hist / (size * size)
            cdf = np.cumsum(pdf)
            cdf_upscaled = np.round(255 * (cdf - cdf.min()) / (cdf.max() - cdf.min() + 1
    e-5)).astype(np.uint8)
            equalized_window = np.zeros_like(window)
            for x in range(size):
                for y in range(size):
                    old_val = window[x, y]
                    equalized_window[x, y] = cdf_upscaled[old_val]
            local_equalized_image[i:i+size, j:j+size] += equalized_window
            count_matrix[i:i+size, j:j+size] += 1
    count_matrix[count_matrix == 0] = 1
    local_equalized_image = local_equalized_image / count_matrix
    local_equalized_image = local_equalized_image.astype(np.uint8)
    hist_eq, bins_eq = np.histogram(local_equalized_image.flatten(), 256, [0, 256])
    image_size_eq = local_equalized_image.shape[0] * local_equalized_image.shape[1]
    pdf_eq = hist_eq / image_size_eq
    cdf_eq = np.cumsum(pdf_eq)
    return local_equalized_image, hist_eq, pdf_eq, cdf_eq
```

3.4 Output

The original and locally equalized images, along with histograms, are shown below.

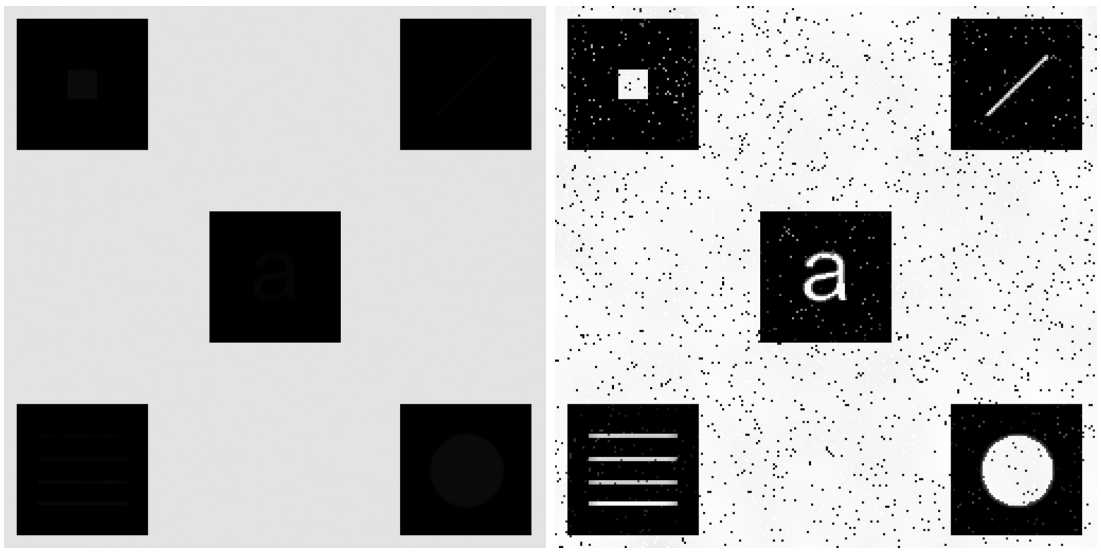

Figure 5: Original Image and Local Equalized Image

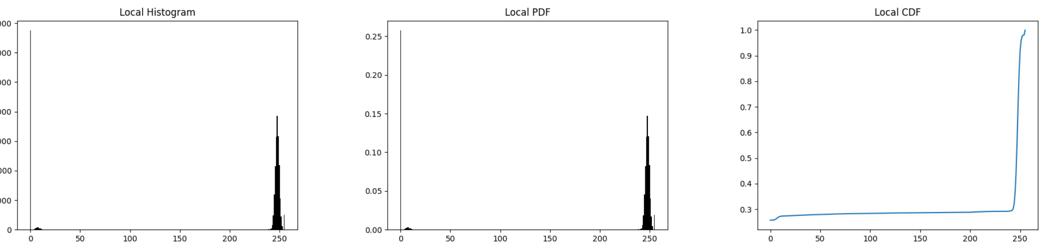

Figure 6: Local Histogram, PDF, and CDF

3.5 Observation

The local equalized image shows enhanced contrast in local regions, making textures and edges sharper, but may have blocky artifacts at boundaries due to windowing. The histogram, PDF, and CDF reflect the local adjustments, showing varied distributions.

Boundary effects occur because edge pixels lack full window data, leading to inconsistencies. Changing stride affects overlap; smaller stride smooths transitions but slows processing, larger stride speeds it up but may cause visible seams. LHE is superior to global equalization for non-uniform lighting, like spotlit scenes, as it adapts locally, preserving details in varying conditions, useful in medical imaging or surveillance for better local feature detection.