Lab Report 2 : Digital Image to Negative Image and Contrast Stretching Image with Histogram

Course Title: Digital Image Processing Lab

Course Code: CSE-406

Date of Performance: October 25, 2024
Date of Submission: September 8, 2024

Submitted to:

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University

Submitted by:

Class Roll	Exam Roll	Name
370	202182	Rubayed All Islam

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Experiment Name

a) Convert a digital image into a negative image.

b) Image enhancement using contrast stretching.

c) Image Histogram

Objective

a) To develop a program that can successfully convert a digital image into its negative counterpart, effectively reversing the intensity values of each pixel to create a complementary image.

b) To implement an image enhancement technique known as contrast stretching, which aims to increase the contrast between the darkest and brightest pixels in an image, thereby improving its visual quality and making details more discernible. This will be achieved by adjusting the intensity values of pixels within a specified range to expand the overall dynamic range of the image.

Experiment 1: Digital to Negative Image Conversion

Python Code:

Listing 1: digital_to_negative.py

```python
# Convert a digital image into a negative image

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load the image
image = Image.open('rubayed_image.jpg')

# Step 2: Convert image to a NumPy array
image_array = np.array(image)

# Step 3: Invert image values (convert to negative)
negative_array = 255 - image_array

# Step 4: Convert the negative array back to a PIL Image object (optional)
negative_image = Image.fromarray(negative_array.astype(np.uint8))

# Step 5: Display the original and negative images using Matplotlib
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_array)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Negative Image')

# Use grayscale colormap
plt.imshow(negative_array, cmap='gray')
plt.axis('off')

plt.show()

# Step 6: (Optional) Save the negative image
negative_image.save('negative_purple.jpg')
```

MATLAB Code:

Listing 2: digital_to_negative.m

```matlab
inputImagePath = '/rubayed_image.jpg';
image = imread(inputImagePath);
if isempty(image)
error('Failed to load image. Check the file path.');
end
if size(image, 3) == 3
image = rgb2gray(image);
end
negativeImage = 255 - image;
figure;
subplot(1, 2, 1);
imshow(imread(inputImagePath));
title('Original Image');
subplot(1, 2, 2);
imshow(negativeImage);
title('Digital Negative Image');
imwrite(negativeImage, 'negative_image.jpg');
disp('Image processing complete. Check the figure window and the saved file.');
```

Output:

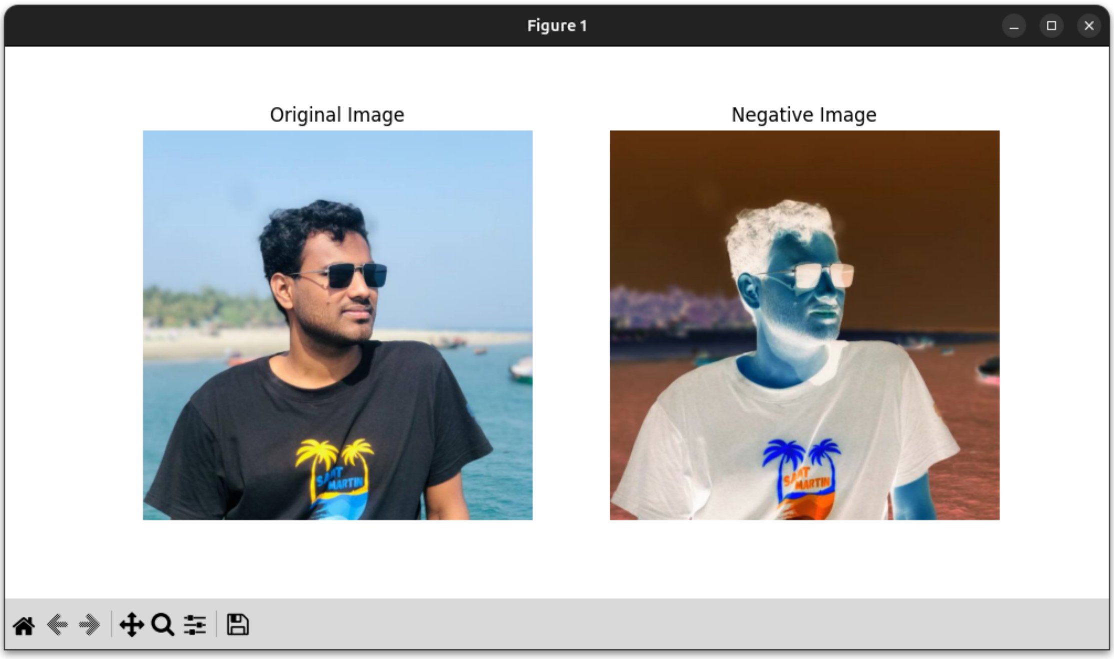

Figure 1: Conversion of Digital to Negative Image

Experiment 2: Enhancing Image Using Contrast Stretching

Python Code:

Listing 3: contrast_stretching.py

```python
# Image enhancement using contrast stretching and histograms of both original and stretched image

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load the color image
image = Image.open(’Butterfly.jpeg’)
image_array = np.array(image)

# Step 2: Split the image into R, G, B channels
r, g, b = image_array[:,:,0], image_array[:,:,1], image_array[:,:,2]

# Step 3: Apply contrast stretching to each channel
def contrast_stretch(channel):
    min_val = np.min(channel)
    max_val = np.max(channel)
    stretched_channel = (channel - min_val) / (max_val - min_val) * 255
    return stretched_channel.astype(np.uint8)

r_stretched = contrast_stretch(r)
g_stretched = contrast_stretch(g)
b_stretched = contrast_stretch(b)

# Print min and max values for debugging
print("Original R,G,B Channel - Min R: ", np.min(r), "Max R: ", np.max(r), "Min G: ", np.min(g), "Max G: ", np.max(g), "Min B: ", np.min(b), "Max B: ", np.max(b))
print("Stretched R Channel - Min: ", np.min(r_stretched), "Max: ", np.max(r_stretched))
print("Stretched G Channel - Min: ", np.min(g_stretched), "Max: ", np.max(g_stretched))
print("Stretched B Channel - Min: ", np.min(b_stretched), "Max: ", np.max(b_stretched))

# Step 4: Merge the stretched channels back together
stretched_image_array = np.stack((r_stretched, g_stretched, b_stretched), axis=2)
stretched_image = Image.fromarray(stretched_image_array)
```

```

# Step 5: Display the original and contrast-stretched images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_array)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Contrast-Stretched Image')
plt.imshow(stretched_image_array)
plt.axis('off')

plt.show()

# Step 6: Save the contrast-stretched image
stretched_image.save('contrast_stretched_color_purple.jpeg')

# Plot image and histogram
def plot_image_and_histogram(image_array, title):
    # Flatten the entire image array to get all pixel values
    all_pixels = image_array.ravel()

    plt.figure(figsize=(15, 6))

    # Display the image
    plt.subplot(1, 2, 1)
    plt.imshow(image_array)
    plt.title(title)
    plt.axis('off')

    # Plot the histogram of all pixel values
    plt.subplot(1, 2, 2)
    plt.hist(all_pixels, bins=256, color='gray', alpha=0.7)
    plt.title('Histogram')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

# Plot the original image and their histograms
plot_image_and_histogram(image_array, 'Original Image')

# Plot the contrast-stretched image and its histogram
plot_image_and_histogram(stretched_image_array,
   'Contrast-Stretched Image')
```

MATLAB Code:

Listing 4: contrast_stretching.m

```
i = imread('/Butterfly.jpeg');
s = imadjust(i, stretchlim(i,[0.05 0.95]),[]);
subplot(2,2,1), imshow(i), title('Original Image');
subplot(2,2,2), imshow(s), title('Contrast Stretched');
subplot(2,2,3), imhist(i), title('Histogram of Original Image');
subplot(2,2,4), imhist(s), title('Histogram of Stretched Image');
```

Output:

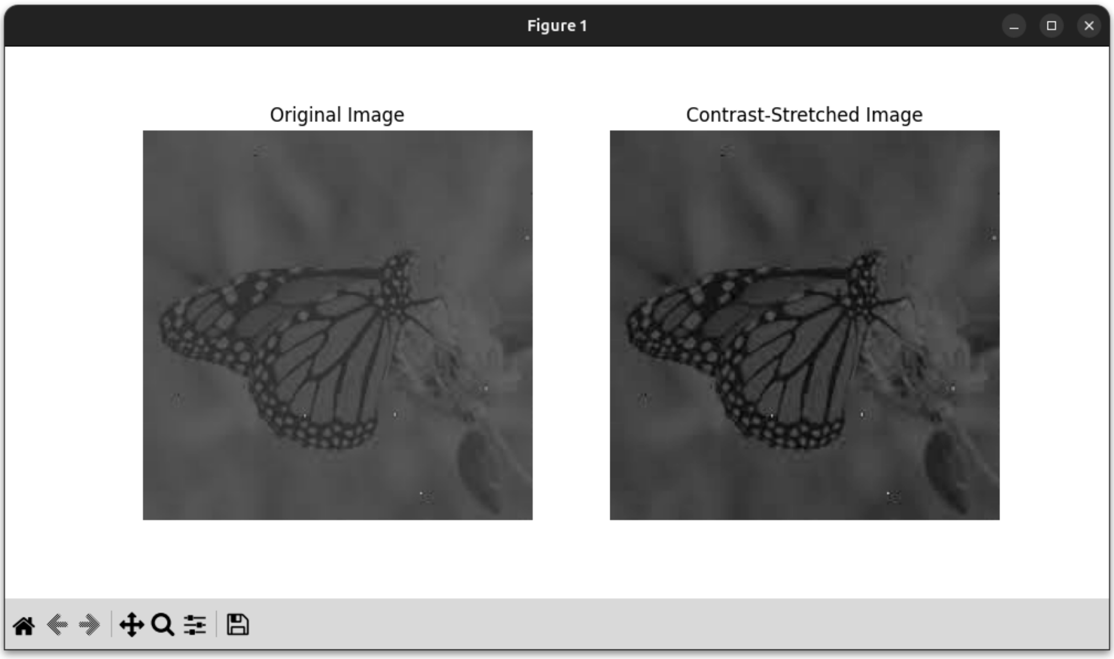

Figure 2: Enhancing Image Quality using Contrast Stretching

Experiment 3: Image Histogram of Poor Contrast, Good Contrast, Over-Exposed, Under-Exposed Images

MATLAB Code:

Listing 5: ImageHistogram.m

```
% Read the original image
img = imread(’FruitBasket.jpg’);

% Convert the image to grayscale
grayImg = rgb2gray(img);

% Create images with different contrast and exposure conditions
poorContrastImg = imadjust(grayImg, [0.3 0.7], []); % Poor contrast
goodContrastImg = imadjust(grayImg, stretchlim(grayImg), []); % Good contrast
overExposedImg = imadjust(grayImg, [], [0.5 1]); % Over-exposed
underExposedImg = imadjust(grayImg, [], [0 0.5]); % Under-exposed

% Define zoom region for histograms
xRange = [0 255]; % Pixel intensity range
yRange = [0 3000]; % Frequency range (adjust based on your data)

% Plot the histograms for each case
figure;

subplot(2,2,1);
imhist(poorContrastImg);
title(’Poor Contrast Image’);
xlabel(’Pixel Intensity’);
ylabel(’Frequency’);
xlim(xRange);
ylim(yRange);

subplot(2,2,2);
imhist(goodContrastImg);
title(’Good Contrast Image’);
xlabel(’Pixel Intensity’);
ylabel(’Frequency’);
xlim(xRange);
ylim(yRange);

subplot(2,2,3);
imhist(overExposedImg);
title(’Over-Exposed Image’);
xlabel(’Pixel Intensity’);
ylabel(’Frequency’);
```

```
xlim(xRange);
ylim(yRange);

subplot(2,2,4);
imhist(underExposedImg);
title('Under-Exposed Image');
xlabel('Pixel Intensity');
ylabel('Frequency');
xlim(xRange);
ylim(yRange);
```

Output:

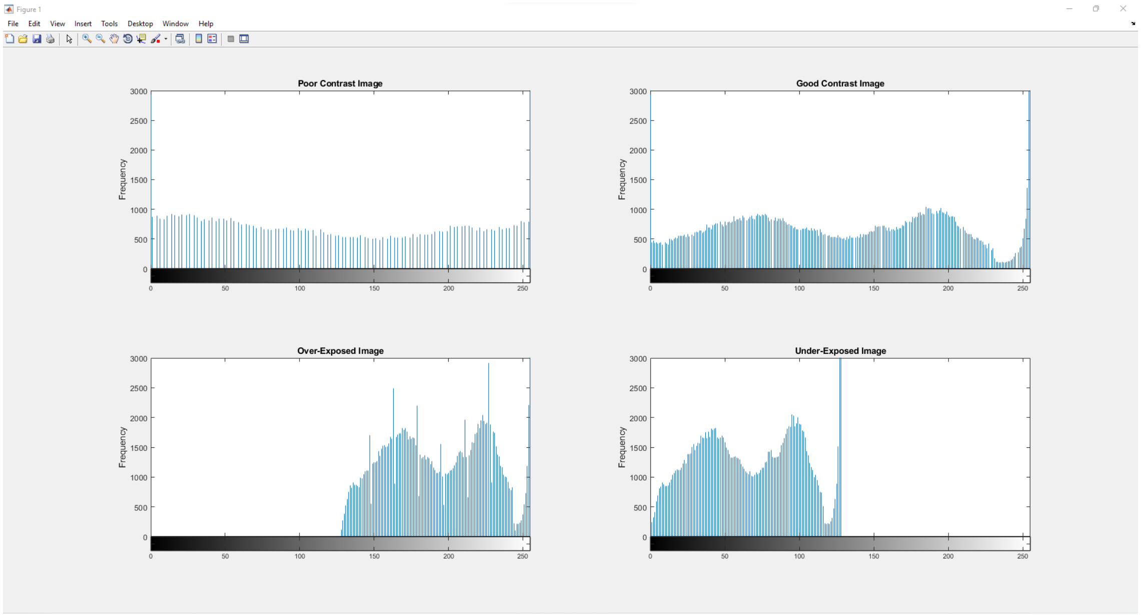

Figure 3: Enhancing Image Quality using Contrast Stretching

Results

1. **Successful Image Selection:** The experiment involves reading and displaying a color image using MATLAB. The code is designed to prompt the user to select an image file from their computer through a GUI. The supported file formats include JPEG, PNG, and BMP, which are common in digital imaging.

2. **User Interaction:** Once the user selects an image, the code reads the image file using the `imread` function and displays it using `imshow` in a new figure window. This process is fundamental in digital image processing, where the first step is typically to load and visualize the image data.

3. **Display of Image:** MATLAB's `uigetfile` function is particularly useful in creating user-friendly applications where manual file selection is needed. The experiment

demonstrates how to combine GUI elements with image processing functions, making it a practical introduction to more advanced image processing tasks.