Lab Report-03

Course title: Digital Image Processing Laboratory

Course code: CSE-406

4th Year 1st Semester Examination 2023

Date of Submission: 8 September 2024

Submitted to-

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Sl	Class Roll	Exam Roll	Name
01	408	202220	Sudipta Singha

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

Lab Report Title

Image Enhancement Using Histogram Equalization in Python

Introduction

Histogram equalization is a method for enhancing the contrast of an image by spreading out the intensity values across the entire range. This technique increases the global contrast of images, especially when their data is concentrated in a narrow range, improving visibility and highlighting details. Histogram equalization is widely used in fields such as medical imaging, remote sensing, and computer vision.

Python code

```
# histogram_equalization.py

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

def histogram_equalization(input_path, output_path):
    # Load the image and convert to grayscale
    img = Image.open(input_path).convert("L")
    img_array = np.array(img)

    # Flatten the image array and calculate histogram
    hist, bins = np.histogram(img_array.flatten(), 256, [0,
      ↪ 256])

    # Calculate cumulative distribution function (CDF)
    cdf = hist.cumsum()
    cdf_normalized = cdf * 255 / cdf[-1]  # Normalize the
      ↪ CDF to [0, 255]

    # Apply histogram equalization
    img_equalized = np.interp(img_array.flatten(),
      ↪ bins[:-1], cdf_normalized)
    img_equalized =
      ↪ img_equalized.reshape(img_array.shape).astype(np.uint8)
    equalized_img = Image.fromarray(img_equalized)

    # Save and display the images
    equalized_img.save(output_path)

    # Plot original and equalized images with histograms
    plt.figure(figsize=(12, 6))
    # Original Image and Histogram
    plt.subplot(2, 2, 1)
    plt.imshow(img, cmap="gray")
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(2, 2, 2)
    plt.hist(img_array.flatten(), 256, [0, 256],
       ↪ color='black')
    plt.title("Histogram of Original Image")

    # Equalized Image and Histogram
    plt.subplot(2, 2, 3)
    plt.imshow(equalized_img, cmap="gray")
    plt.title("Equalized Image")
    plt.axis("off")

    plt.subplot(2, 2, 4)
    plt.hist(img_equalized.flatten(), 256, [0, 256],
       ↪ color='black')
    plt.title("Histogram of Equalized Image")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"          # Replace with the
       ↪ path to your input image
    output_path = "equalized.jpg"      # Path for saving the
       ↪ equalized image
    histogram_equalization(input_path, output_path)
    print(f"Equalized image saved as {output_path}")
```

Input

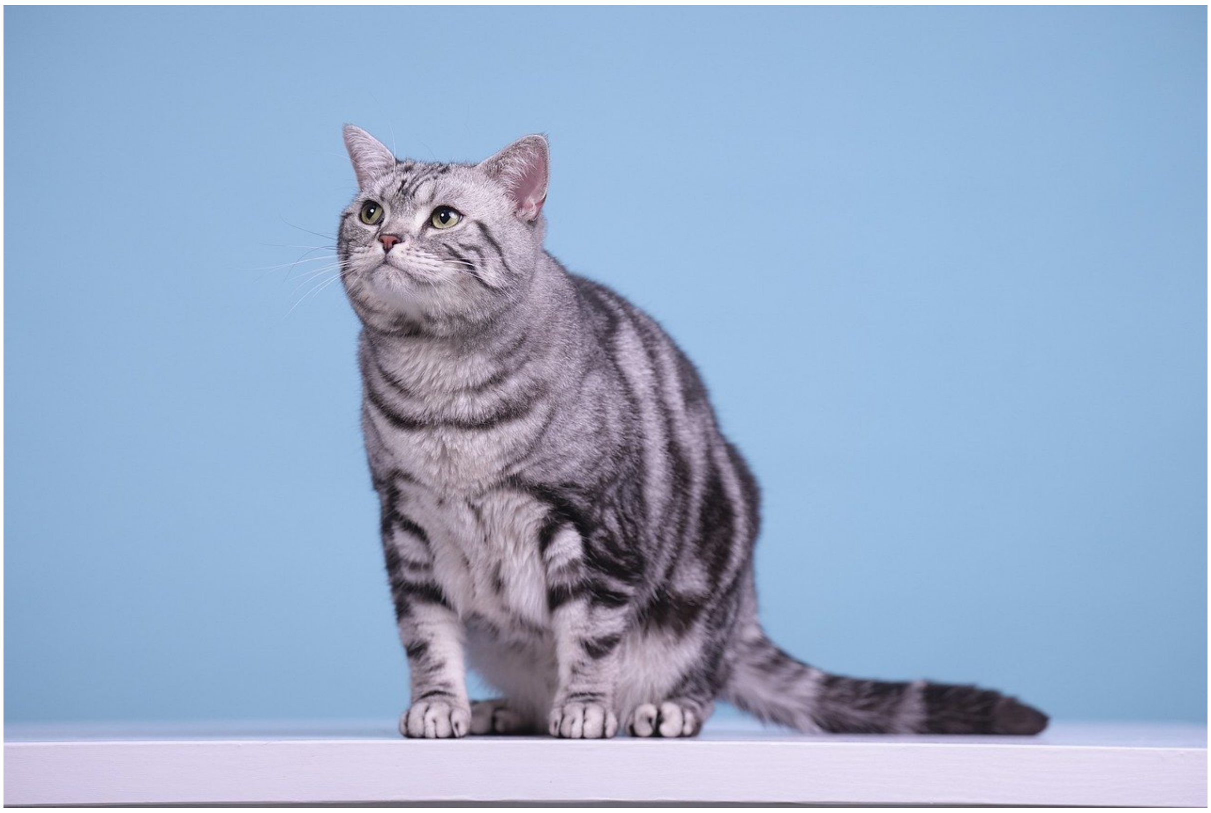
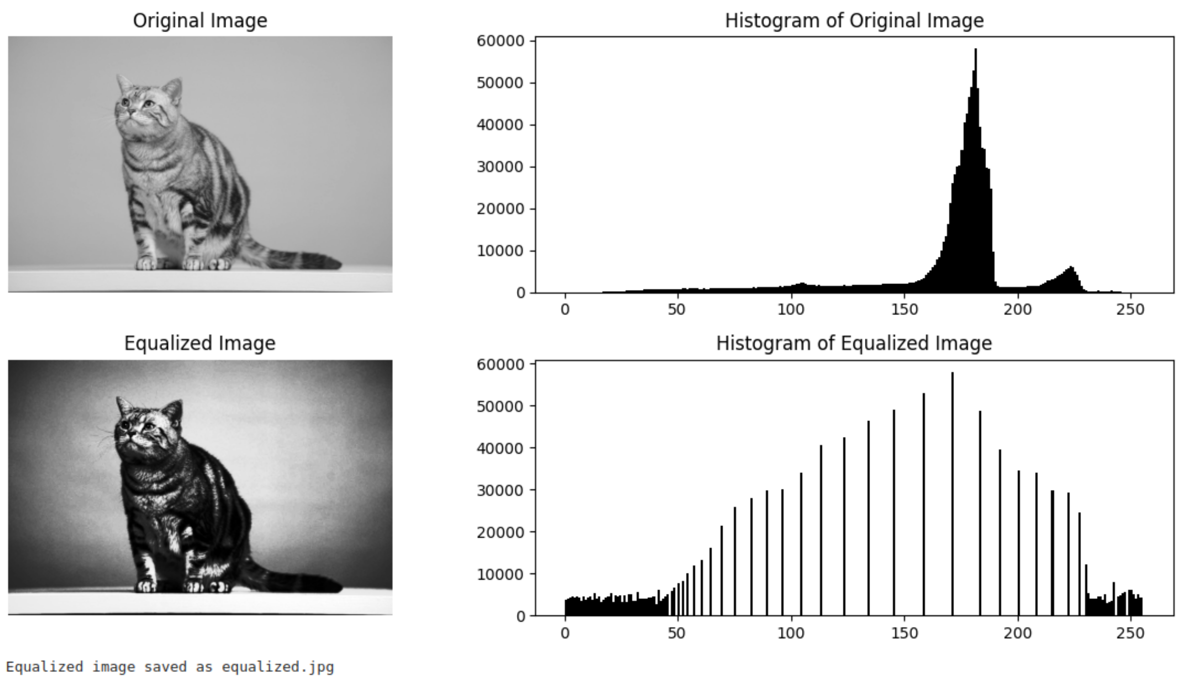


Figure 1: Input Image

Output



```
Equalized image saved as equalized.jpg
```

Figure 2: Output

Lab Report Title

Image Enhancement Using Histogram Specification in Python

Introduction

Histogram specification (or histogram matching) is a process of adjusting the pixel intensity distribution of an image to match a specified histogram. Unlike histogram equalization, which enhances contrast uniformly, histogram specification allows more control by transforming an image's histogram to resemble that of a reference image. This technique is often used in applications where visual consistency across different images is important, such as medical imaging or remote sensing.

Python code

```python
# histogram_specification.py

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

def match_histograms(source_img, reference_img):
    # Convert images to arrays
    source_array = np.array(source_img)
    reference_array = np.array(reference_img)

    # Flatten the image arrays and calculate histograms
    source_hist, bin_edges =
       ↪ np.histogram(source_array.flatten(), bins=256,
       ↪ range=(0, 256))
    reference_hist, _ =
       ↪ np.histogram(reference_array.flatten(), bins=256,
       ↪ range=(0, 256))

    # Calculate cumulative distribution functions (CDFs)
    source_cdf = source_hist.cumsum()
    source_cdf = source_cdf / source_cdf[-1]
    reference_cdf = reference_hist.cumsum()
    reference_cdf = reference_cdf / reference_cdf[-1]

    # Create a lookup table to match source CDF to reference
       ↪ CDF
    lookup_table = np.interp(source_cdf, reference_cdf,
       ↪ bin_edges[:-1])

    # Map the source image pixels using the lookup table
    matched_array = np.interp(source_array.flatten(),
       ↪ bin_edges[:-1],
       ↪ lookup_table).reshape(source_array.shape)
    matched_img =
       ↪ Image.fromarray(matched_array.astype(np.uint8))

    return matched_img

# Display and save images
def display_and_save_images(source_path, reference_path,
   ↪ output_path):
    # Load images
    source_img = Image.open(source_path).convert("L")
    reference_img = Image.open(reference_path).convert("L")

    # Perform histogram specification
    matched_img = match_histograms(source_img, reference_img)

    # Save the matched image
    matched_img.save(output_path)

    # Display the images
    plt.figure(figsize=(15, 5))

    # Original source image
    plt.subplot(1, 3, 1)
    plt.imshow(source_img, cmap="gray")
    plt.title("Source Image")
    plt.axis("off")

    # Reference image
    plt.subplot(1, 3, 2)
    plt.imshow(reference_img, cmap="gray")
    plt.title("Reference Image")
    plt.axis("off")

    # Matched image
    plt.subplot(1, 3, 3)
    plt.imshow(matched_img, cmap="gray")
    plt.title("Matched Image")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
```

```
source_path = "input.jpg"          # Replace with path to
    ↪ the source image
reference_path = "input2.jpg"     # Replace with path to
    ↪ the reference image
output_path = "matched_image.jpg"   # Output path for
    ↪ the matched image
display_and_save_images(source_path, reference_path,
    ↪ output_path)
print(f"Matched image saved as {output_path}")
```

Input

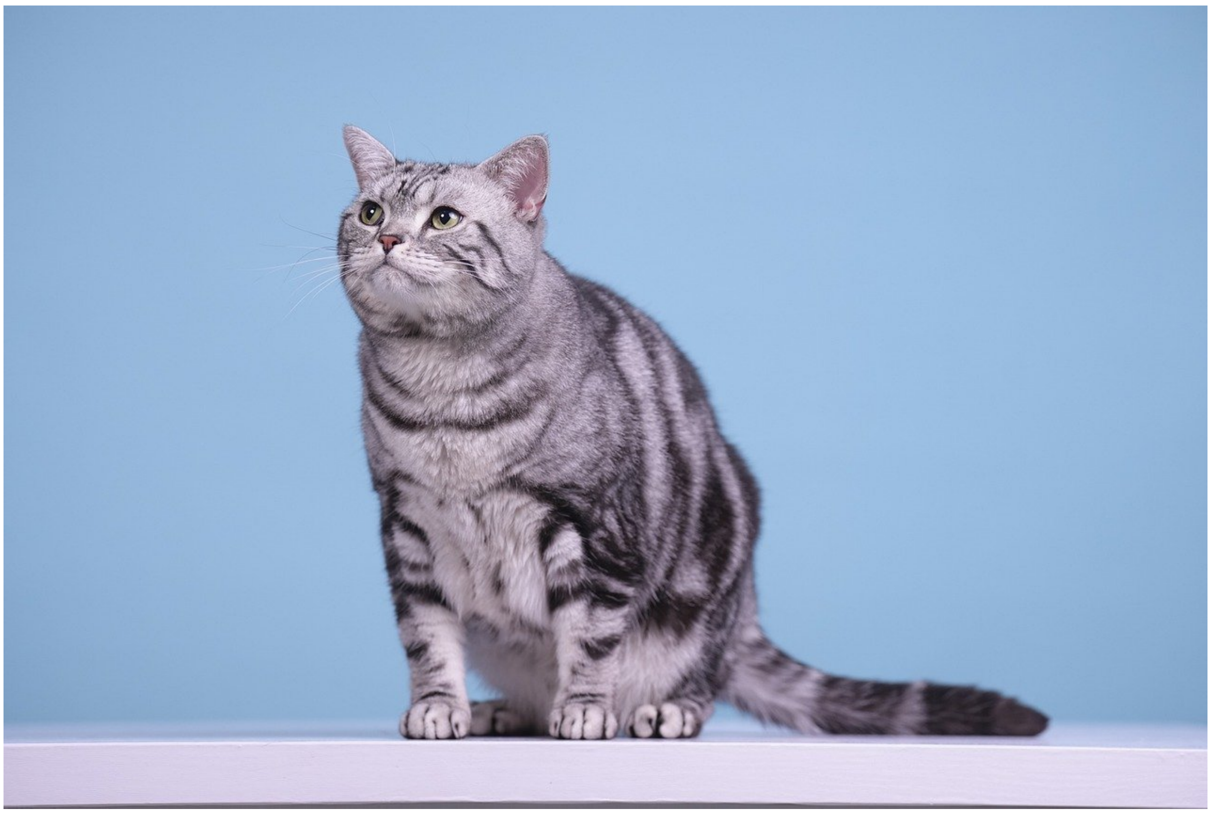
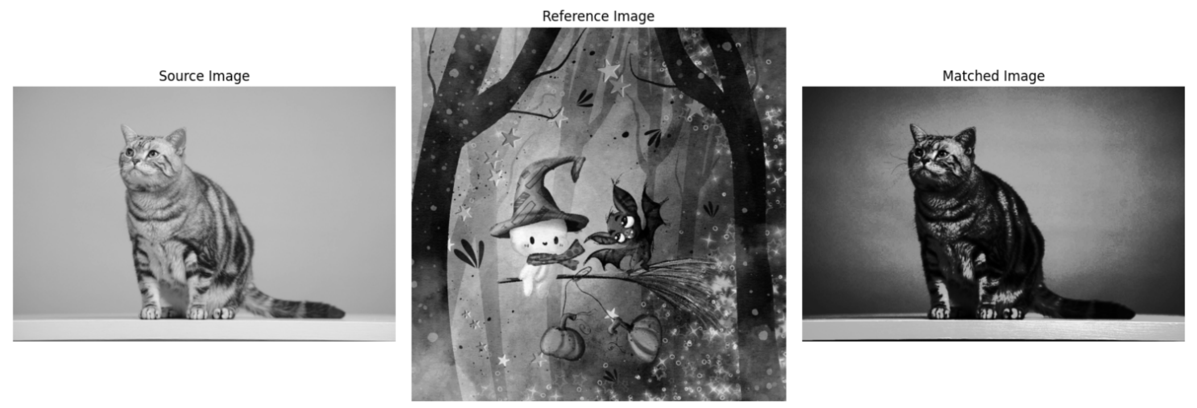


Figure 3:

Output



```
Matched image saved as matched_image.jpg
```

Figure 4: