Lab Report-06

Course title: Digital Image Processing Laboratory

Course code: CSE-406

4th Year 1st Semester Examination 2023

Date of Submission: 06 October 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

RGB to Grayscale Conversion in Python

Introduction

Converting an RGB image to grayscale is a common image processing task. Grayscale images are easier to process and analyze since they contain only intensity information, rather than color information. The conversion is typically done by applying a weighted sum of the red, green, and blue channels, with different weights for each color channel to account for human perception.

Python code

```
# rgb_to_grayscale.py

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

def rgb_to_grayscale(input_path, output_path):
    img = Image.open(input_path)  # Open the image
    img_array = np.array(img)

    # Convert RGB to Grayscale using the formula:
    # Y = 0.2989 * R + 0.5870 * G + 0.1140 * B
    grayscale_array = np.dot(img_array[...,:3], [0.2989,
      ↪ 0.5870, 0.1140])

    # Convert the grayscale array to an image
    grayscale_img =
      ↪ Image.fromarray(grayscale_array.astype(np.uint8))

    # Save the output image
    grayscale_img.save(output_path)

    # Display original and grayscale images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(grayscale_img, cmap="gray")
```

```python
    plt.title("Grayscale Image")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "grayscale_image.jpg"
    rgb_to_grayscale(input_path, output_path)
    print(f"Grayscale image saved as {output_path}")
```

Input

Figure 1:

Output

Figure 2:

Lab Report Title

RGB to CMY Conversion in Python

Introduction

Converting an RGB image to CMY (Cyan, Magenta, Yellow) is an important task in color image processing. CMY color model is often used in color printing as it is the inverse of the RGB color model. The conversion is done by subtracting each RGB component from 255 (for 8-bit color depth).

Python code

```python
# rgb_to_cmy.py

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

def rgb_to_cmy(input_path, output_path):
    img = Image.open(input_path)  # Open the image
    img_array = np.array(img)

    # Convert RGB to CMY using the formula:
    # C = 255 - R, M = 255 - G, Y = 255 - B
    cmy_array = 255 - img_array[...,:3]

    # Convert the CMY array to an image
    cmy_img = Image.fromarray(cmy_array.astype(np.uint8))

    # Save the output image
    cmy_img.save(output_path)

    # Display original and CMY images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cmy_img)
    plt.title("CMY Image")
    plt.axis("off")
```

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

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "cmy_image.jpg"
    rgb_to_cmy(input_path, output_path)
    print(f"CMY image saved as {output_path}")
```

Input

Figure 3:

Output

Figure 4:

Lab Report Title

RGB to HSI Conversion in Python

Introduction

The HSI (Hue, Saturation, Intensity) color model is an alternative to the RGB model, where HSI separates the chromatic content (Hue and Saturation) from the intensity (brightness). The conversion from RGB to HSI involves several steps, including extracting the Hue, Saturation, and Intensity components from the RGB values.

Python code

```
# rgb_to_hsi.py

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

def rgb_to_hsi(input_path, output_path):
    img = Image.open(input_path)  # Open the image
    img_array = np.array(img)

    # Normalize RGB values to [0, 1]
    img_array = img_array / 255.0

    # Extract R, G, B channels
    R = img_array[:,:,0]
    G = img_array[:,:,1]
    B = img_array[:,:,2]

    # Calculate intensity (I)
    I = (R + G + B) / 3

    # Calculate saturation (S)
    numerator = 2 * (np.minimum(np.minimum(R, G), B))
    denominator = R + G + B
    S = 1 - (3 * numerator / denominator)
    S[denominator == 0] = 0  # Handle division by zero

    # Calculate hue (H)
    numerator = 0.5 * ((R - G) + (R - B))
    denominator = np.sqrt((R - G) ** 2 + (R - B) * (G - B))
    theta = np.arccos(numerator / denominator)
    H = theta
    H[B > G] = 2 * np.pi - H[B > G]
```

```
    H = H / (2 * np.pi)  # Normalize to [0, 1]

    # Stack H, S, I to create HSI image
    hsi_img_array = np.stack([H, S, I], axis=-1)

    # Convert the HSI array to an image (in [0, 255] scale
       ↪ for each component)
    hsi_img = Image.fromarray((hsi_img_array *
       ↪ 255).astype(np.uint8))

    # Save the output image
    hsi_img.save(output_path)

    # Display original and HSI images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(hsi_img)
    plt.title("HSI Image")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "hsi_image.jpg"
    rgb_to_hsi(input_path, output_path)
    print(f"HSI image saved as {output_path}")
```

Input

Figure 5:

Output

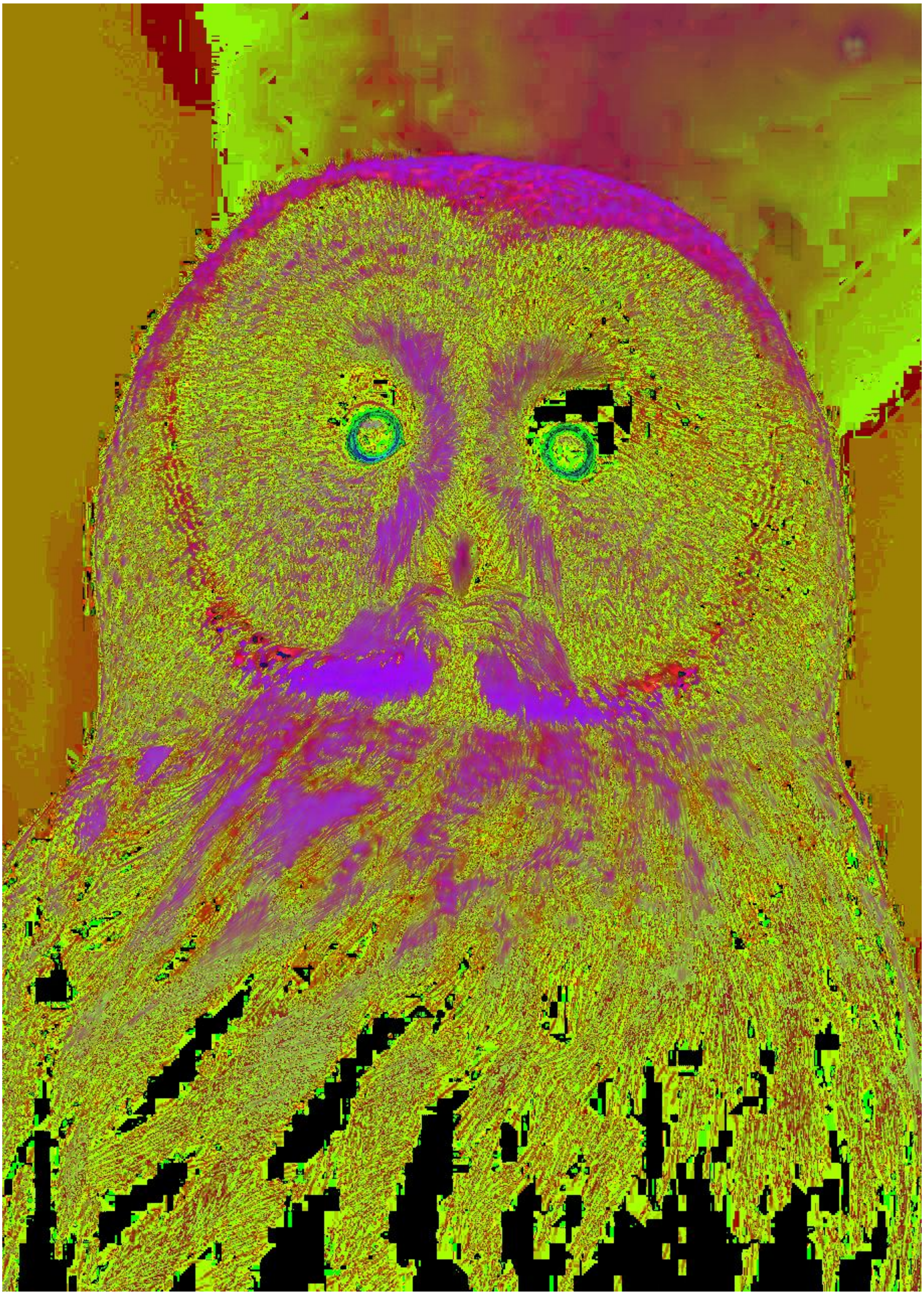

Figure 6:

Lab Report Title

RGB to YIQ Conversion in Python

Introduction

The YIQ color model is primarily used in television broadcasting, especially in NTSC color systems. It separates the image into three components: Y (luminance or brightness), I (in-phase chrominance), and Q (quadrature chrominance). The YIQ model is derived from the RGB model using a linear transformation to capture brightness and color information separately.

Python code

```
# rgb_to_yiq.py

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

def rgb_to_yiq(input_path, output_path):
    img = Image.open(input_path)  # Open the image
    img_array = np.array(img)

    # Normalize RGB values to [0, 1]
    img_array = img_array / 255.0

    # Define the RGB to YIQ transformation matrix
    transformation_matrix = np.array([[0.299,  0.587,
       ↪ 0.114],
                                      [-0.5957, -0.2744,
                                         ↪ 0.3213],
                                      [0.2113, -0.5226,
                                         ↪ 0.3116]])

    # Apply the transformation to RGB
    yiq_array = np.dot(img_array[...,:3],
       ↪ transformation_matrix.T)

    # Extract Y, I, Q channels
    Y = yiq_array[:,:,0]
    I = yiq_array[:,:,1]
    Q = yiq_array[:,:,2]

    # Stack Y, I, Q to create YIQ image
    yiq_img_array = np.stack([Y, I, Q], axis=-1)
```

```
    # Convert the YIQ array to an image (in [0, 255] scale
       ↪ for each component)
    yiq_img = Image.fromarray((yiq_img_array *
       ↪ 255).astype(np.uint8))

    # Save the output image
    yiq_img.save(output_path)

    # Display original and YIQ images
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(yiq_img)
    plt.title("YIQ Image")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path = "yiq_image.jpg"
    rgb_to_yiq(input_path, output_path)
    print(f"YIQ image saved as {output_path}")
```

Input

Figure 7:

Output

Figure 8:

Lab Report Title

RGB Channel Separation in Python

Introduction

RGB channel separation involves isolating the individual Red, Green, and Blue color channels from a color image. This can be useful for various image processing tasks such as analyzing individual color components or enhancing certain color channels.

Python code

```
# rgb_channel_separation.py

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

def rgb_channel_separation(input_path, output_path_red,
    ↪ output_path_green, output_path_blue):
    img = Image.open(input_path)  # Open the image
    img_array = np.array(img)

    # Separate the R, G, B channels
    R = img_array[:,:,0]
    G = img_array[:,:,1]
    B = img_array[:,:,2]

    # Create blank arrays for each channel to maintain the
        ↪ image shape
    R_channel = np.zeros_like(img_array)
    G_channel = np.zeros_like(img_array)
    B_channel = np.zeros_like(img_array)

    # Set the corresponding channels to the original values
    R_channel[:,:,0] = R
    G_channel[:,:,1] = G
    B_channel[:,:,2] = B

    # Convert the channel images to PIL format
    R_img = Image.fromarray(R_channel)
    G_img = Image.fromarray(G_channel)
    B_img = Image.fromarray(B_channel)

    # Save the output images
    R_img.save(output_path_red)
```

```
    G_img.save(output_path_green)
    B_img.save(output_path_blue)

    # Display the original image and the separated RGB
       ↪ channels
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 4, 1)
    plt.imshow(img)
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 4, 2)
    plt.imshow(R_img)
    plt.title("Red Channel")
    plt.axis("off")

    plt.subplot(1, 4, 3)
    plt.imshow(G_img)
    plt.title("Green Channel")
    plt.axis("off")

    plt.subplot(1, 4, 4)
    plt.imshow(B_img)
    plt.title("Blue Channel")
    plt.axis("off")

    plt.tight_layout()
    plt.show()

# Example usage
if __name__ == "__main__":
    input_path = "input.jpg"
    output_path_red = "red_channel.jpg"
    output_path_green = "green_channel.jpg"
    output_path_blue = "blue_channel.jpg"
    rgb_channel_separation(input_path, output_path_red,
       ↪ output_path_green, output_path_blue)
    print("RGB channels separated and saved.")
```

Input

Figure 9:

Output

Figure 10: