Assignment 11

Yehonatan Keypur

July 30, 2024

1 Question 1

Saturation Component (S)

The image is given as fully saturated, so the saturation component image will be constant with a value of 1 across the entire image. This means the S image will appear as a uniform white image.

Intensity Component (I)

All the squares are at their maximum value, so the intensity image will also be constant. Using the equation $I = \frac{R+G+B}{3}$, where the maximum value of any normalized pixel in the RGB image is 1, we get:

$$I = \frac{1+1+1}{3} = \frac{1}{3}$$

Therefore, the intensity image will be constant with a value of $\frac{1}{3}$ across the entire image. This will appear as a uniform gray image.

Hue Component (H)

The hue component image is the most visually interesting of the three. Recall that hue represents an angle, but its range is normalized to [0, 1]. In this normalized representation:

- Red corresponds to $H = 0$ (or 1)
- Green corresponds to $H = \frac{1}{3}$ (approximately 0.33)
- Blue corresponds to $H = \frac{2}{3}$ (approximately 0.67)

In the grayscale representation of the hue image:

- The red square will appear black ($H = 0$)
- The green square will appear as a lower-mid gray ($H = 0.33$)
- The blue square will appear as a lighter shade of gray than the green square ($H = 0.67$)

The important point is the relative brightness of these squares in the hue image: black for red, darker gray for green, and lighter gray for blue. This hue representation follows the counterclockwise direction around the color wheel, where 0 (black) represents red, $\frac{1}{3}$ (dark gray) represents green, and $\frac{2}{3}$ (lighter gray) represents blue.

In Summary

- **S image:** Uniform white (constant value of 1)
- **I image:** Uniform gray (constant value of $\frac{1}{3}$)
- **H image:** Three distinct gray levels - black for red, lower-mid gray for green, and a lighter gray for blue

2 Question 2

We're being asked to propose a modification to the error diffusion algorithm that would better preserve these edge areas. The goal is to maintain the sharp transitions between dark and light pixels, which define the edges in an image.

We need to alter the error diffusion process to be more "edge-aware". This involve changing how the error is calculated or distributed near edge regions, and incorporating edge detection into the process.

Algorithm Overview

The proposed modification to the error diffusion algorithm aims to preserve edges while maintaining the core functionality of error diffusion. This is achieved through the following steps:

1. Edge Detection
2. Adaptive Thresholding
3. Modified Error Distribution

Detailed Approach

Edge Detection
Prior to applying error diffusion, implement an edge detection step:

- Apply a simple edge detection method.
- Create a binary edge map $E(x, y)$ where:

$$E(x,y) = \begin{cases} 1 & \text{if pixel (x,y) is part of an edge} \\ 0 & \text{otherwise} \end{cases}$$

Adaptive Thresholding
Modify the thresholding step to be edge-aware:

$$T(x,y) = \begin{cases} T_{edge} & \text{if } E(x,y) = 1 \\ T_{standard} & \text{if } E(x,y) = 0 \end{cases}$$

Where T_{edge} is a more aggressive threshold that pushes edge pixels more decisively to black or white, and $T_{standard}$ is the regular threshold used in standard error diffusion.

Modified Error Distribution
Adjust the error distribution based on edge presence:

$$Error_{distributed}(x,y) = \begin{cases} \alpha \cdot Error(x,y) & \text{if } E(x,y) = 1 \\ Error(x,y) & \text{if } E(x,y) = 0 \end{cases}$$

Where α is a reduction factor (e.g., $\alpha = 0.5$) that lessens error distribution for edge pixels.

Example

Consider a small 4x4 grayscale image patch near an edge:

$$\begin{bmatrix} 120 & 125 & 130 & 135 \\ 115 & 200 & 210 & 140 \\ 110 & 205 & 215 & 145 \\ 105 & 110 & 115 & 120 \end{bmatrix}$$

Step 1: Edge Detection
After applying edge detection, we might get an edge map like:

$$E = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Step 2: Adaptive Thresholding
Let's say our standard threshold is 128, and our edge threshold is 180. For the pixel at (1,1) with value 200:

- Standard error diffusion would make this white (1) with an error of $200 - 255 = -55$
- Our method uses $T_{edge} = 180$, still making it white (1) but with less error: $200 - 255 = -55$

Step 3: Modified Error Distribution
For the pixel at (1,1), instead of distributing the full error of -55 to neighbors, we might distribute $\alpha \cdot (-55)$, where $\alpha = 0.5$:

- Error to distribute $= 0.5 \cdot (-55) = -27.5$

This reduced error distribution helps prevent the edge from being blurred.

Summary

This modified algorithm preserves edges by:

- Identifying edges before applying error diffusion

- Using more decisive thresholding at edge locations
- Reducing error propagation around edges

These modifications help maintain sharp transitions between dark and light pixels at edge locations, thereby preserving important image features while still benefiting from the error diffusion technique in non-edge regions.

3 Question 3

The complete code for the assignment is attached in a separate file named 'Ex11_Q3'. Please refer to the attached code file for the detailed implementation.

Below is the implementation of the error diffusion dithering algorithm used for image processing:

```python
# error_diffusion.py
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from numpy.typing import NDArray

def quantize(value, levels):
    return levels[np.argmin(np.abs(levels - value))]

def error_diffusion(image, m):
    levels = np.array([0, 64, 128, 192, 255])[:m]
    height, width = image.shape
    output = np.zeros_like(image, dtype=float)
    for y in range(height):
        for x in range(width):
            old_pixel = image[y, x]
            new_pixel = quantize(old_pixel, levels)
            output[y, x] = new_pixel
            error = old_pixel - new_pixel
            if x + 1 < width:
                output[y, x + 1] += error * 7 / 16
            if y + 1 < height:
                if x > 0:
                    output[y + 1, x - 1] += error * 3 / 16
                output[y + 1, x] += error * 5 / 16
                if x + 1 < width:
                    output[y + 1, x + 1] += error * 1 / 16
    return output.astype(np.uint8)

def process_image(image_path: str, m: int, save_to_dir: bool = False) -> None:
    original = np.array(Image.open(image_path).convert('L'))
    dithered = error_diffusion(original, m)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
    setup_axes(ax1, original, 'Original')
    setup_axes(ax2, dithered, f'Dithered ({m} levels)')
    plt.tight_layout()
    plt.show()
    if save_to_dir:
        image_name = image_path.split('/')[-1].split('.')[0]
        output_directory = 'output'
        output_path = f'{output_directory}/{image_name}_dithered_{m}_levels.png'
        Image.fromarray(dithered).save(output_path)

def setup_axes(ax, img, title):
    x_ticks = np.linspace(0, img.shape[1] - 1, 5, dtype=int)
    y_ticks = np.linspace(0, img.shape[0] - 1, 5, dtype=int)
    tick_values = [0, 64, 128, 192, 255]
    ax.imshow(img, cmap='gray')
    ax.set_title(title)
    ax.set_xticks(x_ticks)
    ax.set_yticks(y_ticks)
    ax.set_xticklabels(tick_values)
    ax.set_yticklabels(tick_values)
    ax.set_xlim(0, img.shape[1] - 1)
    ax.set_ylim(img.shape[0] - 1, 0)

from error_diffusion import process_image

# main.py
def main():
    image_path = 'input/tungsten_original.JPG'
    for m in range(2, 6):
        process_image(image_path, m)

if __name__ == "__main__":
    main()
```

4 Question 4

The saturation of a color's complement cannot be determined solely from the original color's saturation because different colors can have the same saturation value, and the complement's saturation depends on the maximum RGB value of the original color, while the original saturation depends on the minimum RGB value, thus requiring knowledge of the individual RGB components to calculate accurately.

Proof

Let's show that the saturation component of the complement of a color image cannot be computed from the saturation component of the input image alone.

Definitions

Let's begin by recalling the definition of saturation in the HSI color model:

$$S = 1 - \frac{3}{\mathrm{R} + \mathrm{G} + \mathrm{B}} \min(\mathrm{R}, \mathrm{G}, \mathrm{B})$$

where R, G, and B are the red, green, and blue components normalized to the range [0, 1].

Complement Image

For the complement image, the RGB values are:

$$\mathrm{R}' = 1 - \mathrm{R}$$
$$\mathrm{G}' = 1 - \mathrm{G}$$
$$\mathrm{B}' = 1 - \mathrm{B}$$

Saturation of Complement
Now, let's express the saturation of the complement image:

$$
\begin{aligned}
S' &= 1 - \frac{3}{\mathrm{R}' + \mathrm{G}' + \mathrm{B}'} \min(\mathrm{R}', \mathrm{G}', \mathrm{B}') \\
&= 1 - \frac{3}{(1-\mathrm{R}) + (1-\mathrm{G}) + (1-\mathrm{B})} \min(1-\mathrm{R}, 1-\mathrm{G}, 1-\mathrm{B}) \\
&= 1 - \frac{3}{3 - (\mathrm{R}+\mathrm{G}+\mathrm{B})} (1 - \max(\mathrm{R}, \mathrm{G}, \mathrm{B}))
\end{aligned}
$$

Comparison and Conclusion
Comparing the expressions for S and S', we observe:

- S depends on $\min(\mathrm{R}, \mathrm{G}, \mathrm{B})$
- S' depends on $\max(\mathrm{R}, \mathrm{G}, \mathrm{B})$

This difference is crucial. It demonstrates that knowing only the saturation S of the original image is not sufficient to determine S' of the complement image. Because the complement operation fundamentally alters the relationships between the RGB components in a non-linear way, the new saturation component S' cannot be derived solely from the original saturation S. The transformation depends on the individual RGB values and their specific changes during the complement calculation, making it impossible to compute S' from S alone.

Example

To further illustrate this, consider a fully saturated red and its complement:

- Fully saturated red: $(\mathrm{R} = 1, \mathrm{G} = 0, \mathrm{B} = 0)$, $S = 1$
- Its complement (cyan): $(\mathrm{R}' = 0, \mathrm{G}' = 1, \mathrm{B}' = 1)$, $S' = 1$

Both these colors are fully saturated ($S = S' = 1$), but they are complements of each other. This example shows that even knowing the original saturation ($S = 1$) is not enough to determine whether the complement will have the same or a different saturation.

Therefore, it is impossible to compute the saturation of the complement image using only the saturation of the input image. The individual RGB values are necessary for this calculation.

—

Thank you for reviewing this assignment.