Signals Analysis Project 2024

Image Compression & Decompression

Cairo University
Faculty Of Engineering
Computer Department

Name	ID
Ahmed Hamdy	9220032
Amir Kedis	9220166

Delivered to

Dr. Michael Melek
Eng. Sayed Kamel

Contents

Chapter 1

Extract Colors Channels

We extracted every channel separately and got the following results:

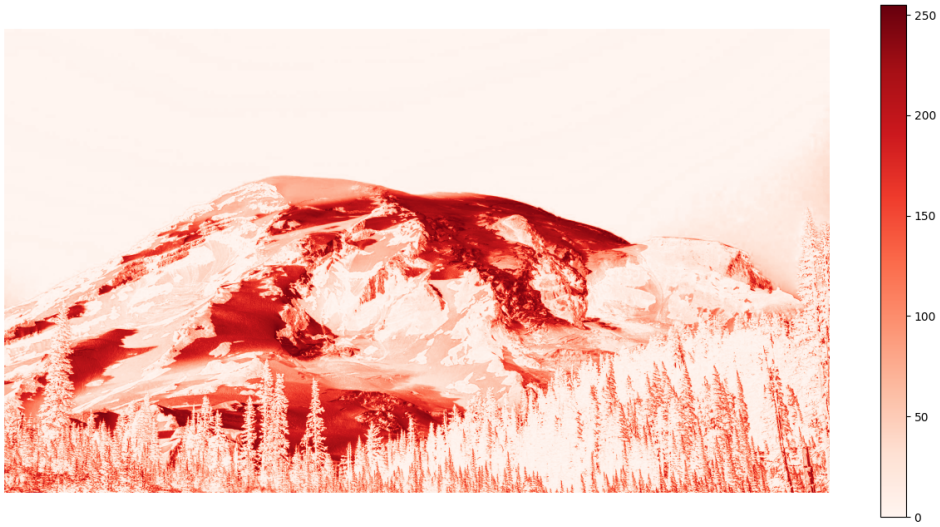

Figure 1.1: R component

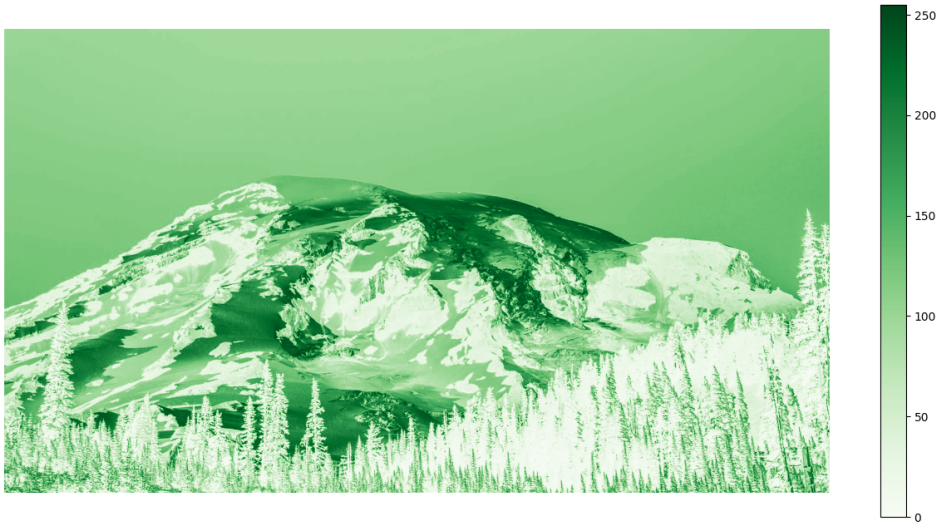

Figure 1.2: G component

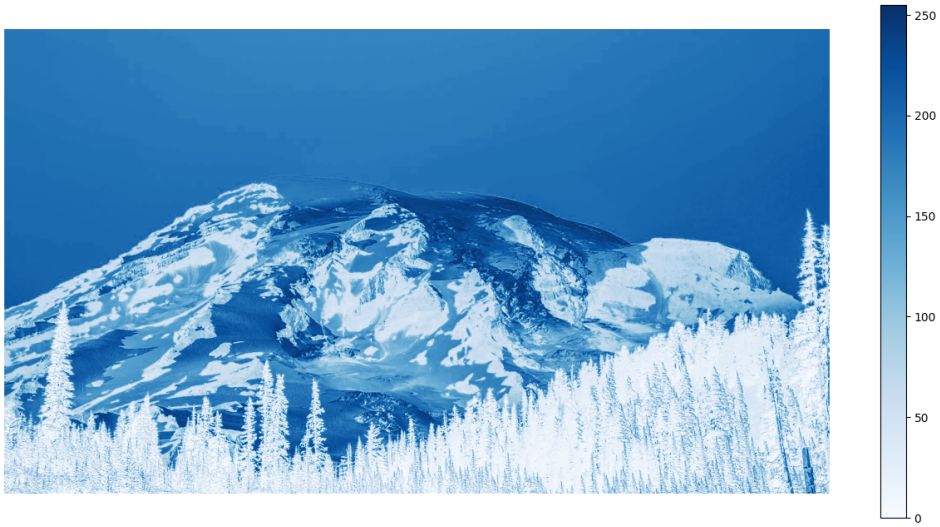

Figure 1.3: B component

Chapter 2

Image Processing

2.1 Compressing & Decompressing

2.1.1 2D DCT Formula

$$C(u,v) = a_u a_v \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} c(u,v) \cos\left(\frac{2x+1}{2M} u\pi\right) \cos\left(\frac{2y+1}{2N} v\pi\right) \tag{2.1}$$

where $0 \leq u \leq M-1$ and $0 \leq v \leq N-1$.

The formula for a_u is given by:

$$a_u = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{N}}, & \text{if } 1 \leq u \leq N-1 \end{cases} \tag{2.2}$$

Similar for a_v

2.1.2 Inverse 2D DCT Formula

$$c(u,v) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} a_u a_v C(u,v) \cos\left(\frac{2x+1}{2M} u\pi\right) \cos\left(\frac{2y+1}{2N} v\pi\right) \tag{2.3}$$

where $0 \leq u \leq M-1$ and $0 \leq v \leq N-1$.

The formula for a_u is given by:

$$a_u = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{N}}, & \text{if } 1 \leq u \leq N-1 \end{cases} \tag{2.4}$$

Similar for a_v

2.1.3 Results

We handle every color channel within the image matrix as a set of 8x8 pixels blocks. Applying the 2D Discrete Cosine Transform (2D DCT) on each block enables us to isolate the lower frequencies. Consequently, we can disregard the remaining frequencies and solely retain the significant ones for storage. Then, we perform the inverse 2D DCT to obtain the decompressed images for every value of m (1, 2, 3, 4) and got the following results:

Figure 2.1: Decompressed Image For m =1

Figure 2.2: Decompressed Image For m =2

Figure 2.3: Decompressed Image For m =3

Figure 2.4: Decompressed Image For m =4

2.1.4 PSNR

Table 2.1: PSNR

M value	PSNR in dB
m=1	19.86
m=2	21.91
m=3	23.77
m=4	25.94

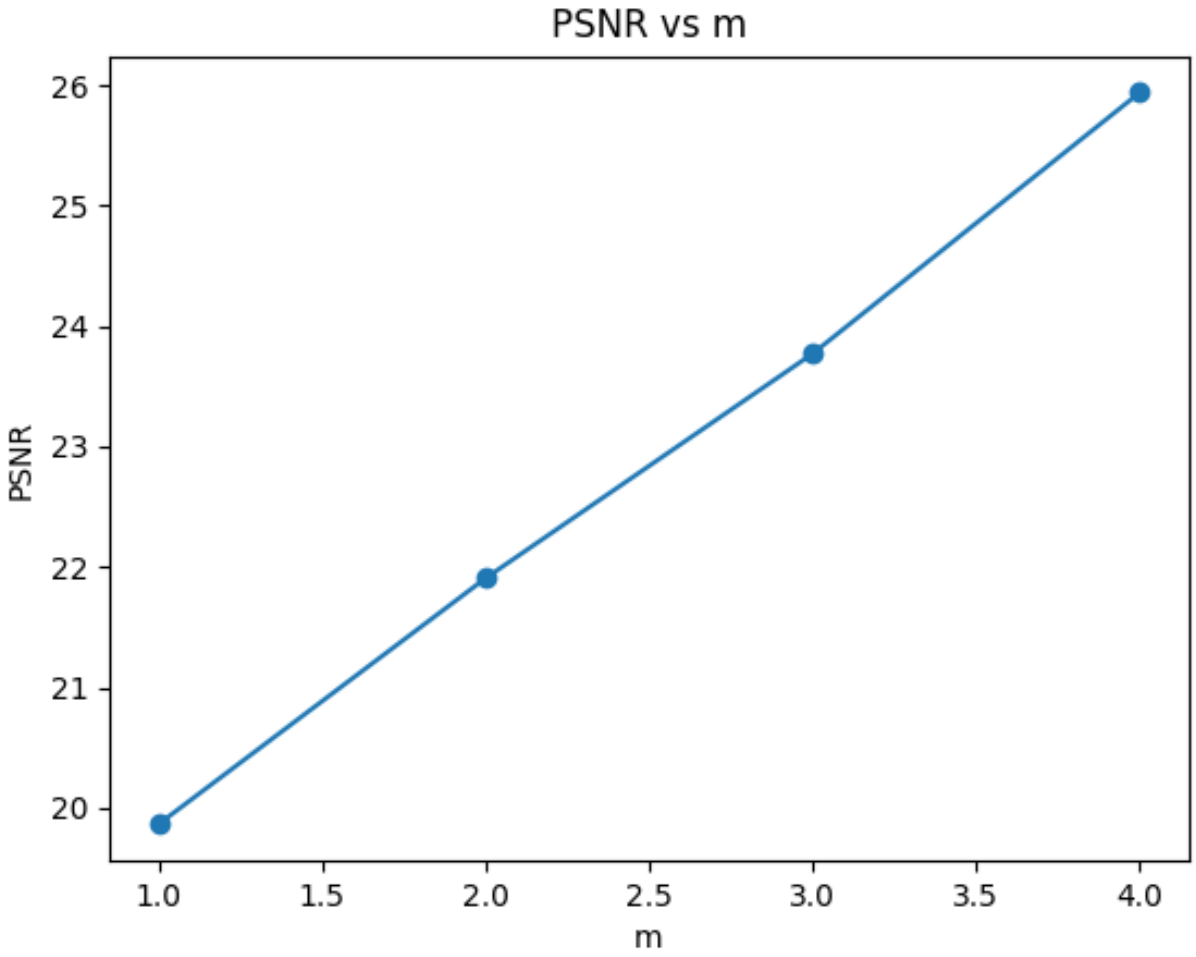

Figure 2.5: PSNR vs m

2.1.5 Compare Sizes

Table 2.2: Sizes Table

File	**Size In Bytes**	**Size Ratio**
Original image	3588902	1
Compressed image (m=1)	139871	0.03897
Compressed image (m=2)	454405	0.12661
Compressed image (m=3)	856028	0.23852
Compressed image (m=4)	1375337	0.38321

Appendix A

Appendix

A.1 Code

```python
#!/usr/bin/python3
"""Image Compression using DTC for signals course."""
# ============================================================================
# ============================== IMPORTS =====================================
# ============================================================================
import os  # file size comparison
import cv2  # image reading showing writing etc
import matplotlib.pyplot as plt  # plotting
import numpy as np  # array / image manipulation
from scipy.fft import dctn, idctn  # DCT and IDCT

# ============================================================================


# ============================================================================
# ============================== FUNCTIONS ===================================
# ============================================================================
def process_image(image, m):
    """
    Compresses and Decompresses an image by applying 2D DCT and retaining top-left coefficients.

    Args:
        image: The image as a NumPy array.
        m: The number of top-left coefficients to retain.

    Returns:
        The decompressed image as a NumPy array.
    """
    compressed_image = np.zeros_like(image)
    decompressed_image = np.zeros_like(image)
    # NOTE: we will compress each channel separately
    for channel in range(3):
        channel_image = image[:, :, channel]
        block_height, block_width = 8, 8
        for i in range(0, image.shape[0], block_height):
            for j in range(0, image.shape[1], block_width):
                block = channel_image[i : i + block_height, j : j + block_width]
                dct_block = dctn(block)

                dct_block[m:, :] = 0
                dct_block[:, m:] = 0

                # NOTE: this is the decompression part
                idct_block = idctn(dct_block)

                # remove overflow values
                idct_block[idct_block < 0] = 0
                idct_block[idct_block > 255] = 255

```

```
                compressed_image[
                    i : i + block_height, j : j + block_width, channel
                ] = dct_block
                decompressed_image[
                    i : i + block_height, j : j + block_width, channel
                ] = idct_block
    # NOTE: saving this to file system is unnecessary made just for simulation
    cv2.imwrite(f"compressed_m_{m}.png", compressed_image)
    cv2.imwrite(f"decompressed_m_{m}.png", decompressed_image)
    return decompressed_image


def compare_sizes(m):
    """
    Compare the sizes of the original and compressed images.

    Args:
        m: The number of top-left coefficients to retain.
    """
    original_size = os.path.getsize("image1.png")
    compressed_size = os.path.getsize(f"compressed_m_{m}.png")
    print(
        f"Original Size: {original_size} byte Compressed Size: {compressed_size} byte image size
    ratio: {compressed_size/original_size}"
    )


def calculate_psnr(original, decompressed):
    """
    Calculate Peak Signal-to-Noise Ratio (PSNR).

    Args:
        original: The original image as a NumPy array.
        decompressed: The decompressed image as a NumPy array.

    Returns:
        The PSNR value in dB.
    """
    return cv2.PSNR(original, decompressed)


# ================================================================================


def main():
    """Do Main Function."""
    # 1. image reading
    image = cv2.imread("image1.png")
    cv2.imshow("Original", image)
    waitForEnter()

    # 2. Extract each color components
    # FIXME: this is a silly way to extract channels with reserving zeros
    red = np.zeros_like(image)
    green = np.zeros_like(image)
    blue = np.zeros_like(image)
    red[:, :, 2] = image[:, :, 2]
    green[:, :, 1] = image[:, :, 1]
    blue[:, :, 0] = image[:, :, 0]
    cv2.imshow("Red", red)
    cv2.imshow("Green", green)
    cv2.imshow("Blue", blue)
    waitForEnter()

    # 3. Loop Compress -> compare size -> decompress -> calc PSNR
    psnr_values = []
    for m in range(1, 5):
        print(f"Compressing with m={m}")
        decompressed_image = process_image(image.copy(), m)
        compare_sizes(m)
        psnr = calculate_psnr(image, decompressed_image)
```

```
        psnr_values.append(psnr)
        print(f"PSNR (m={m}): {psnr:.2f} dB")
        cv2.imshow(f"Decompressed m={m}", decompressed_image)
        waitForEnter()

    # 4. plot PSNR values
    plt.plot(range(1, 5), psnr_values, marker="o")
    plt.xlabel("m")
    plt.ylabel("PSNR")
    plt.title("PSNR vs m")
    plt.show()


def waitForEnter():
    """Wait for enter key to be pressed."""
    while True:
        if cv2.waitKey(1) == 13:
            break
    cv2.destroyAllWindows()


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

Listing A.1: project code