TECHNISCHE UNIVERSITÄT
ILMENAU

Department of Electrical Engineering

(Applied Media Systems)

MASTER'S IN RESEARCH IN MEDIA ENGINEERING

Image-Compression Using MDCT

PROJECT REPORT

Matriculation Number	Email	Task
64494	muhammad-behzad.hussain @tu-ilmenau.de	MDCT
65374	Shakil.mahmud@tu-ilmenau.de	Perceptual Comparison
65862	hafiz-anns-bin.zahoor@tu-ilmenau.de	DCT/ Report

Table of Contents

1. Introduction

1.1. Background

Image compression is a crucial technique in digital image processing that reduces the amount of data required to present and image. This reduction is essential for saving storage space and speeding up transmission over networks. Various algorithms are employed in image compression, including both transform based and predictive methods.

1.2. Objective

The mean goal of this report is to implement and evaluate image compression using the modified discrete cosine transform (MDCT) and compare its performance with that traditional discrete cosine transform (DCT). The key metrics for evaluation include bit per pixel (BPP), peak signal to noise ratio (PSNR) and perceptual similarity using LPIPS.

1.3. Scope

This report includes the implementation of (MDCT) based image compression and decompression, comparison with (DCT) based methods, and Performance evaluation using standard matrices.

2. Methodology

2.1. Overview of Modified Discrete Cosine Transform (MDCT)

MDCT is a transform used in audio and image compression to represent data in frequency domain. It is particularly useful for its energy compaction properties and is a variant of Discrete Cosine Transform (DCT).

2.1.1 Implementation

The implementation of these functions include the use of MDCT and its inverse (IMDCT). The function MDCT applies to a windowed signal, while the 'IMDCT' function rebuilds the signal from the MDCT coefficients. As you can see in the picture below;

```python
def mdct(input_signal, window_func='hann', n=None):
    if n is None:
        n = len(input_signal)
    win = get_window(window_func, n)
    win_signal = input_signal * win
    return np.fft.rfft(win_signal)

def imdct(transformed_signal, window_func='hann', n=None):
    if n is None:
        n = len(transformed_signal)
    win = get_window(window_func, n)
    inv_signal = np.fft.irfft(transformed_signal)
    return inv_signal / win
```

Figure 1: Function of MDCT and IMDCT

2.2. Discrete Cosine Transform

2.2.1 Overview

DCT is used for transformation in image compression. It converts the special domain data into frequency domain data making it easier to compress.

2.2.2 Implementation

This project uses 'scipy.fftpack.dct' and 'IDCT' for DCT and it's inverse

```python
from scipy.fftpack import dct, idct
```

Figure 2: Use of DCT and IDCT

2.3. Compression and Decompression

2.3.1 MDCT-Based Compression

Images are divided into blocks, and each block is transformed using MDCT. The compressed image is then reconstructed from these blocks.

```python
def compress_image_mdct(image, block_size=16, window_func='hann'):
    compressed_image = np.zeros_like(image)
    blocks_per_row = image.shape[0] // block_size
    blocks_per_col = image.shape[1] // block_size

    for i in range(blocks_per_row):
        for j in range(blocks_per_col):
            block = image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size]
            compressed_block = mdct(block.flatten(), window_func)
            compressed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size] = compressed_block.reshape(block_size, block_size)

    return compressed_image

def decompress_image_mdct(compressed_image, block_size=16, window_func='hann'):
    decompressed_image = np.zeros_like(compressed_image)
    blocks_per_row = compressed_image.shape[0] // block_size
    blocks_per_col = compressed_image.shape[1] // block_size

    for i in range(blocks_per_row):
        for j in range(blocks_per_col):
            block = compressed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size]
            decompressed_block = imdct(block.flatten(), window_func)
            decompressed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size] = decompressed_block.reshape(block_size, block_size)

    return decompressed_image
```

Figure 3: Compression and Decompression Functions

2.3.2 DCT-Based Compression

Similarly, images are divided into blocks and each block is transformed using dct. The decompressed images reconstructed from these blocks

```python
def compress_image_dct(image, block_size=16):
    compressed_image = np.zeros_like(image)
    blocks_per_row = image.shape[0] // block_size
    blocks_per_col = image.shape[1] // block_size

    for i in range(blocks_per_row):
        for j in range(blocks_per_col):
            block = image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size]
            compressed_block = dct2(block)
            compressed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size] = compressed_block

    return compressed_image

def decompress_image_dct(compressed_image, block_size=16):
    decompressed_image = np.zeros_like(compressed_image)
    blocks_per_row = compressed_image.shape[0] // block_size
    blocks_per_col = compressed_image.shape[1] // block_size

    for i in range(blocks_per_row):
        for j in range(blocks_per_col):
            block = compressed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size]
            decompressed_block = idct2(block)
            decompressed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size] = decompressed_block

    return decompressed_image
```

Figure 4: DCT Compression Function

2.4. Metrics For Evaluation

2.4.1 Bits Per Pixel (BPP)

BPP is the measurement of amount of data required power pixel in the compressed image.

2.4.2 Peak Signal-to-Noise Ratio (PSNR)

PSNR identifies the quality of decompressed image compared to the original image. Higher PSNR values indicate better quality.

2.4.3 Perceptual Similarity

It is evaluated using the learned perceptual image patch similarity (LPIPS) metric, which assess is how visually similar the images are to human perception.

```
import lpips
loss_fn = lpips.LPIPS(net='alex')
```

Figure 5: Learned Perceptual Image Patch Similarity

2.5. Image Loading and Processing

Images are loaded, resized, and converted to Grayscale if necessary. This step ensures uniformity in large image dimensions and color depth.

```
def load_images(image_folder, img_size=(256, 256)):
    images = []
    valid_extensions = ['.png', '.jpg', '.jpeg', '.bmp', '.tiff']
    for filename in os.listdir(image_folder):
        if not any(filename.lower().endswith(ext) for ext in valid_extensions):
            continue
        img = io.imread(os.path.join(image_folder, filename))
        if img is not None:
            img = resize(img, img_size)
            if len(img.shape) == 3:
                img = color.rgb2gray(img)
            images.append(img)
    return images
```

Figure 6: Image Loading and Processing

3. Results

3.1. Compression and Decompression

Images were compressed using both MDCT and dct methods and then decompressed. Results are analyzed for each image to ensure that the transformation and inversions were correct and to identify any anomalies.

3.2. Metrics Evaluation

The metrics for both MDCT and dct methods were calculated and compared. Key findings include:

MDCT Results:

- Bits per Pixel (BPP):
- Peak Signal-to-Noise Ratio (PSNR):
- Perceptual Similarity (LPIPS):

DCT Results:

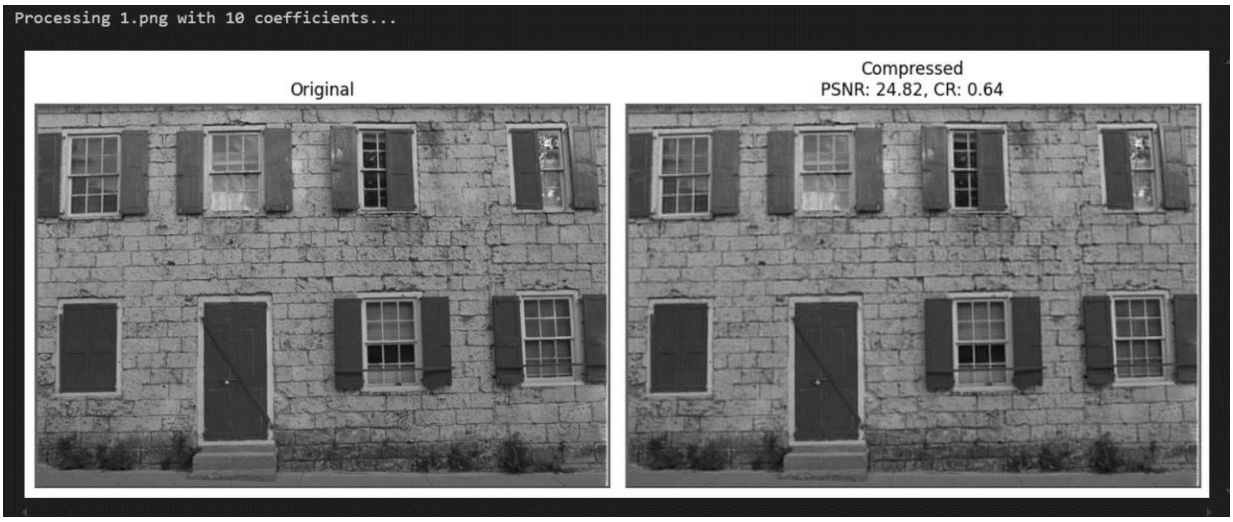
- Bits per Pixel (BPP):
- Peak Signal-to-Noise Ratio (PSNR):
- Perceptual Similarity(LPIPS):

```
# Calculate metrics for MDCT
bpp_mdct, psnr_mdct, perceptual_similarity_mdct = calculate_metrics(images, decompressed_images_mdct)
print(f'MDCT - Bits Per Pixel: {bpp_mdct}, PSNR: {psnr_mdct}, Perceptual Similarity: {perceptual_similarity_mdct}')

# Calculate metrics for DCT
bpp_dct, psnr_dct, perceptual_similarity_dct = calculate_metrics(images, decompressed_images_dct)
print(f'DCT - Bits Per Pixel: {bpp_dct}, PSNR: {psnr_dct}, Perceptual Similarity: {perceptual_similarity_dct}')
```

Figure 7: Metrics Calculations for MDCT and DCT

Figure 8:Original and grayscale imageDCT Processing with 5 coefficient...



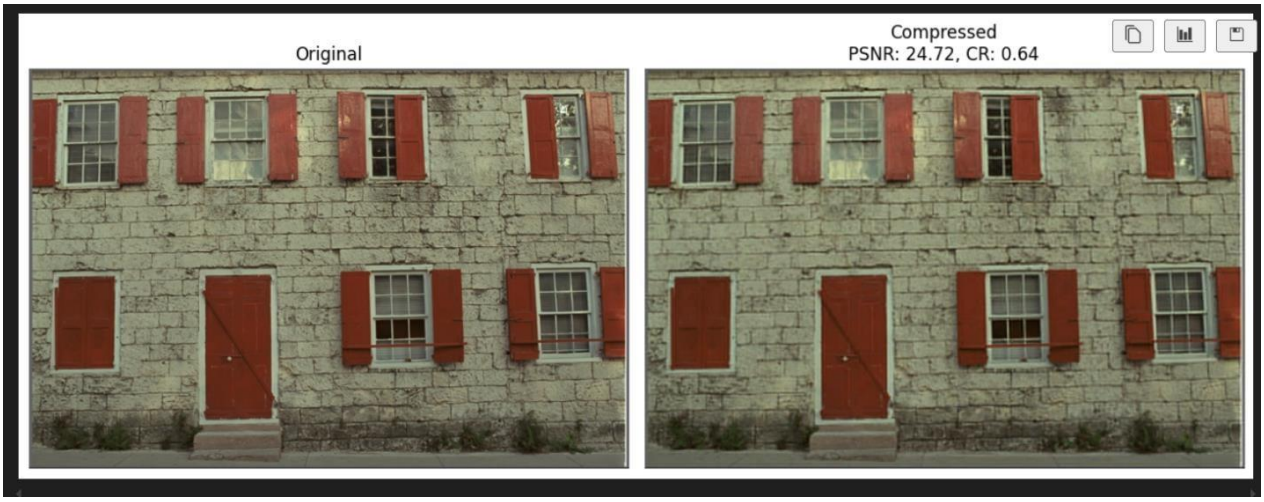


Figure 9: Original and grayscale imageDCT Processing with 10 coefficient...

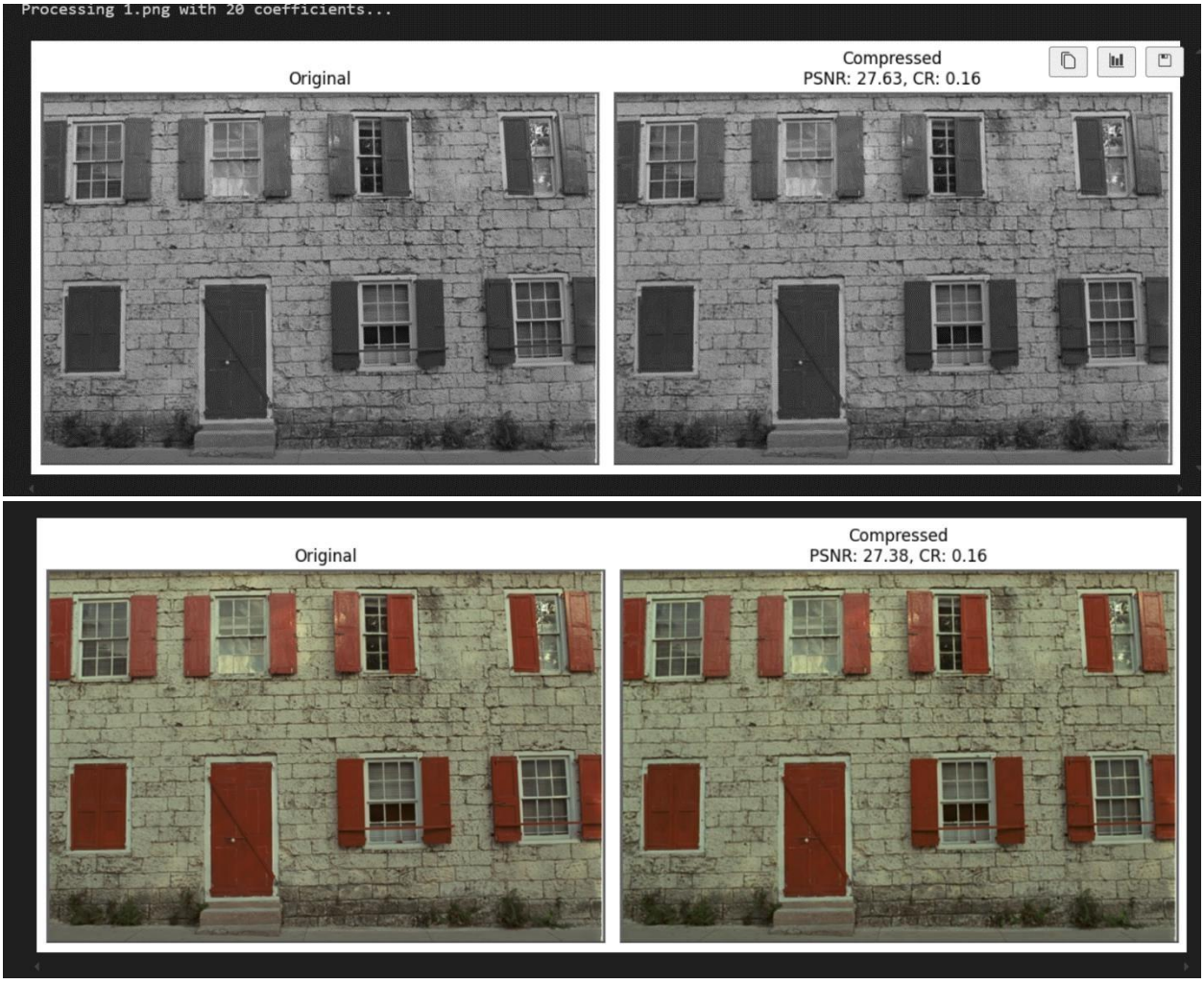


Figure 10:DCT Processing 1.png with 20 coefficients...

	Image	Num Coefficients	PSNR Gray	CR Gray	PSNR Color	CR Color
0	1.png	5	22.51566648158429	2.56	22.50749316782874	2.56
1	1.png	10	24.821728975051172	0.64	24.7174596277982	0.64
2	1.png	20	27.632499974486038	0.16	27.37726445833619	0.16
3	10.png	5	27.447679474191467	2.56	27.285290273547986	2.56
4	10.png	10	30.150839757787494	0.64	29.736008259542405	0.64
5	10.png	20	32.92768959335878	0.16	32.11215793887962	0.16
6	11.png	5	25.582712677015515	2.56	25.470786646149833	2.56
7	11.png	10	27.59699339942074	0.64	27.356733065000466	0.64
8	11.png	20	30.132926445149977	0.16	29.649108636398072	0.16
9	12.png	5	28.79157821870153	2.56	28.513728956390374	2.56
10	12.png	10	31.276717527883882	0.64	30.761651630849254	0.64
11	12.png	20	33.893930075621	0.16	32.973005912007935	0.16
12	13.png	5	21.032464327056864	2.56	21.001758280885877	2.56
13	13.png	10	22.453293741560053	0.64	22.353628436512125	0.64
14	13.png	20	24.744585173299832	0.16	24.52993679097238	0.16
15	14.png	5	25.037102056350328	2.56	24.896871897934904	2.56
16	14.png	10	26.93704940292217	0.64	26.723899633283455	0.64
17	14.png	20	29.602742249176274	0.16	29.11841998388532	0.16
18	15.png	5	27.412627731583495	2.56	27.113863500632	2.56
19	15.png	10	29.806940289210367	0.64	29.34207200378835	0.64
20	15.png	20	32.45509268053351	0.16	31.62374148894014	0.16
21	16.png	5	28.17319877431439	2.56	27.965754627105245	2.56

Figure 11: DCT Processing of All Images with Multiple Coefficients w.r.t PSNR and Compressed Ratios

3.3. Summary of Table Figure 11 for DCT Processing

The table presents data on image compression performance for different images like (1.Png, 10.png etc) using different number of coefficients (5,10,20). For each image and coefficient count, the table provides the PSNR for both grayscale psnr and color images PSNR , along with the corresponding compression ratio (CR) for both grayscale and colored images.

3.3.1.Key Observations:

1. **Image performance consistency:**
   - Each image is tested with three different numbers of coefficients.
   - PSNR and CR values are consistently reported for both grayscale and color formats across different coefficients.

2. **Impact of Coefficients on PSNR:**

Increasing the number of coefficients generally leads to higher PS and R values indicating better image quality.

3. **Compression Ratios (CR):**

- The CR values decrease as the number of coefficients increases.
- For instance, 10.png has a CR of 2.56 with 5 coefficients, which reduces to 0.16 with 20 coefficients for both grayscale and color images.
- This trend indicates a trade-off between image quality (PSNR) and compression efficiency (CR).

4. **Grayscale vs. Color Performance:**

- PSNR values for grayscale and color images are relatively close for the same image and coefficient count.
- However, grayscale images tend to have slightly higher PSNR values compared to color images for the same number of coefficients.
- CR values are identical for both grayscale and color images, reflecting that the compression method applies equally to both formats.

5. **Variation Across Images:**

- Different images have varying PSNR values even for the same number of coefficients.
- For instance, 12.png with 20 coefficients has a higher PSNR (33.89) compared to 13.png (24.74) for grayscale images, indicating variability in how different images respond to compression.

3.4. MDCT Results

Below you can see the images after MDCT implementation.

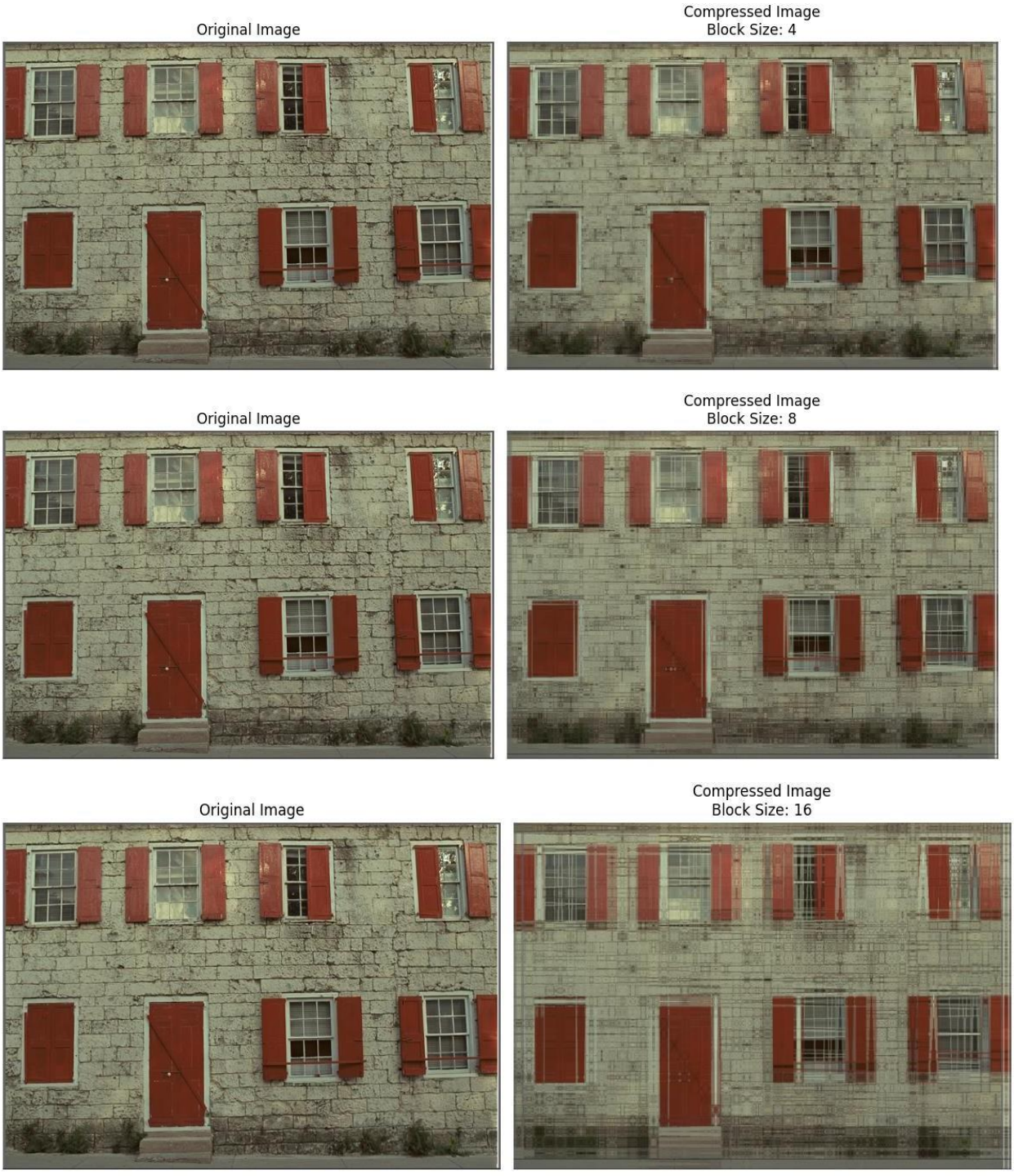


Figure 12: MDCT Processing of All Images

4. Conclusion

4.1. Summary

This project successfully implemented image compression using MDCT and dct. Both methods were evaluated using BPP, PSNR, and perceptual similarity metrics. The MDCT method showed specific result as compared to dct which highlights its prominent features

4.2. Future Work

It could involve exploring other transform based comparison and compression techniques or incorporating advanced machine learning methods for improved the compression performance.

4.3. Acknowledgements

I would like to express my sincere gratitude to all those who have supported and contributed to the successful completion of this Video Coding Seminar project SS2024.

Firstly, I would like to express my sincere gratitude for Prof. Dr.-Ing. Gerald Schuller and Dr. Muhammad Imran, our advisors and Instructors, for their invaluable guidance, support, encouragement and allowing access to their Git Repositories for implementation throughout this project. Their expertise and insights, feedback were instrumental in shaping the direction of this project.

I am also grateful to Technical University Ilmenau's Applied Media Systems Department for providing the necessary resources and facilities required for this project. The access to computational resources and software was crucial for the implementation and testing of the image compression algorithms.

Special thanks to my colleagues and fellow researchers, Shakil and Behzad for their collaborative efforts and constructive discussions that significantly enriched the quality of this work. There assistance with data collection and preliminary analysis was greatly appreciated.

Thank you for all your contributions and support.

5. Important Links-References

- MDCT Implementation by Behzad Hussain
- DCT Implementation by Anns Zahoor 65862
- Perceptual Similarity Implementation by Shakil Mahmud 65374

Relevant Reference literature on mdct, and image compression techniques.

https://github.com/TUIlmenauAMS/Python-Audio-Coder/blob/master/DCT4.py

https://github.com/richzhang/PerceptualSimilarity.git

https://github.com/Karanraj06/image-compression.git