Image and Video Compression Project Report

Prepared by: Charbel Daher

Presented to : Doctor Elias Zaghrini

1. Introduction

This project implements various techniques for image and video compression, including:

- **Discrete Cosine Transform (DCT) based compression**
- **Quantization and dequantization**
- **Zigzag encoding**
- **Run-length encoding**
- **Huffman coding**
- **Arithmetic coding**
- **Motion estimation and compensation for video**

The goal is to achieve high compression ratios while maintaining acceptable image/video quality. Metrics like Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) are used to evaluate the compression performance.

2. Implementation Details

Image Encoding

The `encode_image` function is the main entry point for image compression. It takes the input image, quantization table type (luminance/chrominance), quality factor, and encoding type (Huffman/arithmetic) as parameters.

The key steps are:

1. **`devide_image`: The input image is divided into 8x8 blocks. This is done to enable block-based processing.**
2. **`create_basis_mat`: The DCT basis matrix is pre-computed. This matrix contains the cosine basis functions used in the DCT transform.**
3. **`dct8_image`: Each 8x8 block undergoes Discrete Cosine Transform (DCT). The DCT converts the image from the spatial domain to the frequency domain. It separates the high and low frequency information.**
4. **`quantize`: The DCT coefficients are quantized using a quantization matrix. The quantization matrix is scaled based on the quality factor. Higher quality factors result in less quantization (higher quality, lower compression).**
5. **`to_zigzag`: The quantized blocks are scanned in a zigzag order to convert the 2D block into a 1D sequence. This groups the low-frequency coefficients (which are more important visually) at the start of the sequence.**
6. **`run_len_encode`: Run-length encoding is applied to the 1D sequence. This replaces runs of zeros (which are common after quantization) with a (skip, value) pair, indicating the number of zeros and the next non-zero value.**
7. **Entropy coding: The run-length encoded data is then compressed using either Huffman coding (`huffman_code`) or arithmetic coding (`enocde_arithmetic`). These exploit the statistical redundancy in the data to achieve compression.**

The function returns the encoded bitstream, image dimensions, and the encoding table (Huffman table or arithmetic coding range).

Image Decoding

The `decode_image` function reverses the encoding process to reconstruct the image from the compressed bitstream.

The key steps are:

1. **1. Entropy decoding: The bitstream is decoded using either Huffman decoding (`huff_decode_stream`) or arithmetic decoding (`decode_arithmetic`) based on the encoding type. This recovers the run-length encoded sequence.**
2. **`run_len_decode`: The run-length encoded sequence is decoded to get back the quantized 1D sequence.**
3. **`from_zigzag`: The 1D sequence is converted back to 2D blocks using the inverse zigzag scan.**
4. **`dequantize`: The quantized coefficients are dequantized using the same quantization matrix used during encoding.**
5. **5. `idct8_image`: The Inverse DCT (IDCT) is applied to each block to convert it from the frequency domain back to the spatial domain.**
6. **`combine_image`: The 8x8 blocks are combined to form the reconstructed image.**

The function returns the reconstructed image. Due to the lossy quantization step, the reconstructed image is an approximation of the original image. The level of loss is controlled by the quality factor.

Lossless Compression

The project also includes a lossless compression mode in the `encode_image_lossless` and `decode_image_lossless` functions.

The lossless encoding uses delta encoding (storing the difference between adjacent pixels) followed by Huffman coding. The first column of the image is stored separately to enable reconstruction.

The decoding function reverses this process - it Huffman decodes the bitstream, reconstructs the first column, and then recovers the image by cumulatively adding the deltas.

These lossless functions provide an option for perfect reconstruction at the cost of lower compression ratios compared to the lossy DCT-based method.

Video Compression

The video compression implemented in this project builds upon the image compression techniques and adds inter-frame compression to exploit temporal redundancy between frames.

The key idea is to use motion estimation and compensation to predict the current frame from a previous reference frame. Only the difference (residual) between the predicted and actual current frame needs to be encoded, along with the motion vectors. This results in significant compression as the residuals have much less information compared to the full frame.

The `encode_vid` function is the main entry point for video compression. It takes the input video file, output file name, encoding parameters (quality factor, encoding type), and other options as parameters.

The key steps are:

The video is read frame by frame using OpenCV's `VideoCapture`.

The first frame is intra-coded (I-frame) using the image compression functions (`encode_image` and `decode_image`). This frame serves as the initial reference frame.

For each subsequent frame:

`get_motion_mat`: Motion estimation is performed. For each 8x8 block in the current frame, the best matching block in the reference frame is found. The search is limited to a small neighborhood around the block's position. The motion vector (displacement) is recorded.

The motion vectors are encoded using Huffman coding to compress them.

`apply_motion`: The motion vectors are used to construct a predicted frame from the reference frame. This is done by copying the matched blocks from the reference frame to the corresponding positions in the predicted frame.

The residual (difference) between the predicted frame and the actual current frame is computed.

The residual is encoded using the image compression functions (`encode_image` and `decode_image`) for each color channel separately. Encoding the residual instead of the full frame results in significant compression.

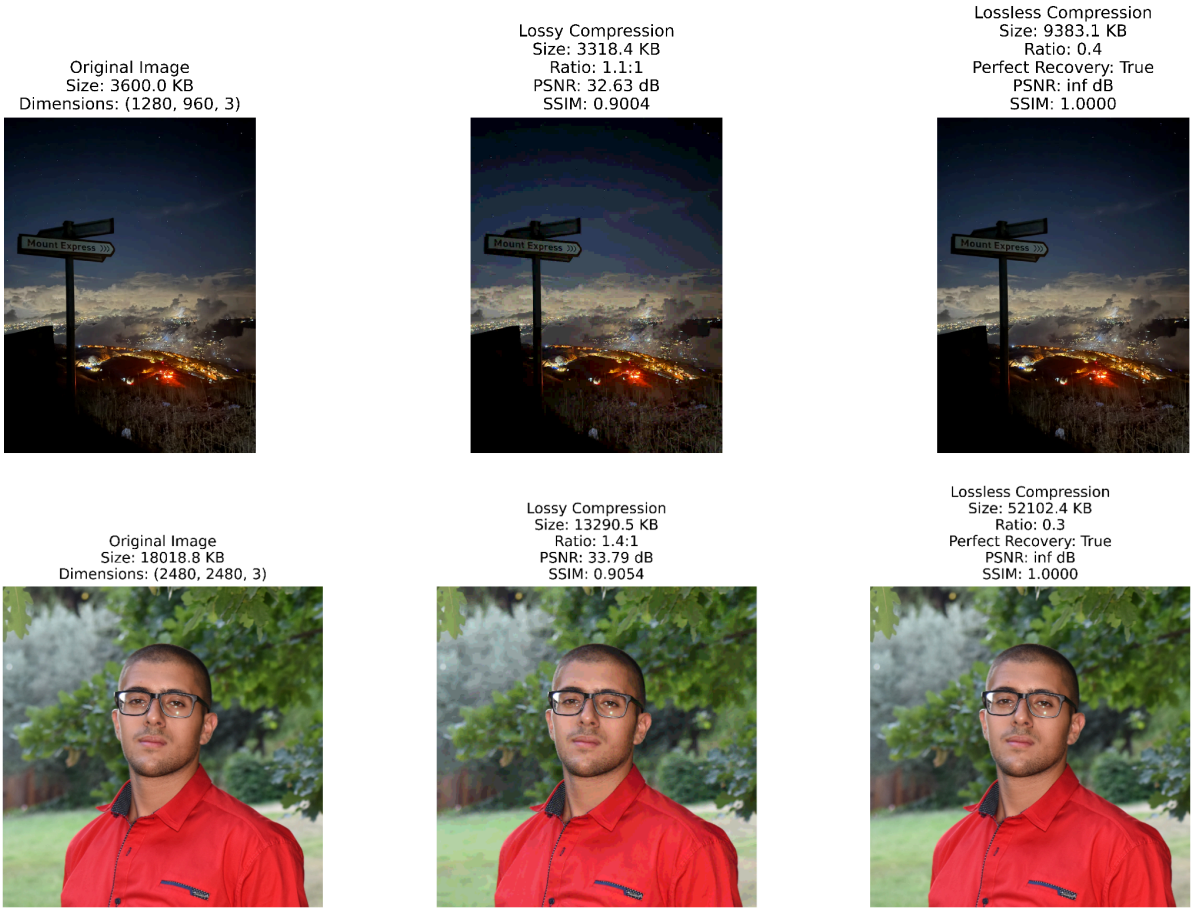
The encoded motion vectors and residuals are written to the output file.

The decoded current frame (predicted frame + decoded residual) is used as the reference frame for the next iteration.

4. The process continues until all frames are encoded. The compressed video is written to the output file.

For video decoding, the inverse process is followed. The I-frame is decoded first using image decompression. Then for each subsequent frame, the motion vectors are decoded, the predicted frame is constructed using motion compensation, the residuals are decoded and added to the predicted frame to reconstruct the current frame.

3. Results



For the video, you can see the videos in the folder

4. Conclusion

In this project, I implemented several fundamental image and video compression techniques. Key learnings include:

The DCT is powerful for converting images to frequency domain and enabling compression

Quantization is a lossy but critical step that allows tuning compression ratio vs quality

Entropy coding techniques like Huffman and arithmetic coding are essential for exploiting statistical redundancy

Motion estimation and compensation are the key techniques that make video compression much more efficient than compressing individual frames