CS 663: Fundamentals of Digital Image Processing
JPEG Image Compression

Jashwanth **Vishal** **Alvin**
22B1033 22B1015 22B0015

Contents

1 Problem Statement

The `JPEG (Joint Photographic Experts Group)` standard is one of the most popular methods for compressing images. It uses a lossy compression technique to make image files smaller while still looking good.

In this project, we created a simple version of a **JPEG-like compression method that works for grayscale**. We also looked at small ways to make the compression better.

2 Description of algorithms implemented

2.1 Encoder

The encoder compresses an input image using the following steps:

1. **Block Splitting and padding:** Split the image into 8×8 blocks (pad the image if necessary).
2. **Discrete Cosine Transform (DCT):** Apply the DCT to each block and get the coefficients into an 8x8 block
3. **Quantization:** Quantize the DCT coefficients by dividing with a scaled quantization matrix $\frac{50}{Q} \cdot$ base_quantmatrix and rounding to an integer.
4. **AC,DC coefficients:** We subtract the values of first 8 * 8 block (DC coefficients) from rest 8*8 blocks (Gives AC coefficients)
5. **Zigzag Scanning:** While flattening the data instead of going row wise go in a zigzag manner in each block so that we get more zeros in end.
6. **Run-Length Encoding (RLE):** Encode the zigzag coefficients using RLE.We do RLE for each 8*8 block and combine these RLE for all blocks except the first into one.
7. **Huffman Encoding:** Compress the RLE output using Huffman coding. Here we use different trees for the first 8*8 block and rest the blocks
8. **Compressed Data:** We write the following data into a .bin file.
   1) The size of the image.
   2) Quality factor.
   3) Huffman encoded maps for both dc and ac separately.
   4) Encoded data.
   5) Quantization matrix

2.2 Decoder

The decoder reconstructs the original image from the compressed file using the following steps:

1. **Load Compressed Data:** Load the binary file containing the encoded data, Huffman codes, and metadata and store them in the required format.

2. **Huffman Decoding:** Decode the Huffman-encoded data to retrieve the RLE sequence of DC and AC coefficients.

3. **Run-Length Decoding:** Decode the RLE sequence to reconstruct the quantized DCT coefficients(AC and DC) in a 1-D sequence.

4. **Inverse Zigzag Scanning:** Restore the 2D block structure of the quantized coefficients by placing these values in a ZigZag manner.

5. **Dequantization:** Multiply the coefficients by the scaled quantization matrix (Calculated in the same way).

6. **Inverse DCT (IDCT):** Apply the IDCT to reconstruct the image blocks to the respective channels.

2.3 Quantization Matrix Scaling

The quality factor affects the compression ratio and image quality. The quantization matrix is scaled as follows:

$$Q_{\text{scaled}} = \max\left(1, \text{round}\left(Q \times \frac{50}{\text{quality factor}}\right)\right) \tag{1}$$

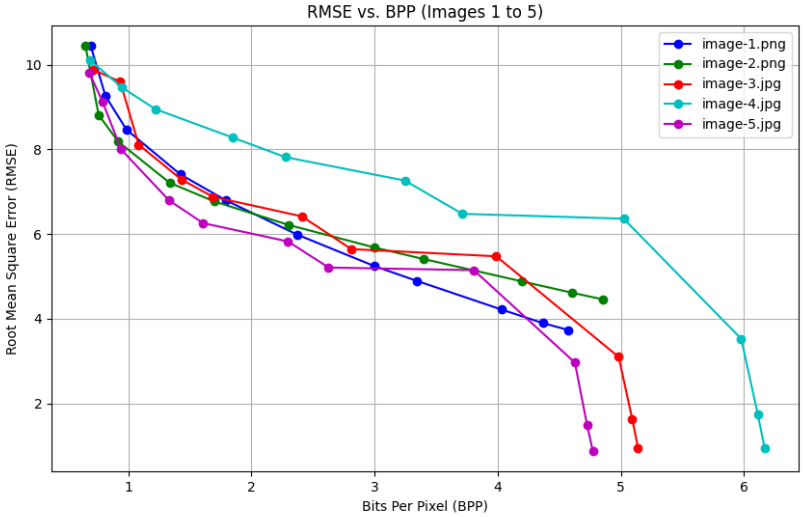
2.4 Datasets

- We used a grayscale image dataset, which includes images downloaded from an online source and some we added ourselves. The dataset can be accessed here: Dataset for Gray Scale.

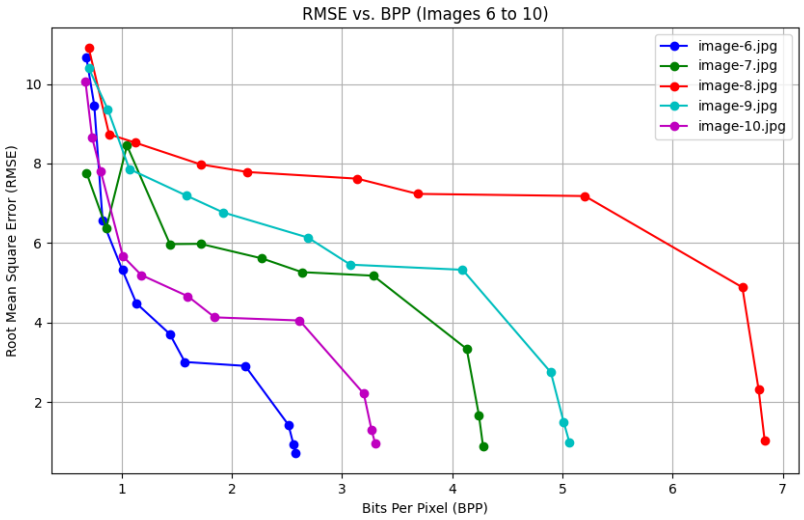
- We used ORL dataset of face images for PCA based compression implementation. The dataset can be accessed here: dataset for PCA

- We need cartoon images to run the research paper implementation . The dataset can be accessed here: Cartoon images

3 Plots (Grayscale) for RMSE vs BPP

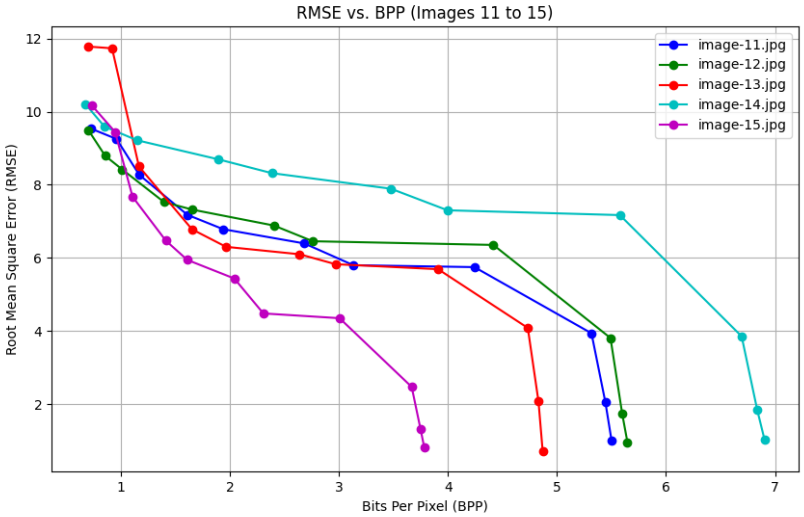


(a) RMSE vs BPP for Group 1

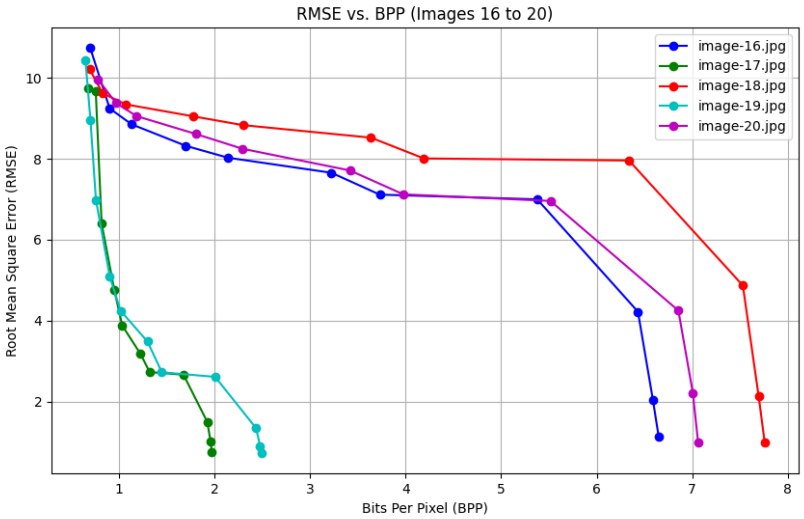


(b) RMSE vs BPP for Group 2

Figure 1: RMSE vs BPP plots for Groups 1 and 2.



(a) RMSE vs BPP for Group 3



(b) RMSE vs BPP for Group 4

Figure 2: RMSE vs BPP plots for Groups 3 and 4.

4 Visual analysis for few grayscale images

(a) Quality factor = 3

(b) Quality factor = 25

(c) Quality factor = 50

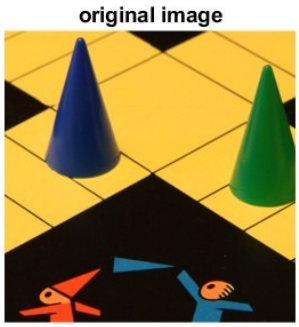
(d) Quality factor = 75

Reconstructed images from our JPEG implementation.

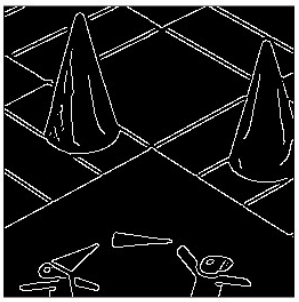
5 Implementation of research paper:Edge-Based Image Compression with Homogeneous Diffusion

5.1 Introduction

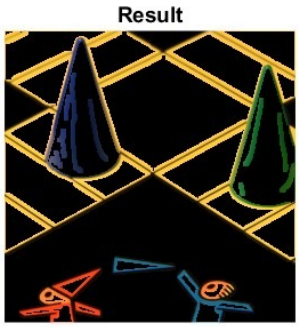
We propose a lossy compression technique for cartoon like images that utilizes edge information for efficient data representation. The locations of image edges are encoded losslessly using the JBIG algorithm. During decoding,missing regions are reconstructed by solving the Laplace equation, which simulates a homogeneous diffusion process to fill in unspecified ares. This approach allows for high-quality reconstruction of the images, leveraging edge locations and adjacent pixel values to maintain sharp details and smooth transitions in cartoon like images.



(a) Original Image



(b) Edge Image



(c) Colours next to the edges

(d) Compressed Image

(a) Original Image

(b) Edge Image

(c) Colours next to the edges

(d) Compressed Image

5.2 Steps of Encoding

- Detecting Edges:
  - The encoding process begins with edge detection using the Canny edge detection Algorithm
  - This step produces well-localized contours that are often closed, particularly for cartoon-like images
- Encoding Contour Locations:
  - The output of the edge detection step is a bi-level edge image(binary mask), which represents the locations of edges
  - This mask indirectly encodes the contour locations for interpolation in the reconstruction phase
  - The bi-level image is compressed and stored using the JBIG compression algorithm
- Encoding Contour Pixel Values:

  - Instead of storing pixel values directly on the edges, pixel intensities on both sides of the edges are stored, as these typically represent regions of different brightness or color
  - Additionally, all pixel values along the border of the image are saved. These values serve as Dirichlet boundary conditions during the diffusion-based interpolation step in the reconstruction phase

- Storing Encoded Data:
  - The edge mask and the pixel intensity values are compressed and stored using ZPAQ compression

5.3 Steps of Decoding and Reconstruction

- Decoding Contour Locations and Pixel Values
  - The encoded file is split into two parts: JBIG data(for the edge mask) and PAQ data(for pixel intensity values)
  - These components are decoded using the respective JBIG and PAQ decoding methods to retrive the edge locations and pixel values.

- Reconstructing Missing Data
  - Using the decoded edge locations and pixel values, the missing pixel intensities are reconstructed
  - This is achieved through homogeneous diffusion, where the laplacian operator is applied iteratively to propagate intensity values between edges
  - The reconstructed image is an interpolation of the decoded data, ensuring smooth transitions while preserving the detected edges.

6 PCA-Based Image Compression

6.1 Introduction

In lectures, we saw that compression can be done by calculating eigenfaces, mean, and storing coefficients for each image (this gives us overall compression as very few eigenfaces should be stored for a large number of images, and each image coefficient doesn't take much space).

$$\text{Compression} = \frac{N \cdot k \cdot 4 + k \cdot \text{size}}{N \cdot \text{size}}$$

Where size is no. of pixels in the image.

6.2 Implementation of Compression and Decompression

6.2.1 Compression

- Flatten the images into vectors and center the data by subtracting the mean image
- Compute the covariance matrix and extract eigenvectors(principal components)
- Select the top k eigenvectors and project the data onto this k-dimensional subspace
- Store the projection coefficients(eigen-coefficients),eigenvectors, and mean image

6.2.2 Reconstruction

- Multiply the eigen-coefficients by the eigenvectors to reconstruct the centered data
- Add the mean image to the centered to recover the approximate original
- The reconstructed image retains key features but with some loss based on k

6.2.3 Method Mathematically

- We will be using the computationally efficient PCA i.e we will take eigenvectors of $X^{\mathrm{T}}X$ and then compute eigenvectors of XX^{T} by computing XV
- Now extract the top k eigenvectors (V_k) and find coefficients for all images by formula $V_k^{\mathrm{T}}X$ and get eigen coefficients for each image by getting corresponding row
- We can store the coefficients for each image and the average image, eigen coefficients leading to compression for the whole set of images
- For reconstructing the image just multiply coefficients with the corresponding eigenface and add all of them to the mean.

6.3 Plots

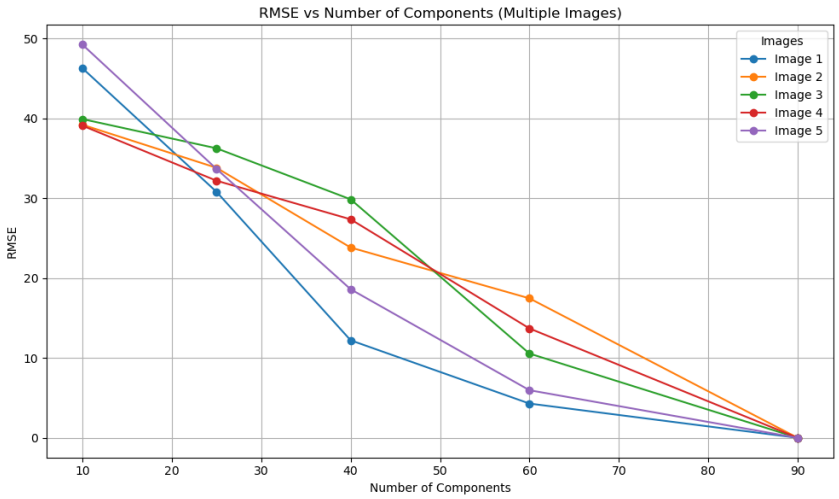


Figure 6: RMSE vs No. of components

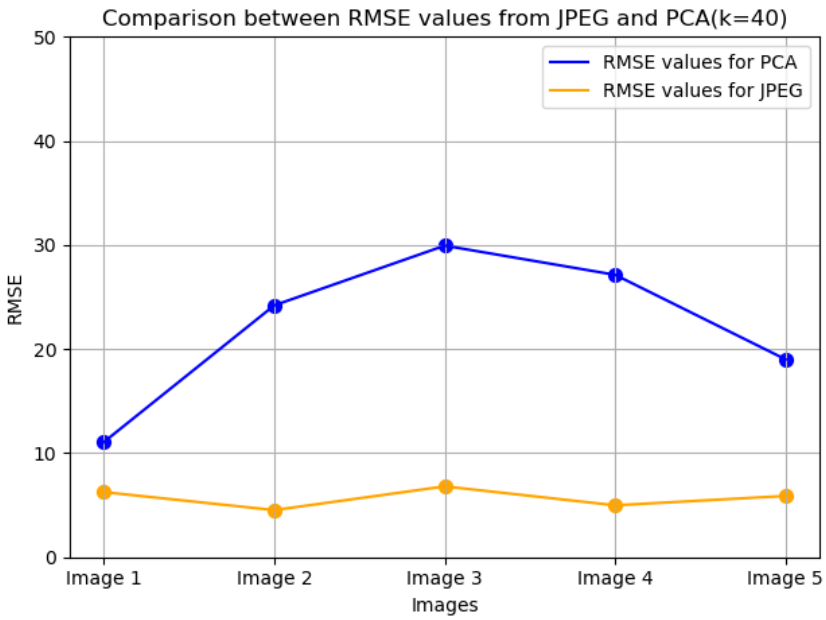


Figure 7: Graph for comparison of our JPEG(Q = 25) and PCA(k = 40) for different images

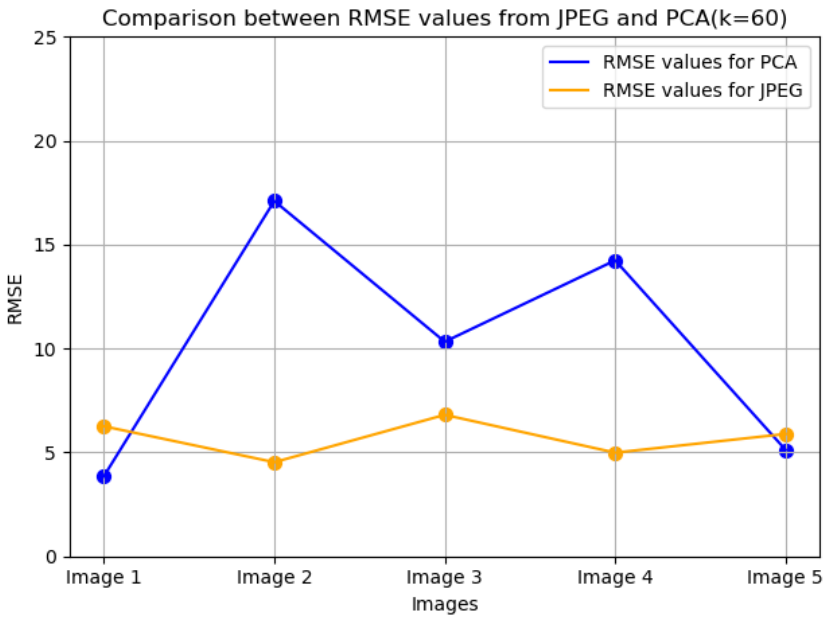


Figure 8: Graph for comparison of our JPEG(Q = 25) and PCA(k = 60) for different images

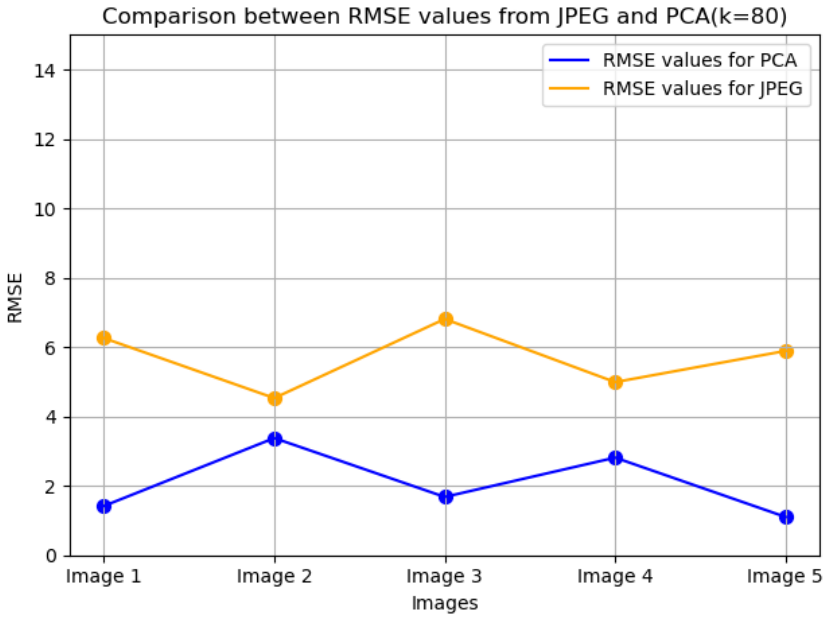


Figure 9: Graph for comparison of our JPEG(Q = 25) and PCA(k = 80) for different images

6.4 Reconstructed Faces and Others

(a) Original Face

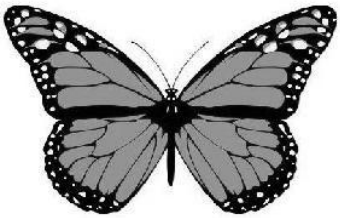
(b) Image compressed by PCA with k = 50

(c) Using our JPEG Q = 25

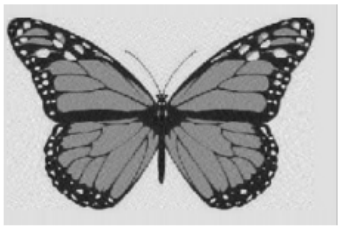
(d) Original Face

(e) Image compressed by PCA with k = 50

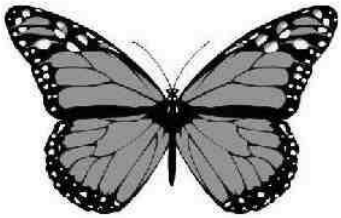
(f) Using our JPEG Q = 25



(g) Original Face



(h) Image compressed by PCA with k = 50

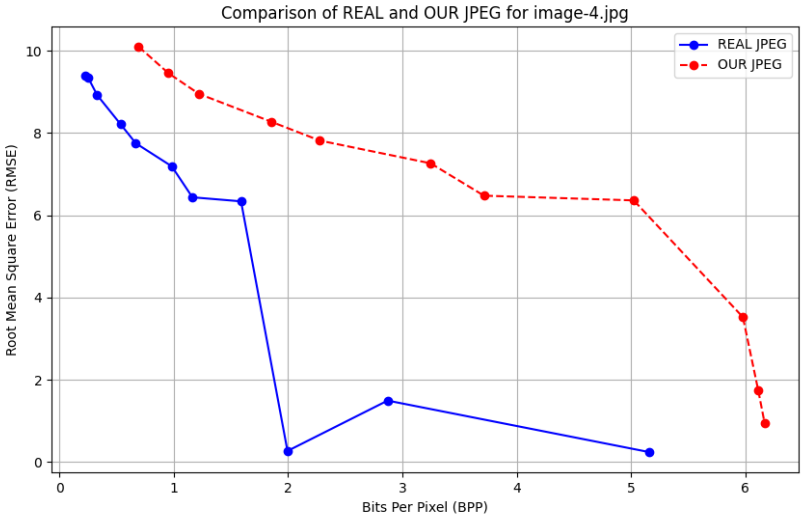


(i) Using our JPEG Q = 25

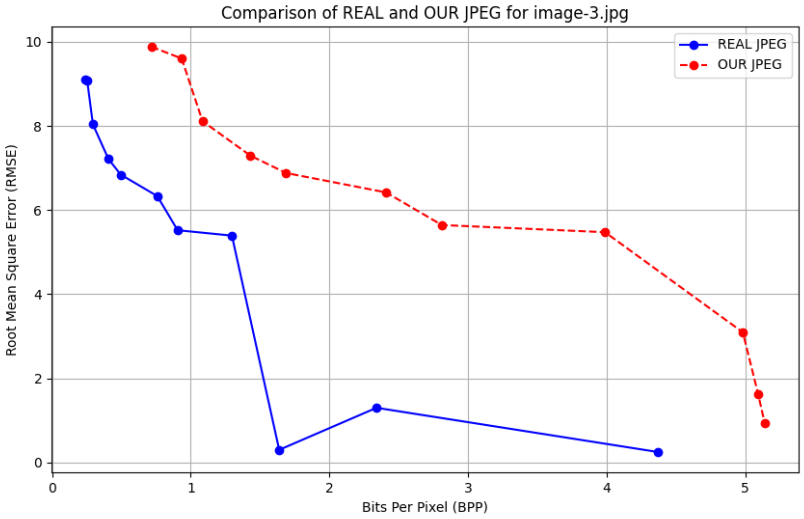
Comparision between PCA and JPEG

7 Comparison with JPEG with our compressor

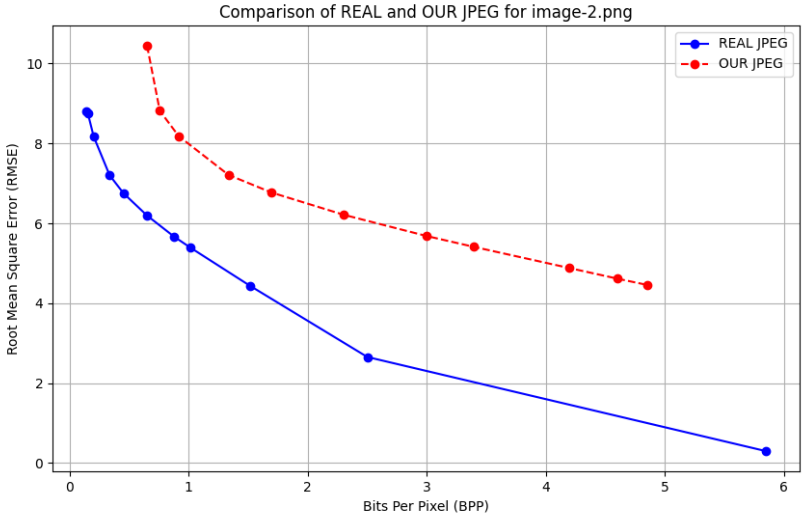
7.1 RMSE vs BPP for Grey Scale Images



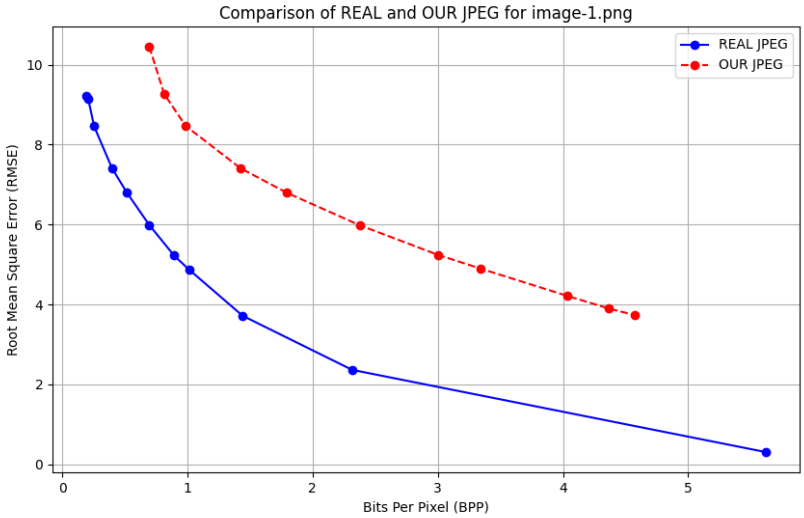
(a) Grey Scale image - 1



(b) Grey Scale image - 2



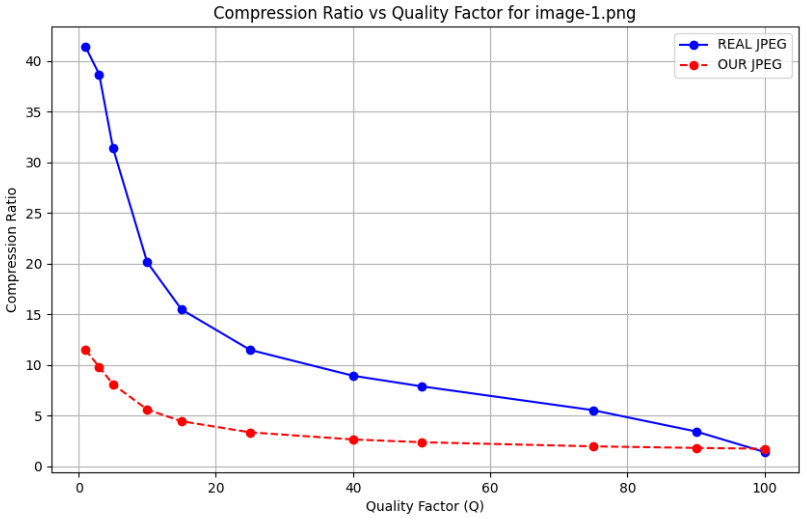
(a) Grey Scale image - 3



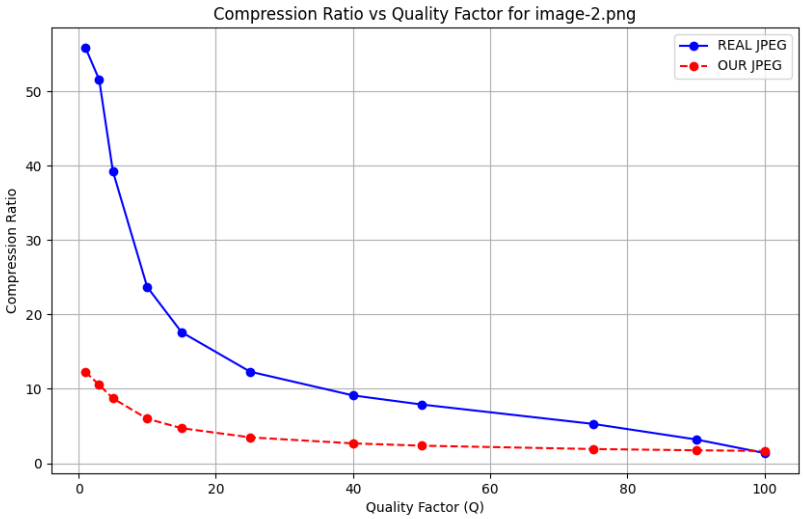
(b) Grey Scale image - 4

Comparison with JPEG for Grey Scale images

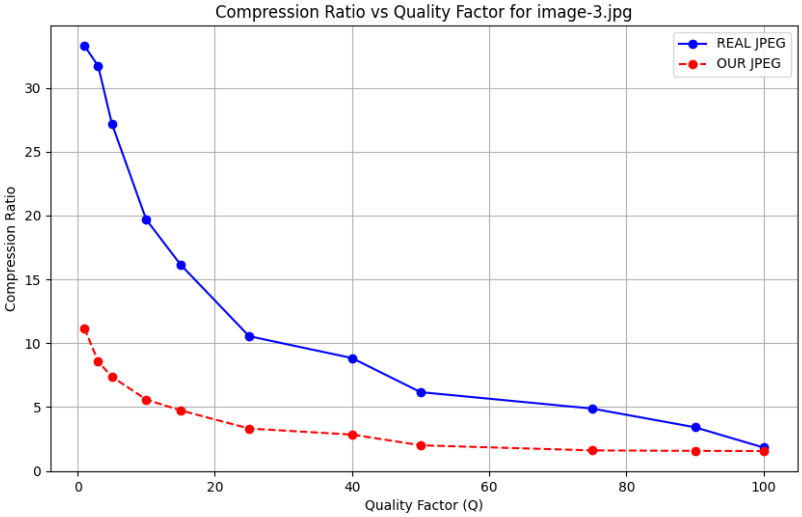
7.2 Compression Ratio of Grey Scale Images



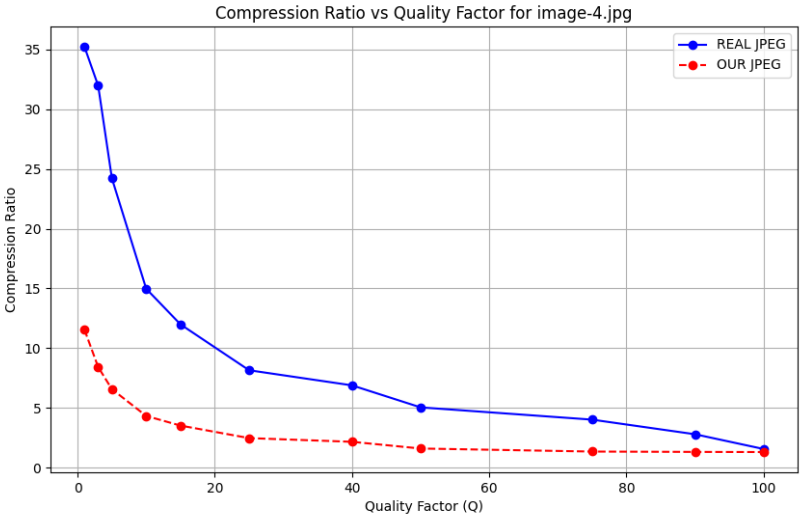
(a) Grey Scale image - 1



(b) Grey Scale image - 2



(a) Grey Scale image - 3



(b) Grey Scale image - 4

Comparison with JPEG for Grey Scale images

8 Contributions

We had done most part as a team. But major part of contribution is as follows :

- **Jashwanth:** JPEG implementation of Grayscale image with huffmann encoding, Comparision with real JPEG and report
- **Vishal:** Research paper implementation and report
- **Alvin:** PCA based image compression,Comparision of PCA based Image compression with our JPEG and report

9 Instructions to Run

- For Running our JPEG Implementation
  - Rename the image files in the formate of `image-i.jpg`, and run `execution.py`
  - It generates the restored images for various Q factors and the plots `RMSE vs BPP` for each image in groups of 5 are created in the results directory.
  - `inbui.py` is the real jpeg implementation found on internet
  - `real vs our jpeg.py` generates the plots with rmse with image for 4 images
  - `compression vs Quality.py` generate the plots of Compression Ratio with Quality factor for 4 images
- To run the research paper Implementation
  - First we need to rename our image names to in the format of `'imi.png'`, also change the for loop range accordingly for various values of i in code manually
  - Then run `encode.m` in matlab, which generates the `pbm` (mask after canny edge detection), `png` (canny edges with color) files
  - Go into data folder, run the `compress.py` in the terminal, which converts `.pbm` to `.jbg` and compresses into an archive
  - Run `decompress.py` in the terminal
  - Finally run the `decode.m` in matlab, which generates the restored image
- To run the PCA code
  - First, change the image folder path in the code
  - Run the jupyter file `pca.ipynb` to run the code. input will be taken from folder, restored images will be formed

References

[1] Michael Mainberger and Joachim Weickert. Edge-based image compression with homogeneous diffusion. In *Computer Analysis of Images and Patterns (CAIP)*, pages 476–483. Springer, 2009.

[1]