CS 663 Digital Image Processing
Project Report

22b0913 | 22b0949 | 22b1030
Umesh Karthikeya | Gnana Mahesh | Varun Guptha

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 both grayscale and color images. 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. **Color Space Conversion:** Convert the RGB image for color images to the YCbCr color space for better compression performance as these are uncorrelated.
2. **Down Subsampling:** Downsample the Cb and Cr channels using a factor of 2 (4:2:0 sampling).
3. **Block Splitting and padding:** Split the image into 8×8 blocks (pad the image if necessary).
4. **Discrete Cosine Transform (DCT):** Apply the DCT to each block and get the coefficients into an 8x8 block(For colour images do it for YCbCr differently).
5. **Quantization:** Quantize the DCT coefficients by dividing with a scaled quantization matrix $\frac{50}{Q} \cdot$ base_quantmatrix and rounding to an integer.
6. **AC,DC coefficients:** We subtract the values of first 8 * 8 block (DC coefficients) from rest 8*8 blocks (Gives AC coefficients)
7. **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.
8. **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.
9. **Huffman Encoding:** Compress the RLE output using Huffman coding. Here we use different trees for the first 8*8 block and rest the blocks
10. **Compressed Data:** We write the following data into a .bin file.
    1) Bit indicating image is colored or not.
    2) The size of the image.
    3) Quality factor.
    4) Huffman encoded Coefficients.
    5) Huffman tree in preorder format with delimiters.

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.
7. **UP Sampling:** Up sample the Cb,Cr channels.
8. **Recombine Channels:** Reconstruct the full image by merging the Y, Cb, and Cr channels for color images.
9. **Color Space Conversion:** Convert the image back to the RGB color space for color images.

2.3 Quantization Matrix Scaling

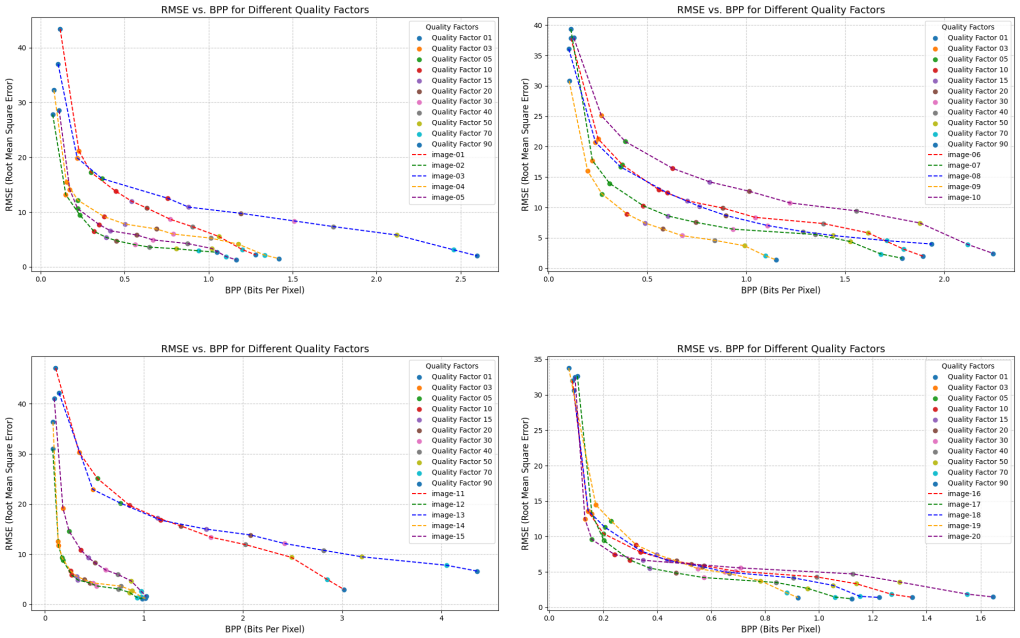
The quality factor affects the compression ratio and image quality. The quantization matrix is scaled as follows:

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

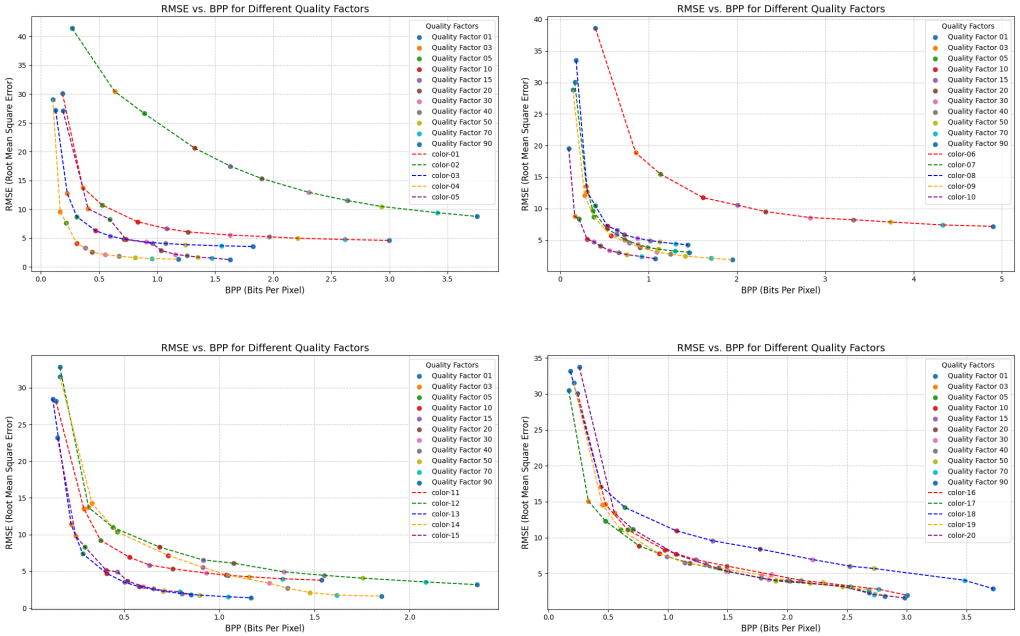
2.4 Datasets

- 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: Grayscale Dataset.
- We compiled a collection of color images by downloading them from Google and various other sources. The dataset is available here: Color Dataset.
- We used ORL dataset of face images for PCA based compression implementation. The dataset can be accessed here: ORL dataset
- 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



4 Plots(Colour) for RMSE vs BPP



5 Visual analysis for few grayscale images

(a) Quality factor = 3 (b) Quality factor = 30

(c) Quality factor = 50 (d) Quality factor = 70

Reconstructed images from our JPEG implementation.

6 Visual analysis for few color images

(a) Quality factor= 1

(b) Quality factor= 5

(c) Quality factor= 30

(d) Quality factor= 70

Reconstructed images from our JPEG implementation.

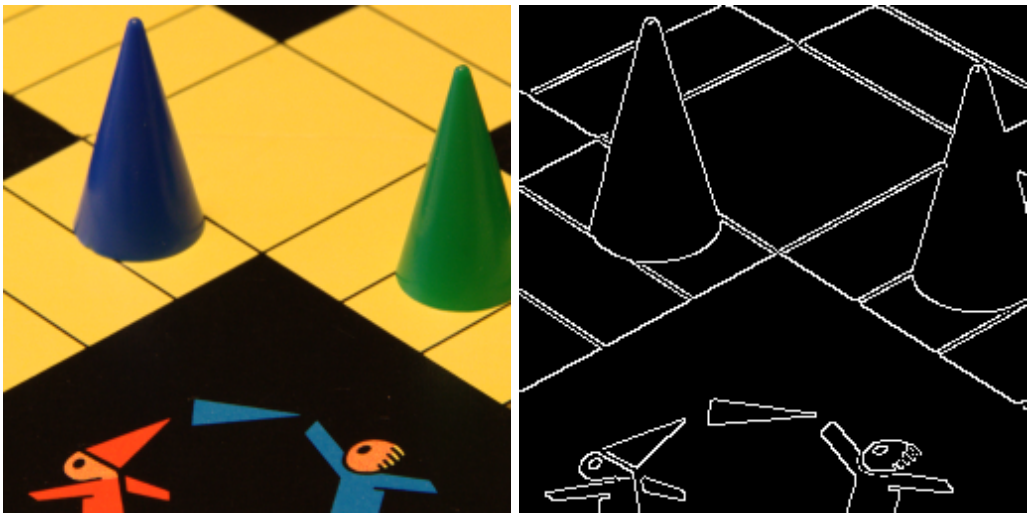
7 Implementation of research paper: Edge-Based Image Compression with Homogeneous Diffusion

7.1 Introduction

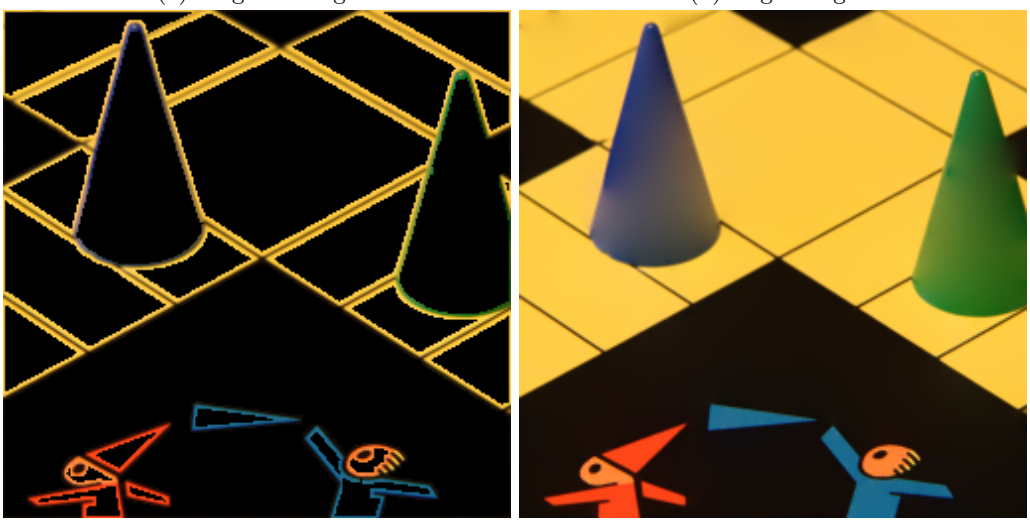
In this paper [1] , we present a lossy compression method for cartoon-like images that exploit information at image edges, The location of edges is stored in a lossless way using JBIG, In the decoding step, information outside these encoded data is recovered by solving the Laplace equation, i.e. we in paint with the steady state of a homogeneous diffusion process

In the implementation, we have used a novel approach that uses edge locations and adjacent gray or color values to reconstruct high-quality cartoon-like images by filling unspecified regions through homogeneous diffusion.

7.2 Examples

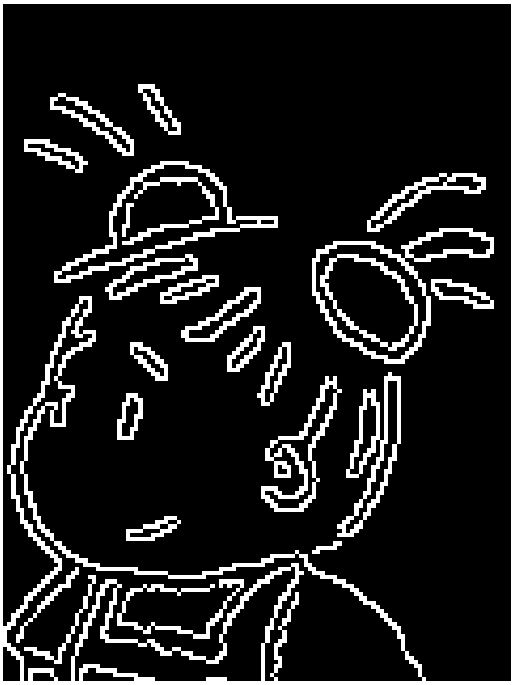


(a) Original Image (b) Edge Image



(c) Colours next to the edges (d) Compressed Image

(a) Original Image



(b) Edge Image

(c) Colours next to the edges

(d) Compressed Image

7.3 Steps of Compression

1. **Detecting Edges:** Our encoding of an image starts with an edge detection. We use canny edge detection for this step. For cartoon-like images, this algorithm gives well-localised contours that are often closed.
2. **Encoding the Contour Location :** The result of the edge detection step can be regarded as a bi-level edge image as illustrated in Fig. (b). This bi-level image encodes indirectly the location that is used for interpolation later on. We store this image by using the JBIG.
3. **Encoding the Contour Pixel Values :** Next, we consider the pixel values we want to store. Since edges usually split areas of different brightness or color, we do not use the pixel values that lie directly on the edge, but store the values from both sides of the edge instead (see Fig. (c)). Additionally, all pixel values from the border of the image domain are stored such that these values are also available as Dirichlet boundary for the diffusion-based filling-in later on.
4. **Storing the Encoded Data :** We will use ZPAQ compression and store both the mask and image containing intensities of edges and intensities of pixels on either side of edges.

7.4 Steps of decoding image

1. **Decoding the Contour Location and Pixel Values :** As a first step of the decoding phase, we split our encoded file into the JBIG data and PAQ data part. Both parts are then decoded by using the JBIG and the PAQ method, respectively.
2. **Reconstructing Missing Data :** So far, we have decoded the edge locations and the pixel values surrounding the edges. The idea is now to apply interpolation to reconstruct grey or color values of pixels that lie between edges. We keep the pixel values at the positions obtained in Step 1 and use homogeneous diffusion for interpolation. Homogenous diffusion can be done by applying Laplacian to the image and adding it to it for multiple iterations.

8 PCA-based image compression for face images

8.1 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).

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

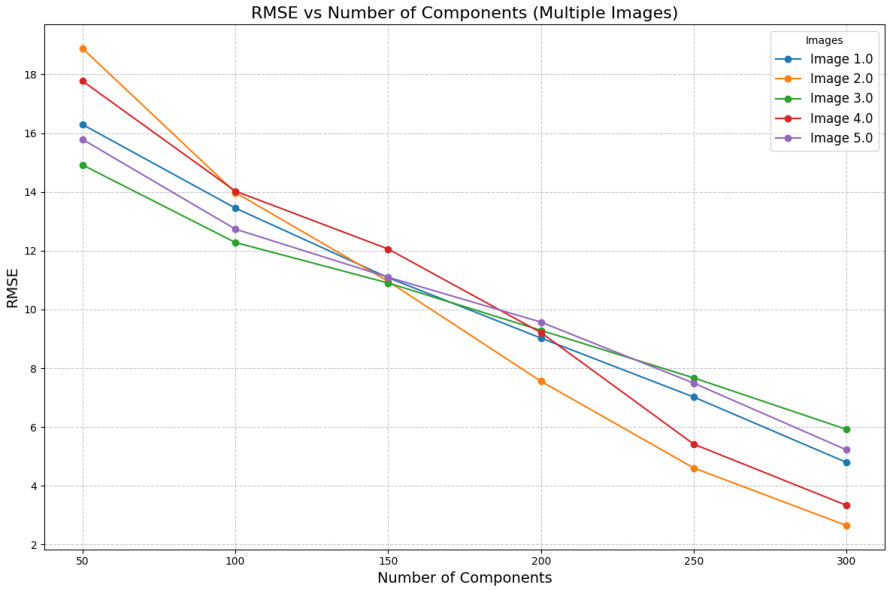
Where size is no. of pixels in the image.

8.2 Method of compression and decompression

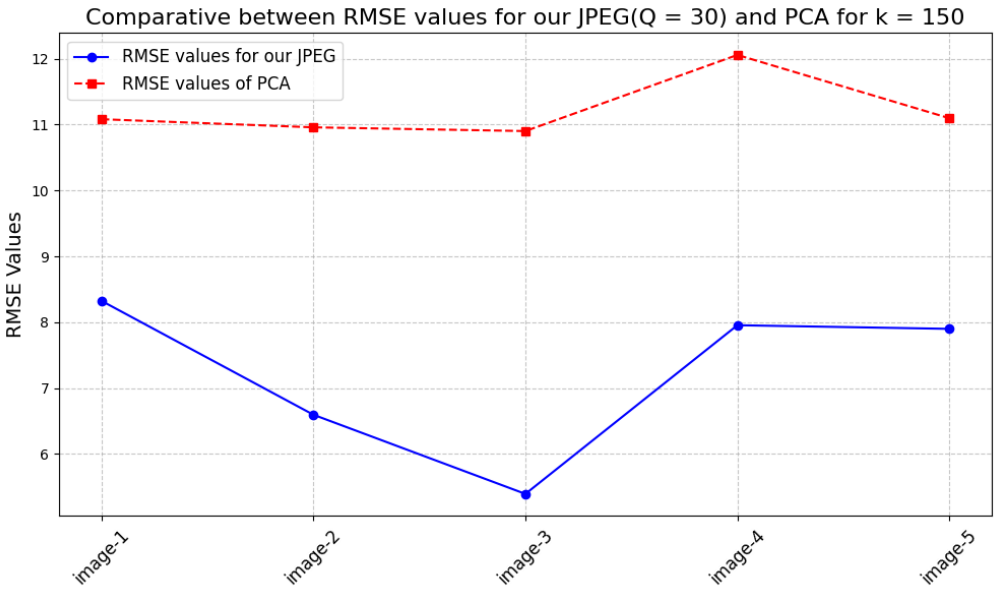
- We will be using the computationally efficient PCA i.e we will take eigenvectors of $X^{\mathrm{T}}X$ and then compute eigenvectors of XX^{T} by computing XV
- Now extract the top k eigenvectors (V_k) and find coefficients for all images by formula $V_k^{\mathrm{T}}X$ and get eigen coefficients for 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.

8.3 Plots

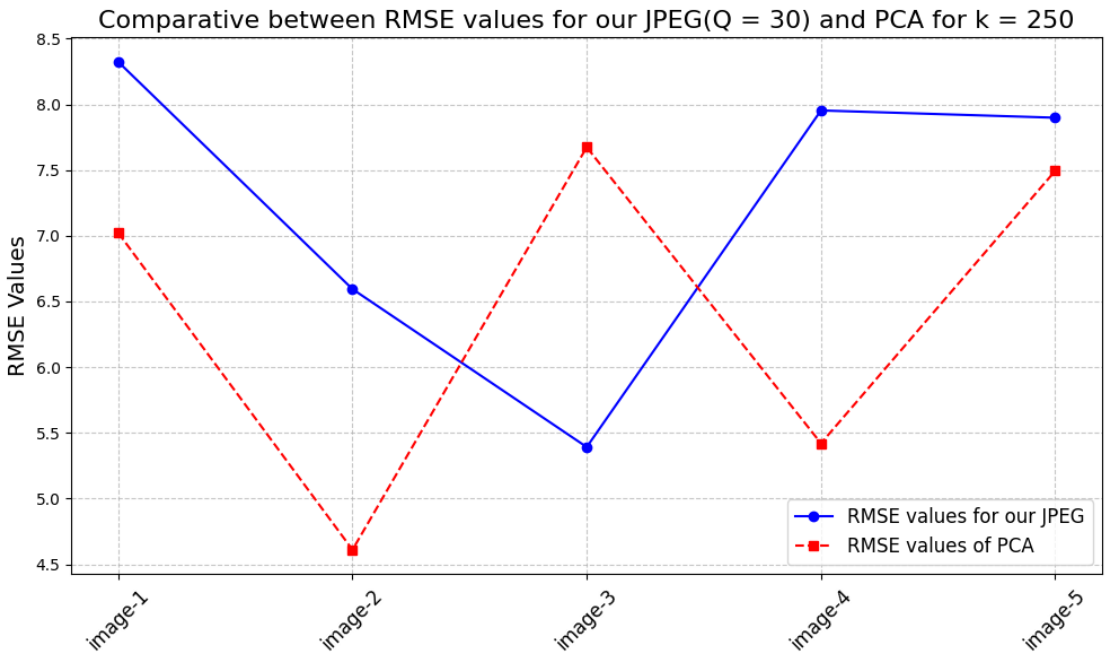
8.4 Comparison of our JPEG and PCA for different k values



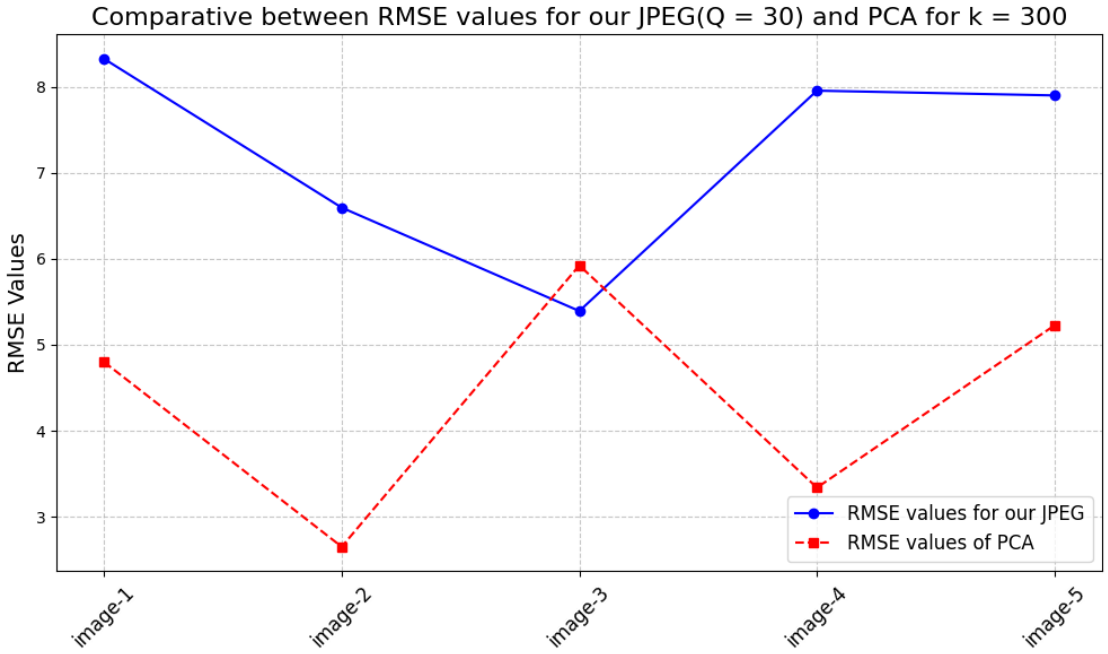
Graph between RMSE and number of components for various images



Graph for comparison of our JPEG(Q = 30) and PCA(k = 150) for different images

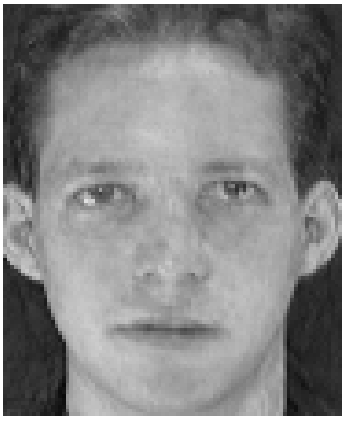


Graph for comparison of our JPEG(Q = 30) and PCA(k = 250) for different images

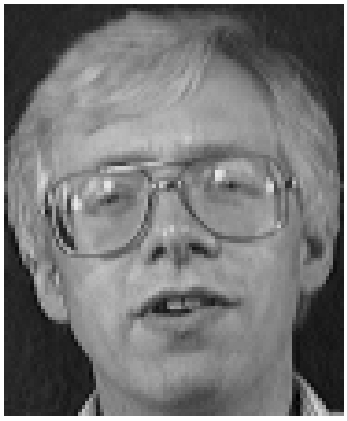


Graph for comparison of our JPEG(Q = 30) and PCA(k = 300) for different images

8.5 Reconstructed Faces



(a) Image compressed by PCA with k = 300



(b) Image compressed by PCA with k = 300

(c) Image compressed by PCA with k = 300

(d) Image compressed by our JPEG with Q = 30

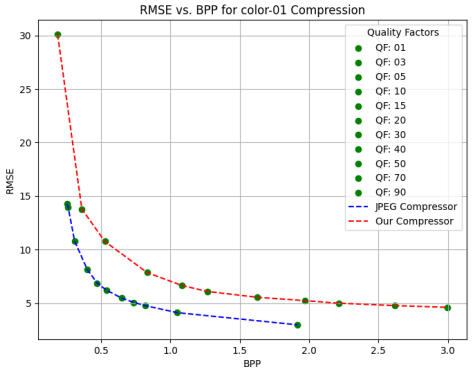
(e) Image compressed by our JPEG with Q = 30

(f) Image compressed by our JPEG with Q = 30

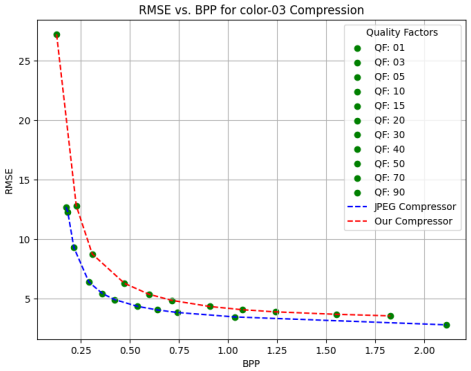
Comparision between PCA and JPEG

9 Comparison with JPEG with our compressor

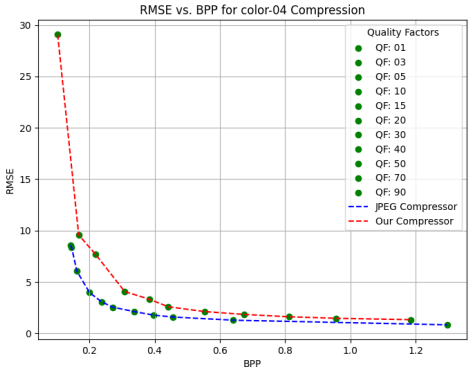
9.1 RMSE vs BPP for Color Images



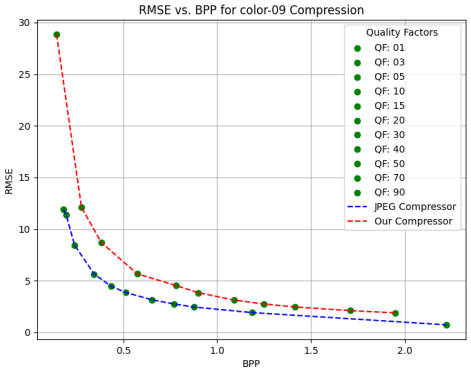
(a) Color image - 1



(b) Color image - 2



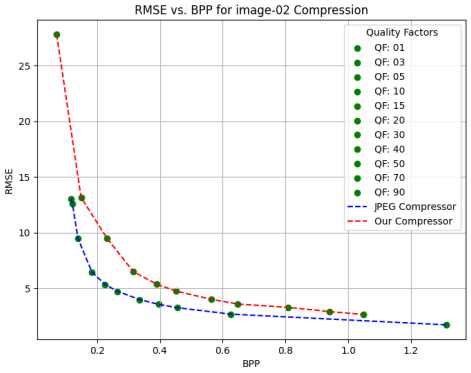
(c) Color image - 3



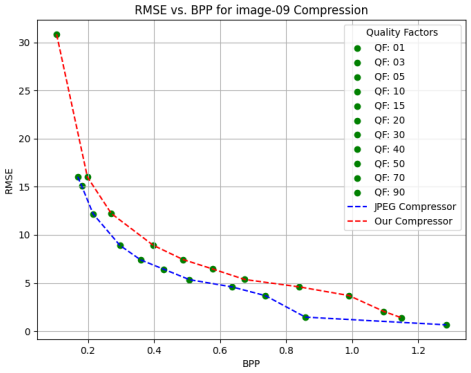
(d) Color image - 4

Comparison with JPEG for color images

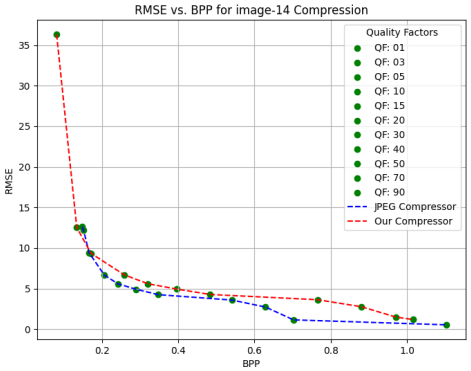
9.2 RMSE vs BPP for Grey Scale Images



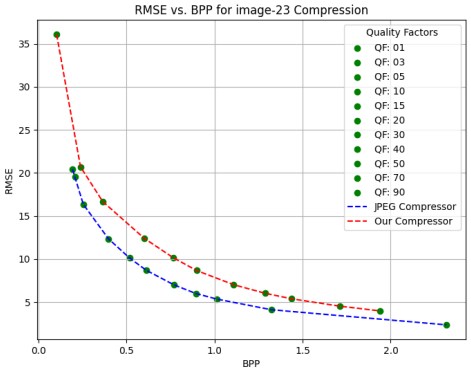
(a) Grey Scale image - 1



(b) Grey Scale image - 2



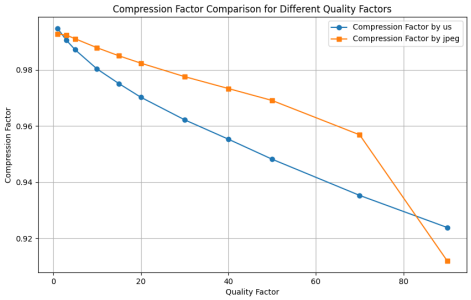
(c) Grey Scale image - 3



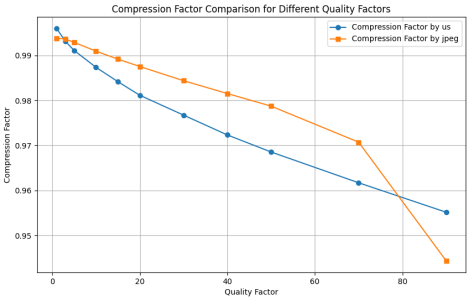
(d) Grey Scale image - 4

Comparison with JPEG for Grey Scale images

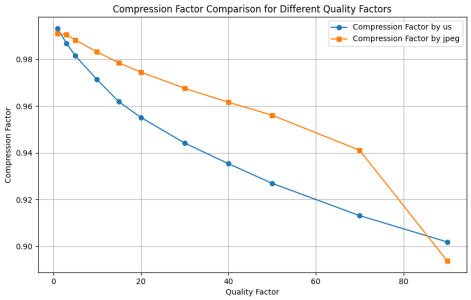
9.3 Compression factor of Color Images



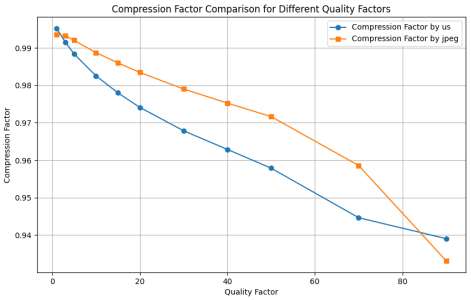
(a) Color image - 1



(b) Color image - 2



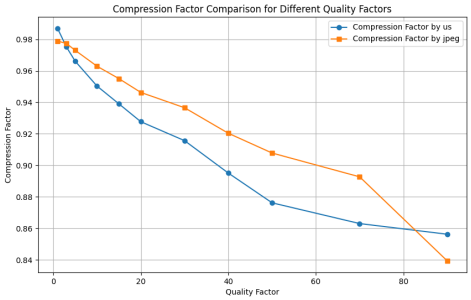
(c) Color image - 3



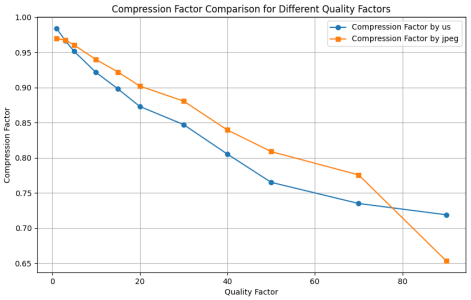
(d) Color image - 4

Comparison with JPEG for color images

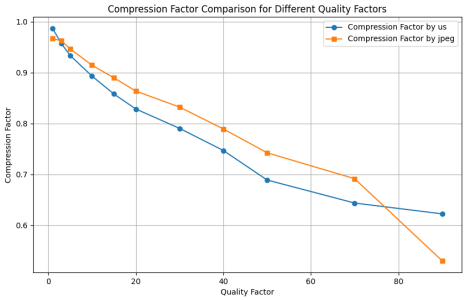
9.4 Compression percentage of Grey Scale Images



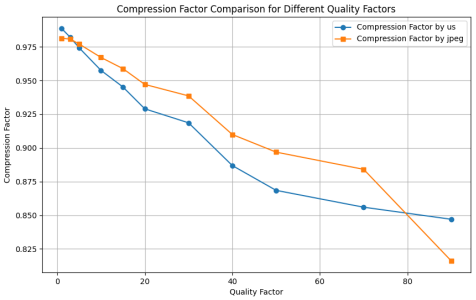
(a) Grey Scale image - 1



(b) Grey Scale image - 2



(c) Grey Scale image - 3



(d) Grey Scale image - 4

Comparison with JPEG for Grey Scale images

9.5 Changing of Quantization Matrix

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	70
14	13	16	24	40	57	80	90
14	17	22	29	51	87	95	100
18	22	37	56	68	109	120	130
24	35	55	64	81	104	140	150
49	64	78	87	103	160	170	180
72	92	95	98	140	160	170	200

New Quantization Matrix

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Old Quantization Matrix

Quality Factor	Input Size	Compressed Size New (Us)	Compressed Size Old (Us)
1	187500	2095	2095
3	187500	4962	4963
5	187500	6924	6946
10	187500	10294	10323
15	187500	12657	12735
20	187500	14688	14830
30	187500	17830	18029
40	187500	20351	20619
50	187500	22569	22895
70	187500	26256	26673
90	187500	28990	29330

Comparison of compressed sizes for quality factors for old and new quantization matrix.

With old quantization matrix

With new quantization matrix

10 Contributions

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

- **Gnana Mahesh:** JPEG implementation of Grayscale image with huffmann encoding, Comparision with JPEG, report.
- **Umesh Karthikeya:** JPEG implementation of Colour images. Conducting experiments, Plotting graphs and automation, report.
- **Varun Gupta:** PCA based image compression,Research paper implementation, Conducting experiments, report.

11 Instructions to Run

- Run `bash main.bash`
- After running the above command,
  - In the `data/output/` a separate folder is created for each image in the `data/input` containing compressed images for different quality factors.
  - Make sure each color image is named as $color - XX.ext$ and each grey scale image is named is $image - XX.ext$ where ext is the extension for that image.
  - And the plots `RMSE vs BPP` for each image is created in the plots directory.
  - In the `data/compress` directory a csv file is created for an image containing the data for the comparison of our compressor and original JPEG compressor
  - And encoding `.bin` file for each image is stored in `data/output/encoded_data` directory
  - In the directory `data/compress` there are two files named `generate_plot.py` and `plot.sh`. Run `plot.sh` file to generate graphs for the comparison of our compressor and JPEG compressor.
- To run the research paper code we need to mention the image name in code manually and then run python3 researchpaper_encoder.py and then run python3 researchpaper_decoder.py
- To run the PCA code run python3 encoder_pca.py to run the code. input will be taken from folders s1,s2,..s40 and output will be in compressed images folder.
- In the inequality $kx - k \geq k \ln x$, the model canceled out k, which should not be done.
- $log(3)ln(3)$ $x = \frac{1}{12}$ Suppose that we have a unit vector $\mathbf{v}$ such that $\mathbf{v} = \begin{pmatrix}\cos\theta & \sin\theta & 0\end{pmatrix}$. Let $\mathbf{w} = \begin{pmatrix}0 & 1 & 2\end{pmatrix}$. We consider a probability distribution on the possible angles θ such that the probability density function of θ is $\frac{1}{2\pi}$. We now define the area A as the area of the shaded region in the plane with normal vector $\mathbf{v}$ and passing through the point $\mathbf{w}$. The boundary of this shaded region is formed by the intersections of this plane with the unit sphere centered at the origin. Find the expected value of the area A.

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.