Figure 1: A compressed Mark Ruffalo

CS 663 Project

Image Compression

Ashwajit Singh 22b1227
Raunak Mukherjee 22b3955
Sreekar Reddy 22b1274

November 25, 2024

Problem Statement

We were asked to build an image compression engine using the JPEG algorithm and then introduce some innovation into the algorithm to attain better results.

JPEG Algorithm

The JPEG algorithm was first implemented as taught in the course. The following steps should give you the general flow of the algorithm:

1. **2D Discrete Tranform:** We divide the image into 8x8 blocks (with mean padding if the height or width are not a multiple of 8 and store the DCT of each block. We divide each element of these 8x8 blocks by the corresponding element in the M matrix (scaled by the quality factor) to quantise the values.

2. **Huffman Encoding:** We take the consecutive differences of the DC coefficients, and run Huffman encoding on this set along with the set of AC coefficients

3. **AC Encoding:** The non-zero AC coefficients are encoded with their run length (number of zeroes following it), the number of bits required to store the Huffman code and the Huffman code itself. Since the run length can be longer than 15 (and often is), we have an additional (15,0,0) block for a set of 15 consecutive zeroes

4. **Writing to file:** We write the height, width, Huffman codes, DC coefficient and AC coefficient bit strings to a file, so it can be decoded

5. **Decoding:** To decode, we simply create a Huffman table from the codes bit string, and use this to determine the DC and AC coefficients, followed by an inverse DCT.

Diffusion Based Compression

The image reconstruction process involves two main steps: the subdivision of the image domain into rectangles and the use of a homogeneous diffusion equation to interpolate pixel values within these rectangles. We base this algorithm on the papers by "Beating the Quality of JPEG 2000 with Anisotropic Diffusion" by Christian Schmaltz, Joachim Weickert, and Andres Bruhn and "Image compression with anisotropic diffusion" by Galic et al.

Rectangular Splitting

The rectangular splitting procedure begins by encoding the boundary of the image using endpoints and midpoints of its edges. The image is then recursively

subdivided into smaller rectangles based on the reconstruction error. Specifically:

- The initial boundary is saved as eight pixels (two endpoints and one midpoint for each of the four edges).
- Reconstruction (using diffusion based interpolation) is performed to compute the Mean Square Error (MSE) between the original image and the current reconstruction.
- If the MSE exceeds a threshold, the rectangle is split along its largest dimension, and the process is repeated for the resulting sub-rectangles.

Encoding Pixel Locations

An added advantage of using the following method rectangular splitting, above and beyond being able to reconstruct the original image from it's scarce version is the added ability to be able to store the pixel locations using a binary tree format. We encode a splitting rectangle using a '1' and a non splitting rectangle using a '0'.

This is preferable to having to store the integer coordinates of the sparse pixels.

Entropy Based Encoding

The intensity values of the pixels along with the aforementioned binary tree are then encoded using an entropy based encoder. We have decided to use a Huffmann encoder to this end to further reduce the size of the image without incurring any losses.

Homogeneous Diffusion for Pixel Interpolation

This method distributes the known pixel values across the domain, leveraging the Laplacian operator to ensure uniform smoothing.

The homogeneous diffusion equation is expressed as:

$$\frac{\partial u}{\partial t} = \Delta u, \tag{1}$$

where Δu represents the Laplacian of the image $u(x, t)$, defined as:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \tag{2}$$

Boundary Conditions and Initialization

To reconstruct the image, Dirichlet boundary conditions are applied using the known pixel values:

$$u(x, t) = G(x), \quad x \in K, \tag{3}$$

where $G(x)$ is the function specifying the pixel values at the subset $K \subset \Omega$ (the image domain). For the remaining region $\Omega \setminus K$, the initial condition is typically set to zero:

$$u(x, 0) = 0, \quad x \in \Omega \setminus K. \tag{4}$$

Steady-State Solution

The interpolation is performed by evolving the homogeneous diffusion equation until it reaches a steady state:

$$R(x) = \lim_{t \to \infty} u(x, t), \tag{5}$$

where $R(x)$ is the reconstructed image. At steady state, the Laplacian vanishes:

$$\Delta R(x) = 0, \quad x \in \Omega \setminus K, \tag{6}$$

ensuring that the interpolated regions are smooth and consistent with the known pixel values at the boundaries.

Algorithmic Overview

The procedure for reconstruction is summarized as:

1. Initialize $u(x, 0) = 0$ for $x \in \Omega \setminus K$ and $u(x, t) = G(x)$ for $x \in K$.
2. Iteratively solve $\frac{\partial u}{\partial t} = \Delta u$ using a numerical scheme such as finite differences until convergence.
3. Obtain the steady-state solution $R(x)$ as the reconstructed image.

This approach produces a smooth interpolation that adheres to the known pixel values while propagating information to the unknown regions.

JPEG on Colour

To implement the JPEG algorithm on a coloured picture, we start by bringing our coloured image into the YCbCr format and then applying the standard JPEG algorithm on each of the Y, Cb and Cr values treating them as individual black and white images. We can further down-sample the Cb and Cr parts of the image and still retain quality since the human eye is less sensitive to the chrominance values.

Given the RGB values of a coloured image, you can convert it to YCbCr using the following relation:

$$
\begin{aligned}
Y &= 16 + (65.481R + 128.553G + 24.966B) \\
C_B &= 128 + (-37.79R - 74.203G + 112B) \\
C_R &= 128 + (112R - 93.786G - 18.214B)
\end{aligned}
$$

Description of Testing Dataset

Both the Diffusion compression algorithm and the JPEG algorithm we implemented were tested on a public domain image set of Mark Ruffalo images, and their RMSE were compared.

This particular dataset was chosen since it's file size and pixel dimensions are highly convenient, in addition to not already being jpg and being grayscale. An addition reason is that human face images are a good test of compression algorithms to the human eye since humans are highly sensitive to even small variations in facial features.

Good and Bad Aspects of the Algorithm

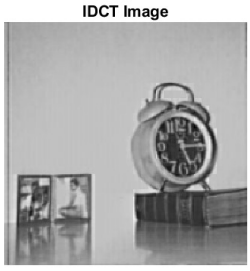
We can observe the block-seam and ringing artifacts created by the JPEG algorithm, and the blurred edges in the implementation of R-EED as expected. Our implementation of JPEG resulted in good compression, but was computationally inefficient (as implemented in MATLAB), which could probably be improved by enabling more parallel processing.

The diffusion based algorithm worked well on some images and not so well on other images (and in rare cases - matches or increases file size) since the rectangular splitting algorithm and the number of bits needed to store depends on the pixel dimensions and is highly discrete.

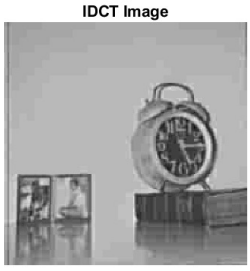
Results using JPEG

(a) Q = 10

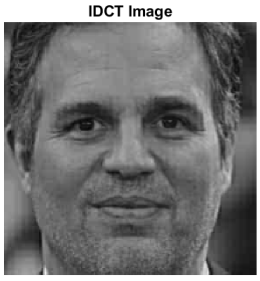
(b) Q = 100



(c) Q = 10



(d) Q = 100

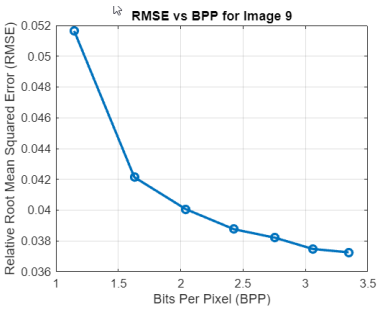


(e) Q = 10

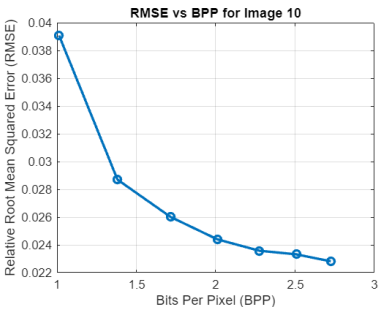
(f) Q = 100

Figure 2: JPEG compression of an aerial photo with two different quality factors

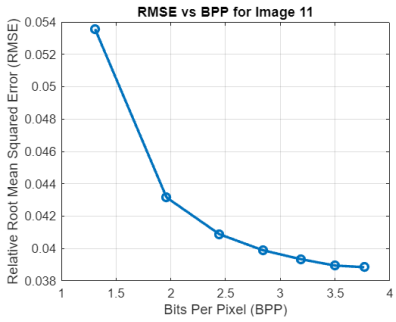
RMSE vs BPP



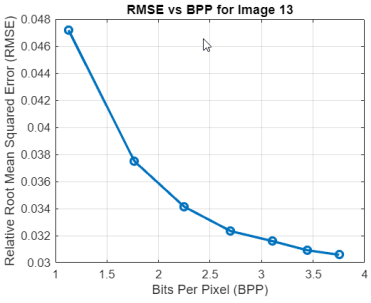
(a) Image 1



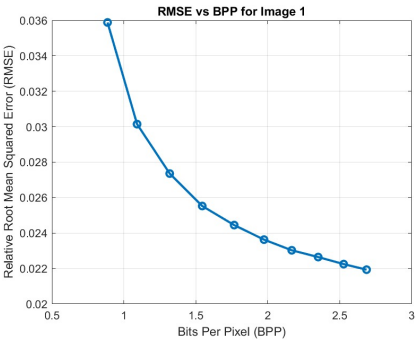
(b) Image 2



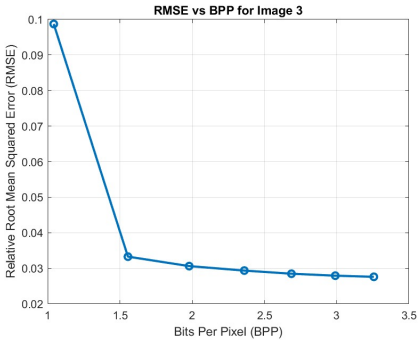
(c) Image 3



(d) Image 4

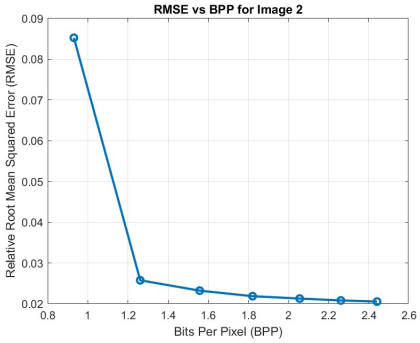


(e) Image 5

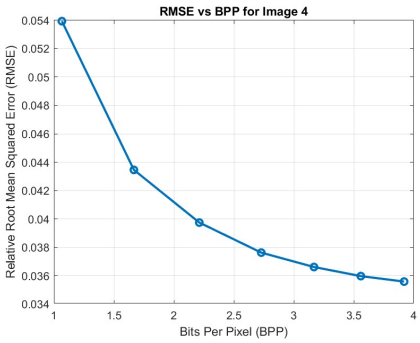


(f) Image 6

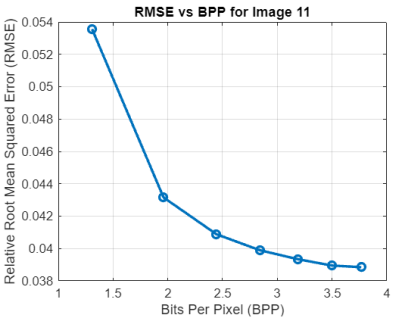
Figure 3: RMSE vs BPP for various Ruffalos and misc objects



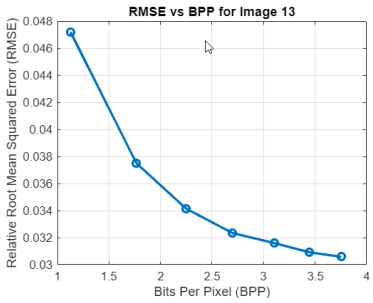
(a) Image 1



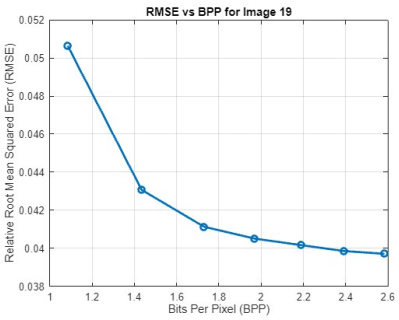
(b) Image 2



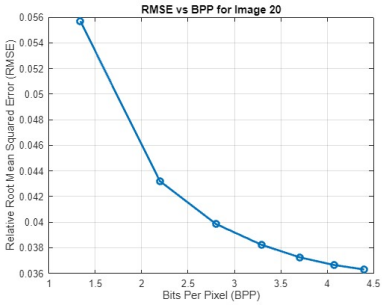
(c) Image 3



(d) Image 4



(e) Image 5

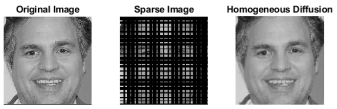


(f) Image 6

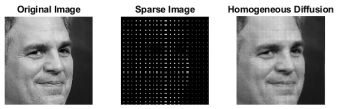
Figure 4: RMSE vs BPP for various Ruffalos and misc objects

Reconstructing Sparse Images using Homogenous Diffusion

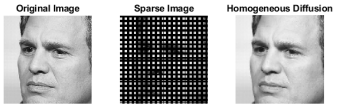
An extensive test was conducted on Mark Ruffalo. The original image (left) is compressed to form a Sparse version (middle) which is reconstructed using Homogenous Diffusion and displayed (right).



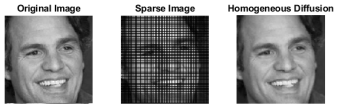
(a) mark_12



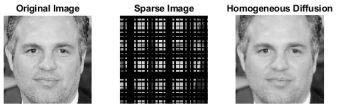
(b) mark_14



(c) mark_15



(d) mark_16

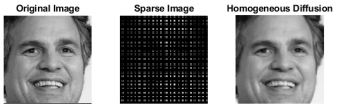


(e) mark_18

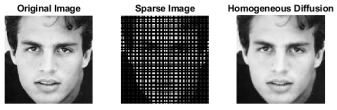
(f) mark_20

(g) mark_4

(h) mark_5



(i) mark_7



(j) mark_8

Figure 5: Cropped images arranged as subfigures.