Part A: Image Compression: JPEG Algorithm Implementation

Problem Statement

The project involves designing an image compression engine based on the JPEG algorithm. The goal is to compress grayscale images.

Algorithm Description

1. **Preprocessing**:
    - **Input Image:** Loaded using `cv2.imread()` in grayscale mode (`cv2.IMREAD_GRAYSCALE`).
    - **Resizing:** Dimensions clipped to multiples of 8 using slicing: `image[:height // 8 * 8, :width // 8 * 8]`.
    - **Centering:** Pixel values shifted by subtracting 128 for DCT compatibility: `image - 128`.
2. **Discrete Cosine Transform (DCT)**:
    - **Blocking:** Image split into 8×88 \times 88×8 blocks using nested loops and array slicing.
    - **DCT Application:** Performed on each block using `scipy.fftpack.dct()` with `norm='ortho'`.
3. **Quantization**:
    - **Quantization Matrix:** Generated from a standard JPEG matrix and scaled by the quality factor QQQ.
    - **Coefficient Quantization:** `block / quant_matrix` followed by `np.round()`.
    - **Zigzag Order:** used to rearrange coefficients using zigzag matrix
4. **Run-Length Encoding (RLE)**
    - Encodes coefficients as (run-length, value) pairs. Implemented using a custom `run_length_encode()` function that tracks zeros and inserts End of Block (EOB) markers for trailing zeros.
5. **Huffman Encoding**:
    - **Frequency Tables:** Constructed from quantized coefficients
    - **Huffman Tree:** Built using a custom `build_huffman_tree()` function.
    - **Encoding:** DC differences and AC RLE pairs encoded into bitstreams using `huffman_encode()` function.
6. **Storage and Retrieval**:
    - Saves the compressed image into a **bitstream** format, including:
        - Quantization matrix, Huffman tables, and encoded coefficients.
    - Reads the bitstream to reconstruct the image:
        - Decodes Huffman bitstreams for DC and AC coefficients.
        - Reconstructs image blocks via inverse DCT.
7. **Error and Compression Metrics**:
    - **RMSE**: Measures the error between original and decompressed images.
    - **BPP**: Measures the average number of bits per pixel in the compressed image.

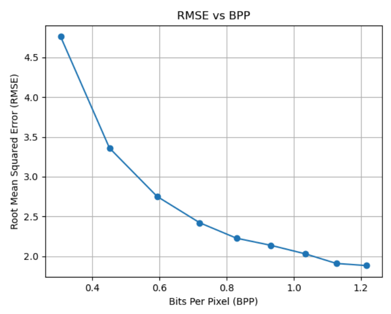
Dataset Description

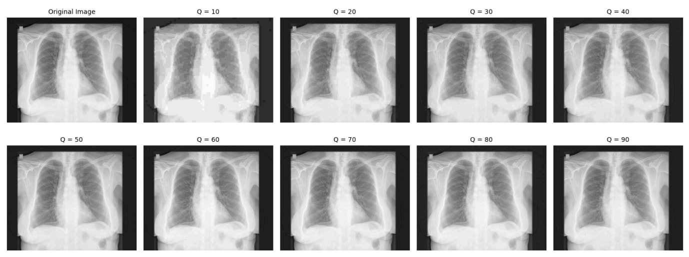
- **Input**: 20 grayscale images of varying dimensions and content.
- Segmentation evaluation database - 200 gray level images along with ground truth segmentations
- Few miscellaneous images

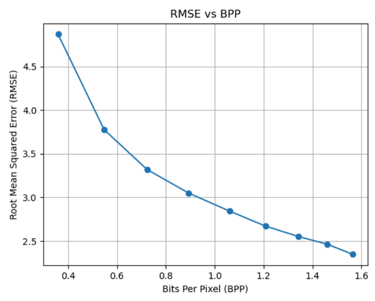
Analysis

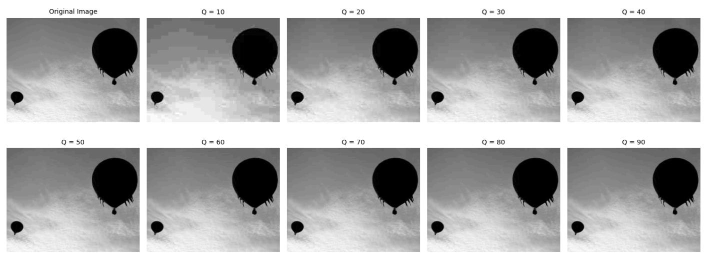
1. RMSE VS BPP graphs, 21 images, tested for 9 Q values (10, 20, 30, 40, 50, 60, 70, 80, 90)
2. All graphs monotonically decreasing
3. 9/20 images - almost same slope between consecutive Q values

Examples:









Reason:

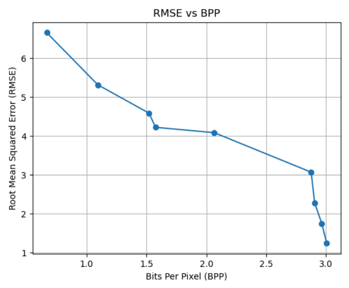
For these images, the compression algorithm applies a similar degree of quantization across the quality factor range, leading to a relatively uniform trade-off between compression efficiency (BPP) and quality loss (RMSE). This indicates that:

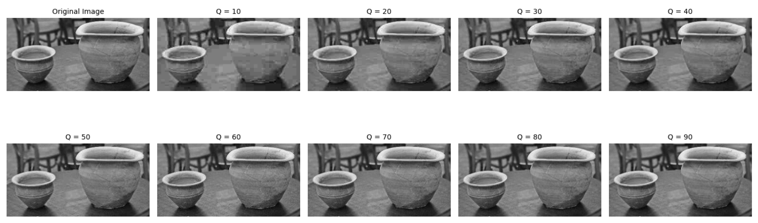
1. The images likely have uniform textures or low complexity.
2. These images may have low spatial frequency components (e.g., smooth gradients or fewer sharp edges), which are less affected by aggressive quantization.

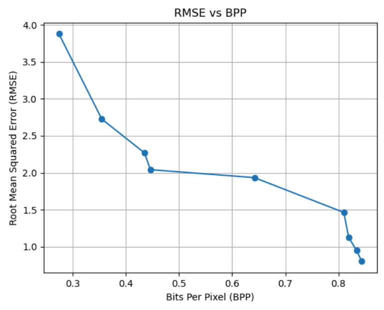
3. Quantization tables remain effective at preserving detail, resulting in a consistent decrease in RMSE as Q increases.

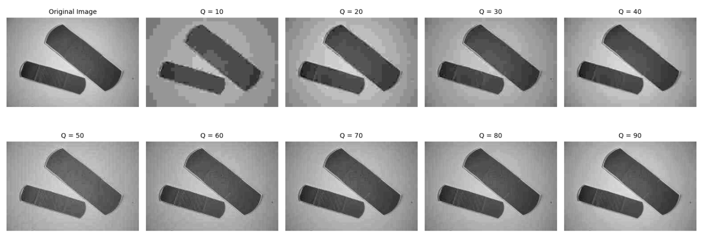
4. 11/20 images - from Q = 30 to Q = 40 (sharp decrease in gradient), Q = 30 to Q = 60 (gradual decrease), Q = 60 to Q = 90 (sharp decrease)

Examples:









Reason:
Quantization Sensitivity:
JPEG compression heavily relies on quantization tables to reduce file size, but the rate of change in RMSE vs. BPP depends on the image's frequency content

- **Q = 30 to Q = 40 (Sharp Decrease)**:
  - At low quality factors (like Q=30), compression is aggressive, and a small increase in Q leads to a significant improvement in image quality (sharp decrease in RMSE).
  - This is because large quantization steps for Q=30 introduce significant artifacts (e.g., blocking), and slightly finer quantization (at Q=40) reduces these artifacts drastically.
- **Q = 40 to Q = 60 (Gradual Decrease)**:
  - Between these values, the quantization process becomes less aggressive, and subsequent increases in Q result in diminishing improvements in quality (gradual decrease in RMSE).

    - The compression starts reaching diminishing returns because most of the large artifacts have already been mitigated, and further improvements in detail retention become subtler.
- **Q = 60 to Q = 90 (Sharp Decrease)**:
    - At high Q values, the quantization steps become very fine, preserving even minor details. However, this leads to sharp increases in file size (higher BPP) while significantly reducing RMSE (sharp decrease).
    - This range is where perceptible image quality closely approaches the original.

5. The common behavior from **Q=10 to Q=30** occurs because aggressive quantization eliminates high-frequency details, leading to a significant RMSE reduction as Q increases and retains more essential image information.

Results - https://github.com/kavanvavadiya/cs663-results

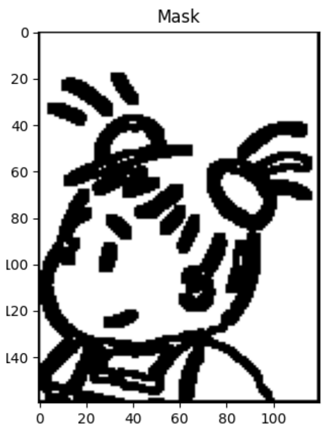
Part B: EDGE BASED IMAGE COMPRESSION

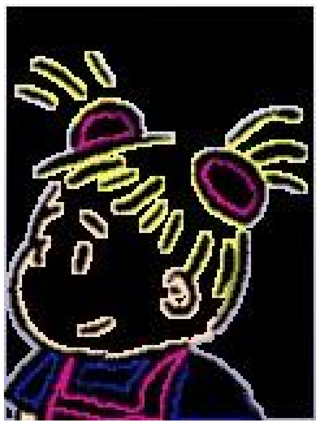
Introduction:

In this part we have used a lossy compression method for cartoon like images that exploits information at image edges. These edges are extracted using the canny edge algorithm. Their locations are stored in a lossless way using **JBIG**. Moreover, we encode the grey or colour values at both sides of each edge by applying subsampling and PAQ coding. 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.

Encoding:

Edge detection: Edge detection algorithms such as Marr Hildreth were used but results were poor, so canny edge algorithm is used.





Edge Image (image which contains contours found by canny edge) is encoded and compressed using **JBIG**. It has been developed as a specialised routine for lossless as well as for lossy compression of bi-level images as mentioned in the paper.

	Image 1	Image 2	Image 3	Image 4	Image 5
Original image size (KB)	20.6	142	193	121	71
JBIG compressed edge image (KB)	0.521	1	3	0.876	2

Storing the encoding data:

Residual Image is encoded and compressed using PAQ compressor along-with the JBIG file to get one compressed version of the image file

Decoding :

Decoding the contour pixel values:

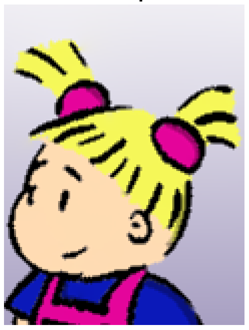
The Archive is decompressed using PAQ decompressor and then the edge image is restored using JBIG decoder.

Reconstructing Missing Data:

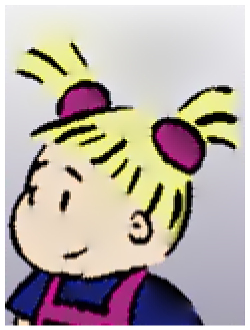
The missing pixels are reconstructed (image in-painting) is performed using homogenous diffusion for interpolation.

$$\partial_t u = \nabla u$$

The reconstructed data satisfies the Laplace equation $\nabla u = 0$.
Such a PDE can be discretised in a straightforward way by finite differences.
We tried anisotropic diffusion but could not make it work



Original Image



Restored Image

Evaluation:

PSNR value is calculated to measure similarity between restored image and original image.

	PSNR	Original image size (KB)	Compressed archive size (KB)	Compression ratio
Image 1	21.47	20.6	15.4	1.34
Image 2	15.29	142	47.7	2.98
Image 3	9.44	193	75	2.57
Image 4	18.54	121	32.9	3.68
Image 5	18.61	71	52.8	1.34

Analysis

It works only for cartoon like images, because of the assumption that area inside a boundary has same colour, which is mostly observed in cartoon images. Also the difference in reconstructed and original image is visible. The compression ratios are high.

Contribution:

Name	Roll Number	Contribution
Anshika Raman	210050014	Jpeg Compression - basic implementation and report Edge based image compression - basic implementation
Kavan Vavadiya	210100166	Edge based image compression - Final Implementation and report Jpeg Compression - basic implementation
Kushal Agarawal	210100087	Jpeg compression - debugging, final Implementation Edge based image compression - basic implementation

References:

- PDEs for Image Interpolation and Compression - Joachim Weickert
- [PDF] Understanding and advancing PDE-based image compression | Semantic Scholar
- Edge-Based Image Compression with Homogeneous Diffusion