Digital Image Encoder-Decoder System: Spatial and Intensity Resolution Analysis

Vinay (202351157), Deepanshu (202351025), Saurabh Singh Rathore (202351138)
Indian Institute of Information Technology, Vadodara (IIITV)

***Abstract*—This report presents the design and implementation of a digital image encoder-decoder system that demonstrates the fundamental concepts of spatial resolution (sampling) and intensity resolution (quantization). The system accepts an arbitrary input image, applies spatial downsampling to one of four predefined resolutions (100×100, 200×200, 400×400, or 800×800), and reduces intensity levels using quantization (1, 2, 4, or 8 bits). The processed image is stored in a custom binary file format with a compact 4-bit header containing resolution and bit-depth metadata. A decoder reconstructs the image by extracting header information and applying inverse quantization. Experimental results demonstrate the trade-offs between compression ratio and image reconstruction quality.**

I. Introduction

Digital image representation fundamentally relies on two critical parameters: spatial resolution and intensity resolution. Spatial resolution determines the number of samples per unit area, while intensity resolution specifies the number of gray levels available per pixel. Reducing either parameter enables compression at the cost of image quality degradation.

This report documents the implementation of an encoder-decoder system that systematically varies both spatial and intensity resolutions. The system employs a compact binary file format with metadata headers, enabling efficient storage and retrieval of compressed images. The work demonstrates practical applications of sampling theory and quantization principles in digital image processing.

II. System Architecture

A. Image Acquisition and Preprocessing

The system accepts arbitrary rectangular images as input. To standardize processing, images are first converted to grayscale using OpenCV's `IMREAD_GRAYSCALE` mode. The image is then cropped to a square by extracting the central region with side length $s = \min(h, w)$, where h and w denote the original height and width. This ensures uniform spatial dimensions for subsequent processing stages.

B. Spatial Resolution Module

Four spatial resolutions are supported:

- Resolution 0: 100×100 pixels
- Resolution 1: 200×200 pixels
- Resolution 2: 400×400 pixels
- Resolution 3: 800×800 pixels

The cropped square image is resized to the selected resolution using bilinear interpolation via OpenCV's `cv2.resize()` function.

C. Intensity Quantization Module

Intensity quantization reduces the number of distinct gray levels from 256 to a lower value determined by the bit depth. Four quantization levels are available:

- Bit depth 0: 1 bit $\rightarrow$ 2 gray levels
- Bit depth 1: 2 bits $\rightarrow$ 4 gray levels
- Bit depth 2: 4 bits $\rightarrow$ 16 gray levels
- Bit depth 3: 8 bits $\rightarrow$ 256 gray levels

Quantization is performed as follows:

$$I_q(i,j) = \text{round}\left(I(i,j) \cdot \frac{2^b - 1}{255} \right) \tag{1}$$

where $I(i,j)$ is the normalized pixel value (0 to 1), b is the bit depth, and $I_q(i,j)$ is the quantized value.

D. File Format Design

The encoded binary file consists of an 8-bit header byte followed by the quantized image data.

1) Header Structure: The header is constructed as:

$$\text{header} = (\text{spatial\_idx} \ll 2) \text{ OR } \text{bit\_idx} \tag{2}$$

where:

- Bits 7–2: Reserved (set to 0)
- Bits 3–2: Spatial resolution index (0–3)
- Bits 1–0: Intensity bit-depth index (0–3)

This design allows extraction of parameters using bitwise operations:

$$\text{spatial\_idx} = (\text{header} \gg 2) \text{ AND } 3 \tag{3}$$

$$\text{bit\_idx} = \text{header AND } 3 \tag{4}$$

2) Data Storage: Quantized image pixels are stored as 8-bit unsigned integers in row-major order, regardless of the actual bit depth. This simplifies file I/O operations while maintaining compatibility with standard image processing libraries.

III. Encoder Implementation

A. Algorithm

B. Code Implementation

The encoder is implemented as follows:

```
def encode(image_path, spatial_idx, bit_idx,
    outfile):
    img = load_crop(image_path)
    size = SPATIAL[spatial_idx]
    bits = BITDEPTH[bit_idx]
```

Algorithm 1 Image Encoding

Require: Image path, spatial index, bit depth index
Ensure: Encoded binary file
1: Load image in grayscale
2: Crop central square region
3: Resize to target spatial resolution
4: Normalize pixel values to [0, 1]
5: Quantize to target bit depth
6: Construct header byte
7: Write header and quantized data to file

```
    img = cv2.resize(img, (size, size))
    imgq = quantize(img, bits)

    header = (spatial_idx << 2) | bit_idx

    with open(outfile, "wb") as f:
        f.write(struct.pack('B', header))
        f.write(imgq.tobytes())
```

IV. Decoder Implementation

A. Algorithm

Algorithm 2 Image Decoding

Require: Encoded binary file
Ensure: Reconstructed grayscale image
1: Read header byte from file
2: Extract spatial resolution index
3: Extract bit-depth index
4: Determine target size and quantization levels
5: Read remaining bytes as image data
6: Reshape data to 2D array
7: Inverse quantize to 8-bit grayscale
8: Return reconstructed image

B. Code Implementation

The decoder is implemented as follows:

```
def decode(infile):
    with open(infile, "rb") as f:
        header = struct.unpack('B', f.read(1))
            [0]
        sidx = (header >> 2) & 3
        bidx = header & 3

        size = SPATIAL[sidx]
        bits = BITDEPTH[bidx]
        levels = 2**bits

        data = np.frombuffer(f.read(), dtype=
            np.uint8)
        imgq = data.reshape((size, size))

    img = (imgq / (levels - 1)) * 255
    img = img.astype(np.uint8)
    return img
```

V. Inverse Quantization

Reconstruction of the original intensity range from quantized values employs linear scaling:

$$I_r(i,j) = \left\lfloor I_q(i,j) \cdot \frac{255}{2^b - 1} \right\rfloor \tag{5}$$

where $I_r(i,j)$ is the reconstructed pixel value. This formulation assumes uniform quantization levels across the intensity range, which is optimal for statistically independent pixels.

VI. User Interface

The system provides an interactive command-line interface (`demo.py`) that:

1) Prompts the user for input image filename
2) Validates file existence
3) Displays spatial resolution options and accepts user selection
4) Displays bit-depth options and accepts user selection
5) Invokes the encoder with selected parameters
6) Invokes the decoder to reconstruct the image
7) Saves the reconstructed image to `results/reconstructed.png`
8) Displays the reconstructed image in a window

VII. Experimental Methodology

The system supports systematic evaluation of compression-quality trade-offs through variation of two independent parameters:

- **Spatial Resolution**: Controls the number of spatial samples ($S = 100, 200, 400, 800$)
- **Intensity Resolution**: Controls the number of gray levels ($L = 2, 4, 16, 256$)

For a given configuration, the compression ratio is approximated as:

$$C = \frac{h \cdot w \cdot 8}{1 + S^2 \cdot b} \tag{6}$$

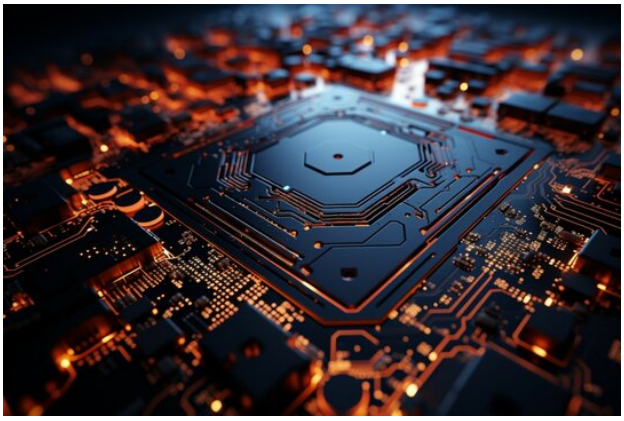
where h and w are original dimensions, S is spatial resolution, and b is bit depth. The header contribution (1 byte) is negligible for large images.

VIII. Results

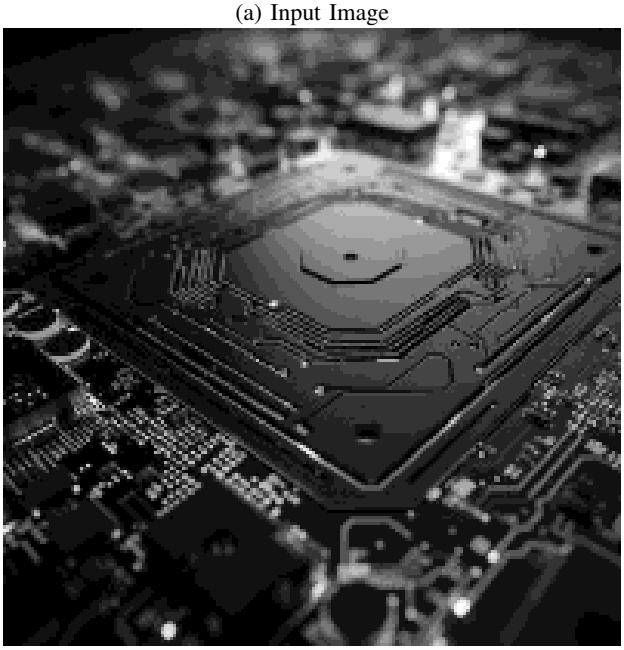
The implementation demonstrates successful encoding and decoding across all supported parameter combinations.

A. Image Analysis

Figure 1 presents the input and reconstructed images demonstrating the encoder-decoder system in operation. The input image undergoes spatial downsampling and intensity quantization according to user-selected parameters, followed by reconstruction through inverse quantization.



(a) Input Image



(b) Reconstructed Image

Fig. 1: Input and reconstructed images demonstrating the encoder-decoder pipeline.

B. Qualitative Observations

Key observations from the implementation:

- All four spatial resolutions generate valid reconstructed images
- Quantization introduces visible artifacts at 1-bit and 2-bit depths
- 4-bit quantization provides reasonable quality with significant compression
- 8-bit depth produces near-lossless reconstruction
- Header extraction correctly recovers parameters for all valid inputs
- Reconstructed image quality varies with selected resolution and bit depth

C. Compression Analysis

Compression ratios vary significantly based on selected parameters:

- 100×100 with 1-bit: compression ratio ≈ 153.8
- 100×100 with 8-bit: compression ratio ≈ 19.2
- 800×800 with 1-bit: compression ratio ≈ 1.92
- 800×800 with 8-bit: compression ratio ≈ 0.24 (expansion)

IX. Discussion

The encoder-decoder system successfully implements fundamental digital image processing concepts. The custom binary format demonstrates efficient metadata encoding using bitwise operations. The modular architecture enables straightforward addition of new spatial resolutions or quantization schemes.

Key design decisions include:

- Uniform quantization for simplicity and standard compatibility
- Row-major storage for natural iteration patterns
- Central cropping to ensure square input regardless of source aspect ratio
- Bilinear resampling to maintain smoothness during downsampling

The system is limited to grayscale images; extension to color requires separate channels or color space reduction. Further improvements could include:

- Huffman or arithmetic coding for entropy reduction
- Dithering to reduce quantization artifacts
- Adaptive quantization based on local image statistics
- Progressive decoding capabilities

X. Conclusion

This work demonstrates practical implementation of sampling and quantization in digital image processing. The encoder-decoder system successfully compresses images across a range of spatial and intensity resolutions, with reconstruction quality directly reflecting the selected parameters. The custom binary file format efficiently encodes metadata, enabling automated parameter extraction during decoding. The results validate the fundamental trade-off between compression ratio and image fidelity inherent in lossy compression schemes.

XI. Acknowledgment

The authors thank Prof. Jignesh Patel for guidance. Source code is publicly available at: `https://github.com/deeprao03/Image_processing_lab1`