Homework #4 – Entropy Coding

312553024 江尚軒

Code

Use a raster scan to visit all 8x8 blocks in these images.

```python
# Divide image into 8x8 blocks
print(">> Divide image into 8x8 blocks...")
blocks = []
for i in range(0, height, 8):
    for j in range(0, width, 8):
        block = image[i:i+8, j:j+8]
        blocks.append(block)
```

8x8 block-based DCT coefficients of “lena.png.”

```python
# Implement custom DCT for an 8x8 block
Tabnine | Edit | Test | Explain | Document | Ask
def dct_2d(block):
    """Apply 2D DCT to an 8x8 block."""
    M, N = block.shape
    c = np.array([1 / np.sqrt(2) if i == 0 else 1 for i in range(max(M, N))])
    x = np.arange(M).reshape(-1, 1)
    y = np.arange(N).reshape(1, -1)

    dct = np.zeros((M, N))
    for u in range(M):
        for v in range(N):
            cos_x = np.cos((2 * x + 1) * u * np.pi / (2 * M))
            cos_y = np.cos((2 * y + 1) * v * np.pi / (2 * N))
            dct[u, v] = (2 / N) * c[u] * c[v] * np.sum(block * cos_x * cos_y)

    return dct
```

Quantize the coefficients with the two quantization tables.

```python
# Quantize using quantization tables
Tabnine | Edit | Test | Explain | Document | Ask
def quantize(block, q_table):
    """Quantize an 8x8 DCT-transformed block."""

    return np.round(block / q_table).astype(np.int32)
```

Do the run length encoding by using a zigzag scan to visit all pixels in one block.

```python
# Run Length Encoding
Tabnine | Edit | Test | Explain | Document | Ask
def run_length_encode(block):
    """Run-Length Encode a 1D block after zigzag scan."""
    zigzag_array = zigzag_scan(block)
    rle = []
    zero_count = 0

    for i in range(len(zigzag_array)):
        if zigzag_array[i] == 0:
            zero_count += 1
        else:
            if zero_count != 0:
                rle.append((0, zero_count))
                zero_count = 0
            rle.append((zigzag_array[i], 1))

    if zero_count > 0:
        rle.append((0, zero_count))

    return rle
```

Do the run length decoding and IDCT to recover the image.

```python
# Run Length Decoding
Tabnine | Edit | Test | Explain | Document | Ask
def run_length_decode(rle):
    """Run-Length Decode a list into an 8x8 block using zigzag scan."""
    flat_block = np.zeros(64, dtype=np.int32)
    position = 0

    for value, count in rle:
        for _ in range(count):
            flat_block[position] = value
            position += 1

    # Use zigzag order to reconstruct the 8x8 block
    block = np.zeros((8, 8), dtype=np.int32)
    for i in range(64):
        index = np.unravel_index(zigzag_indices[i], (8, 8))
        block[index] = flat_block[i]

    return block
```

```python
# Inverse Quantize
Tabnine | Edit | Test | Explain | Document | Ask
def inverse_quantize(block, q_table):
    """Inverse quantize the DCT coefficients."""

    return np.multiply(block, q_table)
```

```python
# Implement custom IDCT for an 8x8 block
Tabnine | Edit | Test | Explain | Document | Ask
def idct_2d(block):
    """Apply 2D IDCT to an 8x8 block."""
    M, N = block.shape
    c = np.array([1 / np.sqrt(2) if i == 0 else 1 for i in range(max(M, N))])
    u = np.arange(M).reshape(-1, 1)
    v = np.arange(N).reshape(1, -1)

    idct = np.zeros((M, N))
    for x in range(M):
        for y in range(N):
            cos_u = np.cos((2 * x + 1) * u * np.pi / (2 * M))
            cos_v = np.cos((2 * y + 1) * v * np.pi / (2 * N))
            idct[x, y] = (2 / N) * np.sum(c[u] * c[v] * block * cos_u * cos_v)

    return np.clip(idct, 0, 255)
```

Comparison

Quantization Table JPEG* is an experiment in which I additionally used the JPEG standard quantization table.

Encoded Size

Quantization Table 1	Quantization Table 2	Quantization Table JPEG*
921348 bytes	537933 bytes	672389 bytes

After compression, the encoded size of Quantization Table 2 is the smallest, followed by Quantization Table JPEG, and the largest is Quantization Table 1.

Running Time

Quantization Table 1	Quantization Table 2	Quantization Table JPEG*
4.84 seconds	4.84 seconds	4.84 seconds

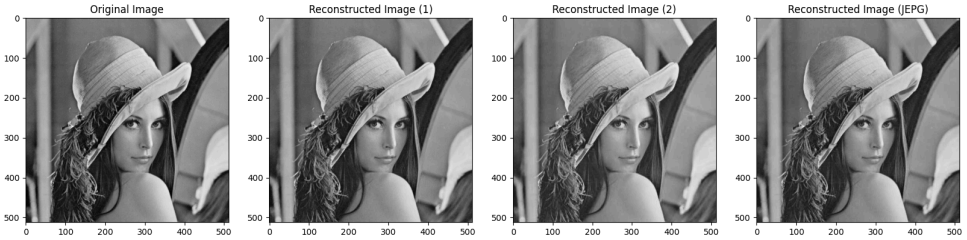
The running times are all the same, which is reasonable since only the Quantization Table differs.

PSNR

Quantization Table 1	Quantization Table 2	Quantization Table JPEG*
37.40 dB	35.24 dB	36.15 dB

After calculating the PSNR between the reconstructed images and the original image, I found that Quantization Table 1 has the highest PSNR, Quantization Table 2 has the lowest, and Quantization Table JPEG's PSNR is in the middle. This suggests that the smaller the compressed image, the lower the quality of the reconstructed image.

Reconstructed Image



After comparing the original image with the reconstructed images, I observed that Quantization Table 1 is the closest to the original image, whereas Quantization Table 2 lacks some shadows and details. Quantization Table JPEG strikes a balance between the two.