Homework 2 : 2D-DCT

Name: 周睦鈞

Student ID: 311553060

Spec:

- 2D-DCT
  - Implement 2D-DCT to transform “lena.png” to DCT coefficients (visualize in log domain).
    - Convert the input image to grayscale first.
    - Visualize the coefficients in the log domain. Feel free to scale and clip the coefficients for visualization.
  - Implement 2D-IDCT to reconstruct the image.
  - Evaluate the PSNR.
- Two 1D-DCT
  - Implement a fast algorithm by two 1D-DCT to transform “lena.png” to DCT coefficients.
- Compare the runtime between 2D-DCT and two 1D-DCT.
- Do **not** use any functions for DCT and IDCT, e.g., cv2.dct
  - Although, you can still use these functions to validate your output.
- Deadline: 2024/10/14 1:19 PM
- Upload to E3 with required files :
  - **VC_HW2_[student_id].pdf**: Report PDF
  - **VC_HW2_[student_id].zip**: Zipped source code (C/C++/Python/MATLAB) and a **README** file

2D-DCT

Two-dimensional DCT & IDCT

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

$$\text{Where } C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases}$$

$0 \le x, y, u, v \le N-1, N^2$: frame size

Two 1D-DCT

One-dimensional DCT & IDCT

$$F(u,v) = \sqrt{\frac{2}{N}} C(u) \sum_{x=0}^{N-1} f(x) \cos\frac{(2x+1)u\pi}{2N}$$

$$f(x,y) = \sqrt{\frac{2}{N}} \sum_{u=0}^{N-1} C(u)F(u) \cos\frac{(2x+1)u\pi}{2N}$$

$$\text{Where } C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u = 0 \\ 1 & \text{otherwise} \end{cases}$$

$0 \le x, u \le N-1, N^2$: frame size

Programming Description:

1. Implementation

Based on the 2D-DCT and two 1D-DCT formulas, I implemented my code as shown below.

However, the runtime of the 2D-DCT is slow due to the nested for loops, so I used matrix multiplication to compute the DCT instead.

```python
def origin_dct_2d(image):
    M, N = image.shape
    dct_result = np.zeros((M, N))
    x_vals = np.arange(M).reshape(-1,1)
    y_vals = np.arange(N).reshape(1, -1)
    for u in range(M):
        for v in range(N):
            sum = np.sum(image * np.cos((2*x_vals+1)*u*np.pi/(2*M)) * np.cos((2*y_vals+1)*v*np.pi/(2*N)))
            cu = np.sqrt(2/M) * np.sqrt(1/2) if u == 0 else np.sqrt(2/M)
            cv = np.sqrt(2 / N) * np.sqrt(1 / 2) if v == 0 else np.sqrt(2 / N)
            dct_result[u, v] = cu * cv * sum
    return dct_result
```

```python
def dct_2d(image):
    """Compute the 2D DCT of an image using numpy's optimized functions."""
    M, N = image.shape

    # Create the cosine transform matrix
    x = np.arange(M)
    y = np.arange(N)
    u = x.reshape(-1, 1)
    v = y.reshape(-1, 1)

    cos_u = np.sqrt(2 / M) * np.cos((2*x + 1) * u * np.pi / (2*M))
    cos_v = np.sqrt(2 / N) * np.cos((2*y + 1) * v * np.pi / (2*N))

    cos_u[0] /= np.sqrt(2)
    cos_v[0] /= np.sqrt(2)

    # Compute DCT using matrix multiplication
    dct_matrix = cos_u @ image @ cos_v.T

    return dct_matrix
```

```python
def dct_1d(signal):
    """Compute the 1D DCT of a signal using numpy's optimized functions."""
    N = len(signal)
    n = np.arange(N)
    k = n.reshape(-1, 1)

    M = np.sqrt(2 / N) * np.cos((2*n + 1) * k * np.pi / (2*N))
    M[0] /= np.sqrt(2)

    return M @ signal
```

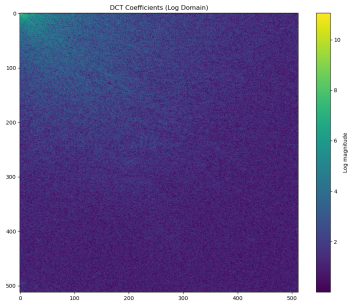
2. Reconstruction Results

2D-DCT

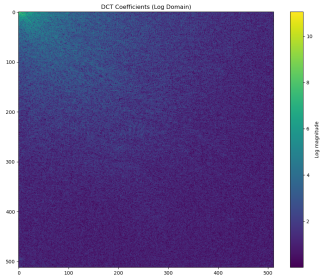
Two 1D-DCT

OpenCV

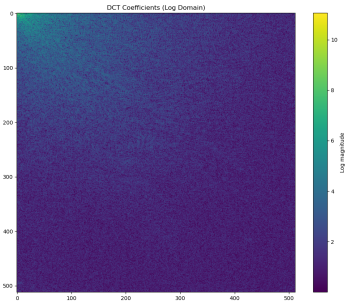
2D-DCT Coefficient (Log domain)



1D-DCT Coefficient (Log domain)

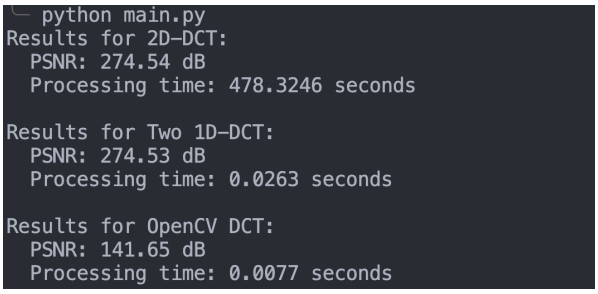


OpenCV Coefficient (Log domain)

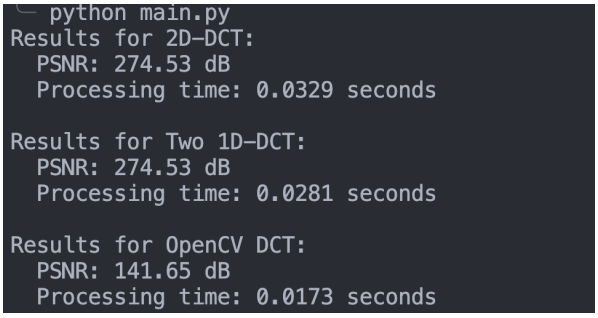


3. Comparison

Runtime and PSNR for origin_2D-DCT



Runtime and PSNR for optimize_2D-DCT



4. Analysis

a. Using matrix multiplication significantly reduces the runtime of computing the DCT in the optimized version compared to the original DCT, as it eliminates the need for iterative calculations in the nested loops

b. Although both the optimized 2D-DCT and two 1D-DCT use matrix multiplication, the 1D-DCT is still faster than the 2D-DCT because the 1D-DCT breaks down the dimensions, which can make matrix computations more efficient.

c. Compared to OpenCV, although the runtime of our implementation is slower, the PSNR results are better. This is because the OpenCV library likely uses faster transformations, but sacrifices PSNR performance as a trade-off for speed.