hw2

2025 年 3 月 27 日

1 HW2

翁正朗 PB22000246 DIP

1.1 1

1. **DFT：**

$$\begin{pmatrix} 8 & -4-4j & 0 & -4+4j \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. **DCT：**

$$\begin{pmatrix} 2 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

3. **Hadamard：**

$$\begin{pmatrix} 8 & 0 & 0 & -8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

4. **Haar（归一化）：**

$$\begin{pmatrix} 2 & 0 & -\sqrt{2} & \sqrt{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

以下为程序实现，可见由于计算精度问题，DCT 存在误差

[14]:
```
import numpy as np
from scipy.fftpack import dct, idct
from scipy.linalg import hadamard

# 输入矩阵
matrix = np.array([
    [0, 1, 1, 0],
    [0, 1, 1, 0],
    [0, 1, 1, 0],
    [0, 1, 1, 0]
])

# DFT
def compute_dft(image):
    M, N = image.shape

    # 构建行频率矩阵和列频率矩阵
    u = np.arange(M)    # 生成一个等差数列, 0 到 M-1
    v = np.arange(N)
    x = np.arange(M)
    y = np.arange(N)

    # 行变换矩阵
    W_x = np.exp(-2j * np.pi * u[:, None] * x / M)
    # 列变换矩阵
    W_y = np.exp(-2j * np.pi * v[:, None] * y / N)

    # 对图像进行离散傅里叶变换
    dft_result = W_x @ image @ W_y.T
    return dft_result

# DCT
def compute_dct(matrix):
    return dct(dct(matrix.T, norm='ortho').T, norm='ortho')
```

```python
def compute_dct_mine(image):
    M, N = image.shape

    # 构建 DCT 变换矩阵
    u = np.arange(M)
    v = np.arange(N)
    x = np.arange(M)
    y = np.arange(N)

    # 计算 DCT 基函数
    W_x = np.sqrt(2/M)*np.cos(np.pi * (2 * x + 1) * u[:,None] / (2 * M))
    W_y = np.sqrt(2/N)*np.cos(np.pi * (2 * y + 1) * v[:, None] / (2 * N))

    # 修正第一个系数的归一化因子
    W_x[0, :] /= np.sqrt(2)
    W_y[0, :] /= np.sqrt(2)

    # 对图像进行离散余弦变换
    dct_result = W_x @ image @ W_y.T
    return dct_result
# Hadamard Transform
def compute_hadamard(matrix):
    # Ensure matrix is of size 2^n x 2^n
    H = hadamard(4)
    print(H)
    print(H@H.T) # H*H^T=nI, n is the size of the matrix

    # H2=hadamard(2)
    # print(np.kron(np.kron(H2,H2),H2))

    # return np.dot(np.dot(H, matrix), H)
    return H @ matrix @ H.T

# Haar Transform
# def compute_haar(matrix):
#     def haar_step(v):
```

```
#         result = np.zeros_like(v)
#         half = v.shape[0] // 2
#         for i in range(half):
#             result[i] = (v[2 * i] + v[2 * i + 1]) / np.sqrt(2)
#             result[half + i] = (v[2 * i] - v[2 * i + 1]) / np.sqrt(2)
#         return result

#     def haar_2d(mat):
#         rows_transformed = np.apply_along_axis(haar_step, 1, mat)
#         return np.apply_along_axis(haar_step, 0, rows_transformed)

#     return haar_2d(matrix)

#compute Haar matrix
def haarMatrix(n, normalized=False):
    # Allow only size n of power 2
    n = 2**np.ceil(np.log2(n))
    if n > 2:
        h = haarMatrix(n / 2)
    else:
        return np.array([[1, 1], [1, -1]])

    # calculate upper haar part
    h_n = np.kron(h, [1, 1])
    # calculate lower haar part
    if normalized:
        h_i = np.sqrt(n/2)*np.kron(np.eye(len(h)), [1, -1])

        # combine parts
        h = 1/np.sqrt(n)*np.vstack((h_n, h_i))

    else:
        h_i = np.kron(np.eye(len(h)), [1, -1])
        # combine parts
        h = np.vstack((h_n, h_i))
```

```python
    return h

def compute_haar(matrix):
    d1=matrix.shape[0]
    d2=matrix.shape[1]

    Hn=haarMatrix(d1,normalized=True)
    return Hn@matrix@Hn.T

# 计算变换
dft_result = compute_dft(matrix)
dct_result = compute_dct(matrix)
hadamard_result = compute_hadamard(matrix)
haar_result = compute_haar(matrix)

# 输出结果
print("DFT Result:\n", np.round(dft_result,4))
print("\nDCT Result:\n", dct_result)
print("\nmy DCT Result:\n", compute_dct_mine(matrix))
print("\nHadamard Result:\n", hadamard_result)
print("\nHaar Result:\n", haar_result)
```

```
[[ 1  1  1  1]
 [ 1 -1  1 -1]
 [ 1  1 -1 -1]
 [ 1 -1 -1  1]]
[[4 0 0 0]
 [0 4 0 0]
 [0 0 4 0]
 [0 0 0 4]]
DFT Result:
 [[ 8.+0.j -4.-4.j  0.+0.j -4.+4.j]
 [-0.-0.j -0.+0.j  0.-0.j  0.+0.j]
 [ 0.-0.j -0.+0.j  0.+0.j  0.+0.j]
 [ 0.-0.j -0.+0.j  0.+0.j  0.+0.j]]


DCT Result:
```

```
 [[ 2.  0. -2.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

my DCT Result:
 [[ 2.00000000e+00  1.11022302e-16 -2.00000000e+00 -4.44089210e-16]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-1.11022302e-16 -6.16297582e-33  1.11022302e-16  2.46519033e-32]
 [-2.77555756e-16 -2.15704154e-32  2.77555756e-16  5.54667824e-32]]

Hadamard Result:
 [[ 8  0  0 -8]
 [ 0  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0  0]]

Haar Result:
 [[ 2.          0.         -1.41421356  1.41421356]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]]
```

1.2 2

均方误差为：

$$\mathrm{MSE} = \sum_{i=33}^{64} 1 = 32$$

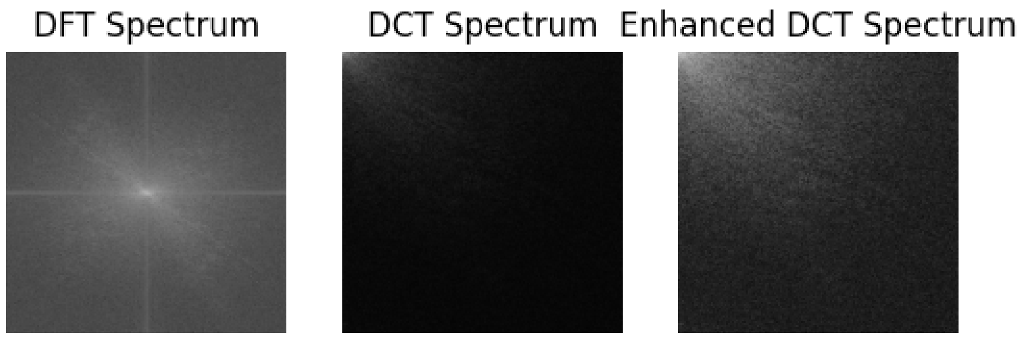
所以，原始图像和重建图像之间的均方误差是 32。

1.3 3

分别适用库函数和手动实现，其中库函数实现中使用了对数变换增强视觉对比度。1. 使用库函数实现

[6]:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取灰度图像
image = cv2.imread('lena.bmp', cv2.IMREAD_GRAYSCALE)

# 检查图像是否成功加载
if image is None:
    print("图像加载失败")
else:
    # 计算离散傅里叶变换 (DFT)
    dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:, :, 0],␣
 ↪dft_shift[:, :, 1]))

    # 显示 DFT 频谱图像
    plt.subplot(131), plt.imshow(magnitude_spectrum, cmap='gray')
    plt.title('DFT Spectrum'), plt.axis('off')

    # 计算离散余弦变换 (DCT)
    dct = cv2.dct(np.float32(image))
    dct_log = np.log(np.abs(dct) + 1)

    # 显示 DCT 频谱图像
    plt.subplot(132), plt.imshow(dct_log, cmap='gray')
    plt.title('DCT Spectrum'), plt.axis('off')

    plt.subplot(133), plt.imshow(np.log(np.abs(dct_log)+1), cmap='gray')
    plt.title('Enhanced DCT Spectrum'), plt.axis('off')

    plt.show()
```

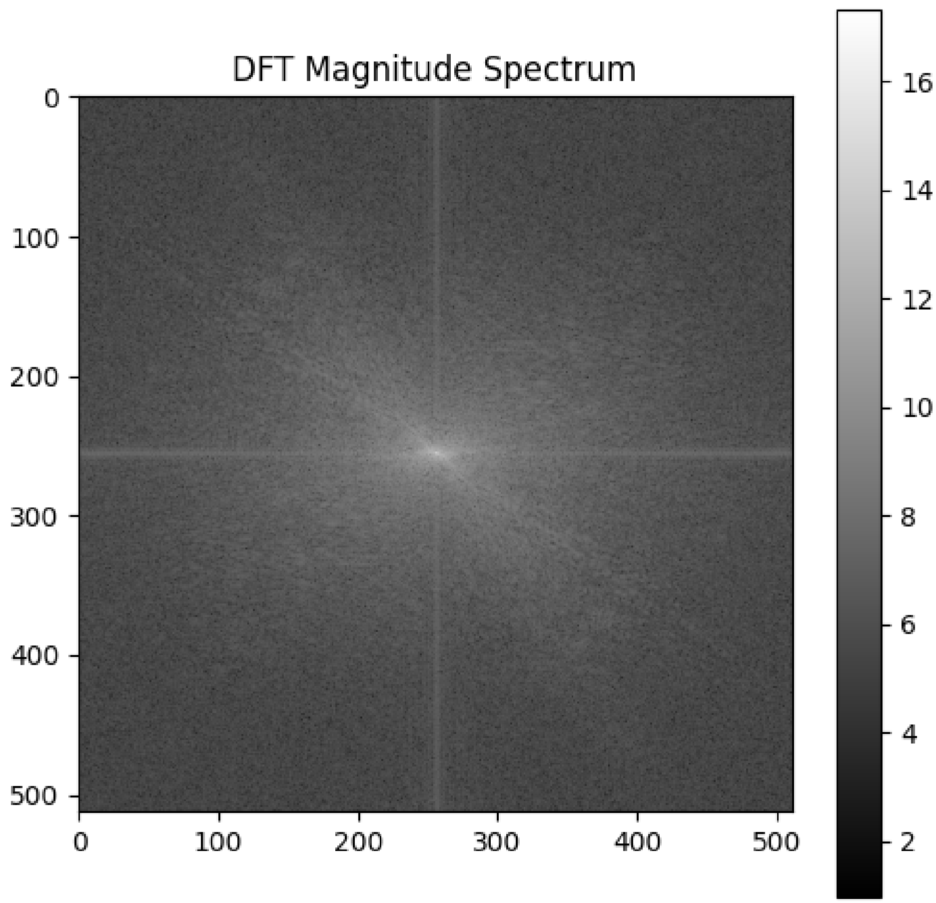


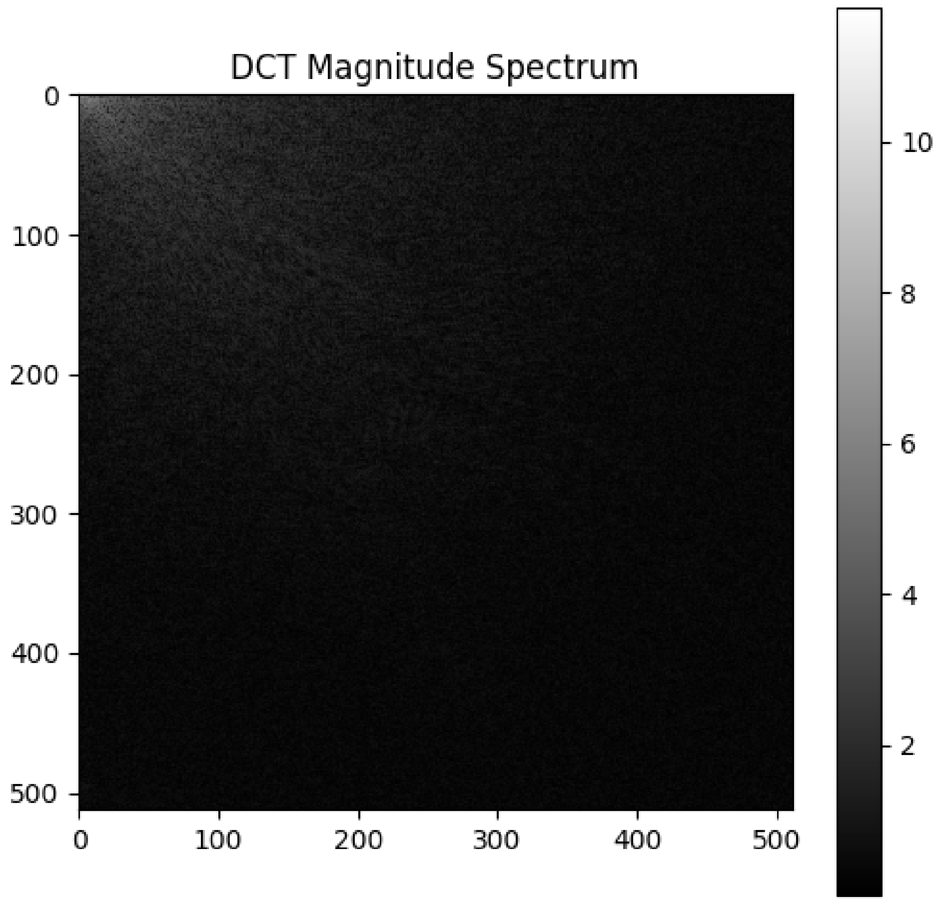
手动实现

```
[3]: import numpy as np
import cv2
import matplotlib.pyplot as plt

# 1. 使用矩阵形式实现离散傅里叶变换 (DFT)
def dft_2d_matrix(image):
    M, N = image.shape

    # 构建行频率矩阵和列频率矩阵
    u = np.arange(M)
    v = np.arange(N)
    x = np.arange(M)
    y = np.arange(N)

    # 行变换矩阵
    W_x = np.exp(-2j * np.pi * u[:, None] * x / M)
    # 列变换矩阵
    W_y = np.exp(-2j * np.pi * v[:, None] * y / N)

    # 对图像进行离散傅里叶变换
    dft_result = W_x @ image @ W_y.T
    return dft_result

# 2. 使用矩阵形式实现离散余弦变换 (DCT)
```

```python
def dct_2d_matrix(image):
    M, N = image.shape

    # 构建 DCT 变换矩阵
    u = np.arange(M)
    v = np.arange(N)
    x = np.arange(M)
    y = np.arange(N)

    # 计算 DCT 基函数
    W_x = np.sqrt(2/M)*np.cos(np.pi * (2 * x + 1) * u[:,None] / (2 * M))
    W_y = np.sqrt(2/N)*np.cos(np.pi * (2 * y + 1) * v[:, None] / (2 * N))

    # 对图像进行离散余弦变换
    dct_result = W_x @ image @ W_y.T
    return dct_result

# 3. 图像预处理函数
def preprocess_image(image_path):
    # 使用 cv2 加载图像并转换为灰度图像
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"无法加载图像 {image_path}")

    # 将图像转换为浮动格式并中心化
    img = np.float32(img)
    return img

# 4. 频谱图像显示函数
def display_spectrum(transformed_image,is_dft=True):
    # 计算幅度谱
    magnitude = np.abs(transformed_image)


    if is_dft == True:
        # 将频谱图像中心化（使用 fftshift）
```

```python
        magnitude_shifted = np.fft.fftshift(magnitude)
    else:
        magnitude_shifted =magnitude

    # 对数尺度增强显示效果
    log_magnitude = np.log(1 + magnitude_shifted)

    # 显示频谱图像
    plt.figure(figsize=(6, 6))
    plt.imshow(log_magnitude, cmap='gray')
    if is_dft == True:
        plt.title('DFT Magnitude Spectrum')
    else:
        plt.title('DCT Magnitude Spectrum')
    plt.colorbar()
    plt.show()

# 5. 主程序：加载图像，进行 DFT 和 DCT 变换，并显示频谱图像
def main():
    image_path = 'lena.bmp'  # 图像路径
    img = preprocess_image(image_path)

    # 1) 计算离散傅里叶变换
    dft_result = dft_2d_matrix(img)
    display_spectrum(dft_result,is_dft=True)

    # 2) 计算离散余弦变换
    dct_result = dct_2d_matrix(img)
    display_spectrum(dct_result,is_dft=False)

if __name__ == '__main__':
    main()
```


DFT Magnitude Spectrum
0
100
200
300
400
500
0
100
200
300
400
500
16
14
12
10
8
6
4
2


DCT Magnitude Spectrum
0
100
200
300
400
500
0
100
200
300
400
500
10
8
6
4
2