TEL AVIV UNIVERSITY

Image Processing Lab

2022-2023

Group #8

Experiment #6

Preliminary Report

Pair No. 43

ID1: 301613501

ID2: 208560086

1. The Discrete Cosine Transform (DCT) was chosen as the transform domain for the JPEG image compression standard because it has several desirable properties for image compression.

   One of the main advantages of the DCT over other transforms, such as the Discrete Fourier Transform (DFT), is that it is more efficient at representing signals that are concentrated in a narrow frequency band. This makes it well-suited for representing the types of signals that are commonly encountered in image data, which tend to have significant energy at low frequencies and rapidly decreasing energy at higher frequencies.

   Another advantage of the DCT is that it is computationally efficient to implement. The DCT can be computed using fast algorithms, such as the Fast DCT algorithm, which has a complexity of $O(n\log(n))$.

   One important difference between the DFT and DCT is how they treat a periodic extension of a signal. The DFT assumes that the signal is periodic, and the transform is computed using this assumption. This can cause problems when the signal has discontinuities at the edges, as the periodic extension will introduce additional, artificial frequencies into the transform.

   The DCT, on the other hand, does not assume that the signal is periodic. Instead, it assumes that the signal is smooth and has limited energy at high frequencies. This makes it more suitable for representing signals with discontinuities at the edges, such as images with sharp boundaries.

   In terms of border processing, the DFT can introduce artifacts at the borders of an image due to the periodic extension of the signal. The DCT, on the other hand, does not have this problem, as it does not assume periodicity. This makes the DCT more suitable for image compression, as it can represent the edges of an image more accurately without introducing artifacts.

2. Prove section.

Claim:

$$X^{(DCT)}[k] = 2Re\left\{e^{-\frac{j\pi k}{2N}}\widetilde{X}_{2N}^{(DFT)}[k]\right\}$$

Proof:

The DFT of $\tilde{x}[n]$ is given by:

$$\tilde{X}_{2N}^{(DFT)}[k] = \sum_{n=0}^{2N-1}\tilde{x}[n]e^{-\frac{j2\pi nk}{2N}} = \sum_{n=0}^{N-1}x[n]\cdot e^{-\frac{j2\pi nk}{2N}} + \sum_{n=N}^{2N-1}0\cdot e^{-\frac{j2\pi nk}{2N}} = \sum_{n=0}^{N-1}x[n]\cdot e^{-\frac{j2\pi nk}{2N}}$$

$$\tilde{X}_{2N}^{(DFT)}[k] = e^{\frac{j\pi k}{2N}}\sum_{n=0}^{N-1}x[n]\cdot e^{-\frac{j2\pi nk}{2N}}e^{-\frac{j\pi k}{2N}} = e^{\frac{j\pi k}{2N}}\sum_{n=0}^{N-1}x[n]\cdot e^{-\frac{j\pi k(2n+1)}{2N}}$$

Now, let us take the real argument of the expiration and multiply by 2:

$$2Re\left\{e^{-\frac{j\pi k}{2N}}\tilde{X}_{2N}^{(DFT)}[k]\right\} = 2Re\left\{e^{-\frac{j\pi k}{2N}}e^{\frac{j\pi k}{2N}}\sum_{n=0}^{N-1}x[n]\cdot e^{-\frac{j\pi k(2n+1)}{2N}}\right\}$$

$$= 2\sum_{n=0}^{N-1}x[n]\cos\left(\frac{\pi k(2n+1)}{2N}\right) = X^{(DCT)}[k]$$

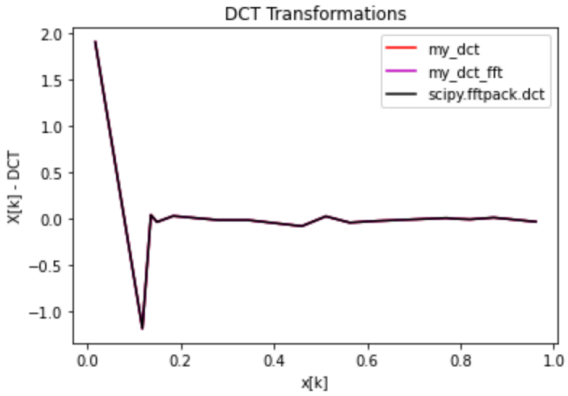
Because the DCT of $x[n]$ is

$$X^{(DCT)}[k] = 2\sum_{n=0}^{N-1}x[n]\cos\left(\frac{\pi k(2n+1)}{2N}\right), \qquad 0 \le k \le N-1$$

3. Implement DCT using both the direct method and FFT.

Result

```
my_dct_err = 4.066e-31
my_dct_fft_err = 1.917e-32
```



Implementation (With NO LOOPs ☺)

```
def my_dct(x):
 N = len(x)
 n,k = np.meshgrid(range(N),range(N))
 arg = k*np.pi*(2*n+1)/(2*N)
 cos_mat = np.cos(arg)
 y = 2*x @ cos_mat.T
 y[0] = y[0]/np.sqrt(4*N)
 y[1:] = y[1:]/np.sqrt(2*N)
 return y

from scipy.fft import fft
def my_dct_fft(x):
 N = len(x)
 x_pad = np.pad(x,(0,N)) # Pad the input signal with N/2 zeros on both sides
 x_fft_pad = fft(x_pad)
 x_fft = x_fft_pad[:N]
 k = np.linspace(0,N-1,N)
 arg = -(1j*np.pi*k)/(2*N)
 y = 2 * np.real(np.exp(arg)*x_fft)
 y[0] = y[0]/np.sqrt(4*N)
 y[1:] = y[1:]/np.sqrt(2*N)
 return y

N = 16
```

```
x = np.sort(np.random.rand(N))

A = my_dct(x)
B = my_dct_fft(x)
C = dct(x, norm =' ortho')

my_dct_err = np.mean(np.square(A - C))
my_dct_fft_err = np.mean(np.square(B - C))
print('my_dct_err = {:.4}'.format(my_dct_err))
print('my_dct_fft_err = {:.4}\n'.format(my_dct_fft_err))

plt.figure()
plt.plot(x, A, 'r', label = 'my_dct')
plt.plot(x, B, 'm', label = 'my_dct_fft')
plt.plot(x, C, 'k', label = 'scipy.fftpack.dct')
plt.legend()
plt.show()
```

4. Implement zigzag ordering pattern of a matrix of arbitrary size.

```
def zigzag(M,N):
 x = np.arange(M*N).reshape(M,N)
 if M%2 == 0:
  order = np.concatenate([np.diagonal(x[::-1,:],i)[::(2*(i%2)-1)][::-1]
                for i in range(1-x.shape[0],x.shape[0]+1)])
 elif M%2 != 0:
  order = np.concatenate([np.diagonal(x[::-1,:],i)[::(2*(i%2)-1)]
                for i in range(1-x.shape[0],x.shape[0]+1)])
 return order
```

We also tried the recommended dimensions as captured below.

```
[291] zigzag(3,4)

      array([ 0,  4,  1,  2,  5,  8,  9,  6,  3,  7, 10, 11])
```

```
[292] zigzag(4,4)

      array([ 0,  4,  1,  2,  5,  8, 12,  9,  6,  3,  7, 10, 13, 14, 11, 15])
```

```
[293] zigzag(8,8)

      array([ 0,  8,  1,  2,  9, 16, 24, 17, 10,  3,  4, 11, 18, 25, 32, 40, 33,
             26, 19, 12,  5,  6, 13, 20, 27, 34, 41, 48, 56, 49, 42, 35, 28, 21,
             14,  7, 15, 22, 29, 36, 43, 50, 57, 58, 51, 44, 37, 30, 23, 31, 38,
             45, 52, 59, 60, 53, 46, 39, 47, 54, 61, 62, 55, 63])
```

```
[294] zigzag(16,16)

      array([  0,  16,   1,   2,  17,  32,  48,  33,  18,   3,   4,  19,  34,
              49,  64,  80,  65,  50,  35,  20,   5,   6,  21,  36,  51,  66,
              81,  96, 112,  97,  82,  67,  52,  37,  22,   7,   8,  23,  38,
              53,  68,  83,  98, 113, 128, 144, 129, 114,  99,  84,  69,  54,
              39,  24,   9,  10,  25,  40,  55,  70,  85, 100, 115, 130, 145,
             160, 176, 161, 146, 131, 116, 101,  86,  71,  56,  41,  26,  11,
              12,  27,  42,  57,  72,  87, 102, 117, 132, 147, 162, 177, 192,
             208, 193, 178, 163, 148, 133, 118, 103,  88,  73,  58,  43,  28,
              13,  14,  29,  44,  59,  74,  89, 104, 119, 134, 149, 164, 179,
             194, 209, 224, 240, 225, 210, 195, 180, 165, 150, 135, 120, 105,
              90,  75,  60,  45,  30,  15,  31,  46,  61,  76,  91, 106, 121,
             136, 151, 166, 181, 196, 211, 226, 241, 242, 227, 212, 197, 182,
             167, 152, 137, 122, 107,  92,  77,  62,  47,  63,  78,  93, 108,
             123, 138, 153, 168, 183, 198, 213, 228, 243, 244, 229, 214, 199,
             184, 169, 154, 139, 124, 109,  94,  79,  95, 110, 125, 140, 155,
             170, 185, 200, 215, 230, 245, 246, 231, 216, 201, 186, 171, 156,
             141, 126, 111, 127, 142, 157, 172, 187, 202, 217, 232, 247, 248,
             233, 218, 203, 188, 173, 158, 143, 159, 174, 189, 204, 219, 234,
             249, 250, 235, 220, 205, 190, 175, 191, 206, 221, 236, 251, 252,
             237, 222, 207, 223, 238, 253, 254, 239, 255])
```