Homework 3

Multimedia Signal Processing

2D-DCT, 2D-IDCT and BTC

Prof. Jing-Ming Guo (郭景明)

Student name : Victor Andrean

ID number : M10507814

National Taiwan University of Science and Technology

Department of Electrical Engineering

Fall 2017

A. Implementation of 2D-DCT and 2D-IDCT

A.1. Implementation 2D-DCT

The Discrete Cosine Transform (DCT) is a process to transforms a signal from a spatial domain representation into a frequency domain representation. In an image, most of the energy will be concentrated in the lower frequencies, so if we transform an image into its frequency components and throw away the higher frequency coefficients, we can reduce the amount of data needed to describe the image without sacrificing too much image quality. However, in this section, I only show how to convert the pixels value of an image from the spatial domain into frequency domain.

$$\text{2-D DCT:}\quad F(i,j)=\frac{2}{N}C(i)C(j)\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}f(x,y)\cos\left[\frac{(2x+1)i\pi}{2N}\right]\cos\left[\frac{(2y+1)j\pi}{2N}\right] \tag{1}$$

$$C_i, C_j=\frac{1}{\sqrt{2}}\ for\ \ i,j=0$$
$$C_i, C_j=1\ otherwise \tag{2}$$

File name: DCT_IDCT.m

```
clear all;clc;
%% Input Image
 A=imread('lena.bmp');
%size of image
[row,col]=size(A);
%convert to double
A=double(A);
%% compression
p=8;     %Block Size
n=p^2; %number of pixels in one block
%2D-DCT process
f=A-128;
for R=1:p:row
   for C=1:p:col
        for i=0:p-1
            for j=0:p-1
                sum=0;
                for x=0:p-1
                    for y=0:p-1

sum=sum+f(x+R,y+C)*cos((2*x+1)*i*pi/(2*p))*cos((2*y+1)*j*pi/(2*p));
                    end
                end

                Ci=1;Cj=1;
                if i==0
                   Ci=1/sqrt(2);
                end
                if j==0
                    Cj=1/sqrt(2);
                end

                F(i+R,j+C)=(2/p)*Ci*Cj*sum;
            end
        end
   end
end
F=round(F);
```

L1 L2 L3 L4 L5 L6

Basically, the way to convert an image from spatial domain to frequency domain is based on equation 1 and 2. Equation 1 is a Discrete Cosine Transform (DCT) Formula. It can be clearly explained using figure 1. The top-left corner is a DC component and the highest frequency occur at the bottom-right corner. "i" and "j" respectively denote the position of the corresponding pixel value in frequency domain matrix.

Considering the pixel value in the spatial domain is in between 0 to 255, in order to perform DCT, we need to shift the pixels in spatial domain become in between -128 to 127. In **L1** 8x8 block is defined. **L2** describe the code to shift the value of the pixel in the spatial domain. Afterward, we perform DCT block-wisely (8x8) based on equation 1. **L3** in a listing program to do addition in equation 1. Equation 2 is represented in **L4** in listing program**. L5** is the final calculation for finding the value of the pixel in the frequency domain for each corresponding row and column. Finally, in **L6,** the value of all pixels in the frequency domain are obtained.

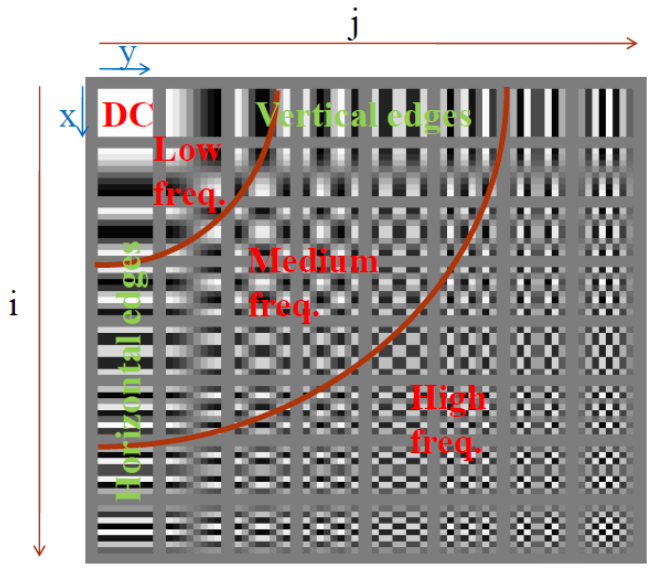

Figure 1. DCT Basis (8x8)

Matrix "F", in figure 2, shows the value of pixels in the frequency domain. We can observe that the high value of the pixels is always concentrated at the top-left corner of the block (8x8). Figure 3 shows the output image in the frequency domain. In addition, the negative value inside matrix F will be ignored. The code is shown in **L11.**

F

512x512 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	259	5	3	0	0	-1	-5	6	246	18	3	5	-8
2	8	-1	1	-5	2	3	-4	3	12	-3	2	5	-3
3	-5	0	-2	2	-1	0	2	-2	-5	2	-2	-3	0
4	2	1	2	1	-1	-1	0	1	-1	-2	1	1	2
5	-1	-1	0	-1	2	1	-1	-1	3	1	0	0	-2
6	1	0	-2	0	-2	1	2	1	0	0	-1	0	1
7	-2	0	3	2	2	-2	-1	-1	-3	0	1	-1	1
8	1	0	-2	-2	-1	2	1	1	3	0	-1	1	-1
9	228	5	-5	0	0	2	0	1	229	5	2	2	6
10	-3	-2	-4	-1	3	0	0	-1	3	3	1	3	0
11	4	-4	2	-1	-3	1	3	-1	0	1	-3	2	1
12	4	-5	2	0	0	-2	0	2	0	0	2	0	4
13	0	1	-2	2	0	0	0	1	3	3	2	-4	0
14	0	1	-4	3	-1	-1	2	-1	1	-2	0	-1	1
15	1	0	-3	2	1	0	0	-1	2	1	1	-3	1
16	2	-2	4	-1	-2	2	-1	1	-1	1	-1	0	-1

Figure 2. The value of pixels in the frequency domain.

Figure 3. Output Image in Frequency Domain (from 2D-DCT)

A.2. Implementation 2D-IDCT

The inverse discrete cosine transform (IDCT) decodes an image into the spatial domain from a representation of the data in frequency domain. The step required to invert the data from the frequency domain to spatial domain is fairly simple. We can just perform equation 3 below.

$$\text{2-D IDCT:} \quad f(x,y) = \frac{2}{N}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1} C(i)C(j)F(u,v)\cos\left[\frac{(2x+1)i\pi}{2N}\right]\cos\left[\frac{(2y+1)j\pi}{2N}\right] \tag{3}$$

L7 shows the program listing to do addition in equation 3. **L8** is the final process to calculate the pixel value in the spatial domain. Is was mentioned previously that in order to do DCT from spatial domain to frequency domain, we need to shift the range from 0 to 255 become -128 to 127. Conversely, after IDCT process, in order to return the value in the spatial domain, the pixel value after the IDCT process need to be shifted by adding 128 for every single pixel value. **L9** shows the process in Matlab. Furthermore, program listing **L10** and **L11**, are used to plot the output images in the frequency domain and spatial domain (the output of 2D-IDCT). The output of 2D-IDCT is shown in figure 4.

File name: DCT_IDCT.m

```
%%%%%%% 2D-IDCT process %%%%%%%%%%%%%%%%%%%
for R=1:p:row
   for C=1:p:col
        for x=0:p-1
            for y=0:p-1

                sum=0;
                for i=0:p-1
                    for j=0:p-1
                    %%%%%%%%%%%%%%%%%%%%%
                    Ci=1;Cj=1;
                    if i==0
                       Ci=1/sqrt(2);
                    end
                    if j==0
                        Cj=1/sqrt(2);
                    end
                    %%%%%%%%%%%%%%%%%%%%%

sum=sum+(Ci*Cj)*F(i+R,j+C)*cos((2*x+1)*i*pi/(2*p))*cos((2*y+1)*j*pi/(2*
p));
                    end
                end

            f(x+R,y+C)=(2/p)*sum;
            end
        end
   end
end
image_SD=round(f+128);

image_FD=uint8(F);
figure('name','frequency domain image','numbertitle','off');
imshow(image_FD);
imwrite(image_FD,'frequency domain image.bmp');

image_SD=uint8(image_SD);
figure('name','Inverse Image','numbertitle','off');
imshow(image_SD);
```

L7

L8

L9

L10

L11

Figure 4. Output Image in Spatial Domain (from 2D-IDCT)

B. Implementation of Block Truncation Coding Images

Block Truncation Coding (BTC) is a type of lossy image compression technique for greyscale images. It divides the original images into blocks and then uses a quantizer to reduce the number of grey levels in each block whilst maintaining the same mean and standard deviation. Equation 4, 5 and 6 are respectively used to calculate mean, mean square, and sigma value. The entire process is well pictured in figure 5.

Like DCT process, BTC is also done blockwisely. After an image is divided into blocks, each block is processed separately. For every block, mean, sigma, low mean, and high mean values are calculated by using respectively equation 4, 6, 8, and 9. The next step is to get a bitmap matrix. The bitmap matrix can be obtained by comparing the original pixels of an image with the mean value for the corresponding block. If the original pixel value is more than the mean, the corresponding cell in the bitmap matrix will be filled by "1". Otherwise, it will value "0". This condition is clearly stated mathematically by equation 7. Using the same way, the rest blocks are processed.

$$\bar{h} = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} h_{i,j} \tag{4}$$

$$\bar{h}^2 = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} {h_{i,j}}^2 \tag{5}$$

$$\bar{\sigma} = \sqrt{\bar{h}^2 - \left(\bar{h}\right)^2} \tag{6}$$

$$b_{i,j} = \begin{cases} 1, if\ h_{i,j} \geq \bar{h} \\ 0, if\ h_{i,j} < \bar{h} \end{cases} \tag{7}$$

Thereafter, the information from bitmap matrix we already obtained can be used to construct the compressed BTC image. The mapping of the new values subject to equation 8, 9, and 10. Based on equation 10, if the corresponding bitmap value is 1, high mean value ("b") will be placed into corresponding pixels in the BTC image. Otherwise, low mean value ("a") will be placed. The complete process is well-depicted in figure 6, 7, 8 respectively for the original pixel values, bitmap pixel values, and the BTC image.

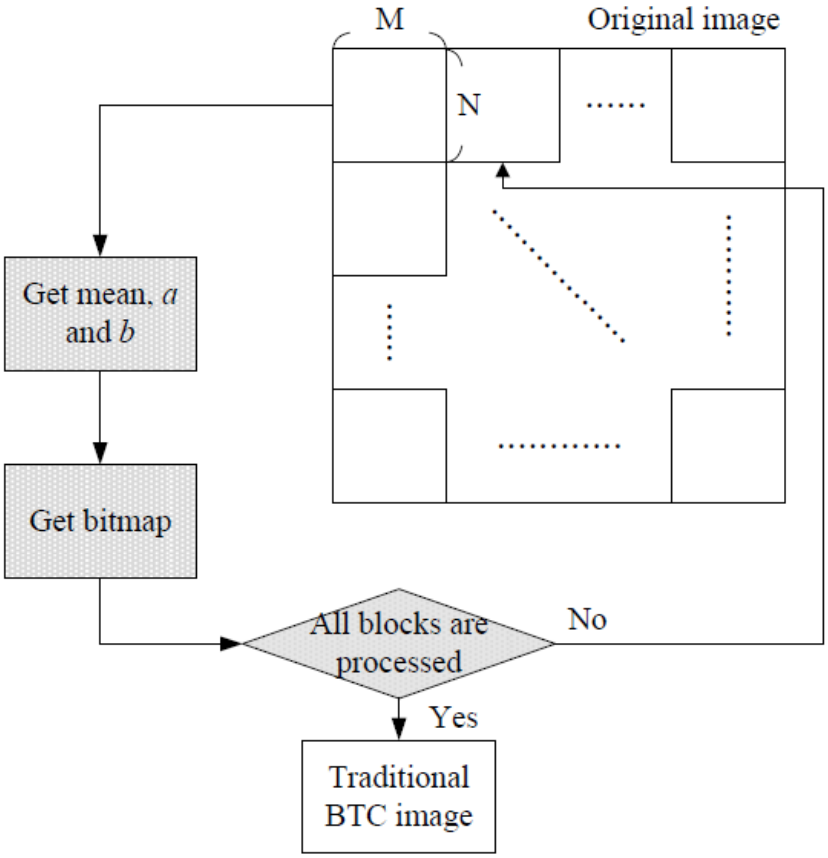

Figure 5. Flowchart BTC-Image Compression.

$$a = \bar{h} - \bar{\sigma}\sqrt{\frac{q}{m-q}} \tag{8}$$

$$b = \bar{h} + \bar{\sigma}\sqrt{\frac{m-q}{q}} \tag{9}$$

$$wh_{i,j} = \begin{cases} b, if\ b_{i,j} = 1 \\ a, if\ b_{i,j} = 0 \end{cases} \tag{10}$$

A

512x512 uint8

	1	2	3	4	5	6	7	8
1	162	162	162	161	162	157	163	161
2	162	162	162	161	162	157	163	161
3	162	162	162	161	162	157	163	161
4	162	162	162	161	162	157	163	161
5	162	162	162	161	162	157	163	161
6	164	164	158	155	161	159	159	160
7	160	160	163	158	160	162	159	156
8	159	159	155	157	158	159	156	157

Figure 6. The original pixels value of the testing image

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	0	1	1
2	1	1	1	1	1	0	1	1
3	1	1	1	1	1	0	1	1
4	1	1	1	1	1	0	1	1
5	1	1	1	1	1	0	1	1
6	1	1	0	0	1	0	0	0
7	0	0	1	0	0	1	0	0
8	0	0	0	0	0	0	0	0

Figure 7. 8x8 bitmap for the first block.

BTCimage

512x512 uint8

	1	2	3	4	5	6	7	8
1	162	162	162	162	162	157	162	162
2	162	162	162	162	162	157	162	162
3	162	162	162	162	162	157	162	162
4	162	162	162	162	162	157	162	162
5	162	162	162	162	162	157	162	162
6	162	162	157	157	162	157	157	157
7	157	157	162	157	157	162	157	157
8	157	157	157	157	157	157	157	157

Figure 8. The matrix of BTC image.

The program listing below shows the implementation of BTC using Matlab. The mean value, the mean square value, and the sigma values can be done by listing **L12**. **L13** is to calculate low mean and high mean values. **L14** is the process to map the low mean and the high mean value into the BTC compressed image. **L15** is used to plot the BTC image. Figure 10 to figure 14 show the output images after BTC process with various kind of "block size". Table 1 shows Compression Ratio value with various kind of "block-size". Bit rate/ bit per pixel can be calculated by equation 11. After knowing bit rate, compression Ratio (CR) can be calculated by equation 12.

$$\boldsymbol{Bit\ rate} = \frac{Low\ mean + High\ mean + Bitmap}{blocksize} \tag{11}$$

$$\boldsymbol{CR} = \frac{8}{\boldsymbol{Bit\ rate}} \tag{12}$$

File name: mybtc.m

```
clear all;clc;
%% Input Image
A=imread('lena.bmp');
%size of image
[row,col]=size(A);
%convert to double
I=double(A);
p=8;     %Block Size
n=p^2; %number of pixels in one block
for R=1:p:row
    for C=1:p:col
        h_avg=0;
        h_ms=0;
        for i=0:p-1
            for j=0:p-1
                h_avg=h_avg+I(i+R,j+C);
                h_ms=h_ms+I(i+R,j+C)^2;
            end
        end
        h_avg=h_avg/(p^2);
        h_ms=h_ms/(p^2);
        sigma=sqrt(h_ms-h_avg^2);

        %get bitmap
        for i=0:p-1
            for j=0:p-1
                if I(i+R,j+C)>h_avg
                  bitmap(i+1,j+1)=1;
                else
                  bitmap(i+1,j+1)=0;
                end
            end
        end
        q=sum(sum(bitmap));
        a=h_avg-sigma*sqrt(q/(n-q));            %low mean
        b=h_avg+sigma*sqrt((n-q)/q);            %high mean

        %mapping high mean and low mean value
        for i=0:p-1
            for j=0:p-1
                if bitmap(i+1,j+1)==1
                  BTCimage(i+R,j+C)=b;
                else
                  BTCimage(i+R,j+C)=a;
                end
            end
        end

    end
end
BTCimage=uint8(BTCimage);
figure('name','Original Image','numbertitle','off');
imshow(A);
figure('name','BTC-Compressed Image','numbertitle','off');
imshow(BTCimage);
imwrite(BTCimage,'BTC-Compressed Image.bmp');
```

L12

L13

L14

L15

Figure 9. The Original Image

Figure 10. BTC-compressed image (Block size=4x4)

Figure 11. BTC-compressed image (Block size=8x8)

Figure 12. BTC-compressed image (Block size=16x16)

Figure 13. BTC-compressed image (Block size=32x32)

Figure 14. BTC-compressed image (Block size=64x64)

As we can see on the table 1, as the Blocksize increase, the Compression Ratio (CR) will increase but the image's quality will reduce. The higher compression ratio, the lower image quality.

Table 1. Compression Ratio value with various kind of "block-size".

Image	p	Blocksize (pxp)	Low mean(bit)	High mean (bit)	Bitmap (pxp)	Bit rate(Bpp)	Compression Ratio (CR)
1	4	16	8	8	16	2	4:1
2	8	64	8	8	64	1.25	6.4:1
3	16	256	8	8	256	1.0625	7.53:1
4	32	1024	8	8	1024	1.015625	7.88:1
5	64	4096	8	8	4096	1.00390625	7.97:1

References:

1. http://imageprocessing-sankarsrin.blogspot.tw/2016/12/block-truncation-coding-image.html
2. https://en.wikipedia.org/wiki/Block_Truncation_Coding
3. http://ieeexplore.ieee.org/document/1094560/
4. https://unix4lyfe.org/dct/
5. http://www.mathematicajournal.com/issue/v4i1/article/8188_Watson.mj.pdf