EE113 Project (2016 Spring)

Youngjae Cho (204669505)

Jaekwan Ahn (604057669)

Part Ⅰ

Step 1.

1.

Lena.jpg

cameraman.tif

2.

ans =

  512 512

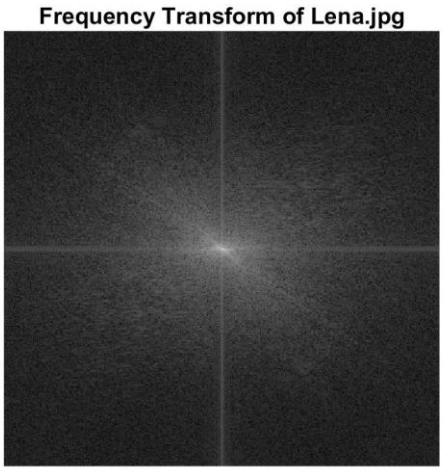
ans =

  256 256

Lena.jpg: 512*512

cameraman.tif: 256*256

3.



Step 2.

1.

Lena convolution with h1

Lena convolution with h2

Cameraman convolution with h1

Cameraman convolution with h2

2.

```
ans =

  305   305

ans =

  256   256
```

The size of the image results from the convolution of the "cameraman.tif" image with the filter h_2 is larger than the size of the "cameraman.tif" image.

C = conv2(A,B) computes the two-dimensional convolution of matrices A and B.

The size of C is determined as follows: if [ma,na] = size(A), [mb,nb] = size(B), and [mc,nc] = size(C), then mc = max([ma+mb-1,ma,mb]) and nc = max([na+nb-1,na,nb]).

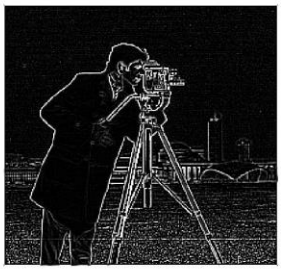
Thus, in this case, the size of the "cameraman.tif" image is (256 x 256), the size of filter h2 is (50 x 50), and consequently, the size of the image results from convolution is (256 + 50 - 1) x (256 + 50 - 1), which is equivalent to (305 x 305).

Part Ⅱ

1.

Lena convolution with h3

Cameraman convolution with h3



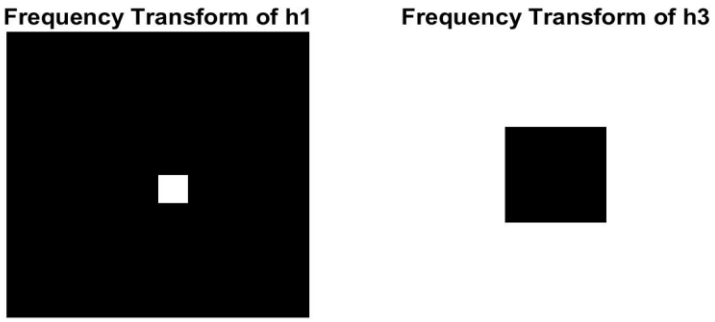
The images above emphasize only the edges and boundaries. On the other hand, the images blur overall in Step 2, Part Ⅰ.

Thus, this filter, h_3, enables to detect edges in image while the previous filters, h_1 and h_2, are used when applying "smoothening" to images that is they create the image blurring effect.

2.

Based on the images above, the filter h_1 shows image blurring, but the filter h_3 shows edge detection. Image blurring is a low pass filter whereas edge detection is a high pass filter. Similarly, low pass frequency components denote image blurring whereas high pass frequency components denote edge detection.

The images below display the frequency transforms of h_1 and h_3 respectively. The frequency transform with the filter h_1 shows the white square in the middle of the image which means a low pass filter, whereas the frequency transform with the filter h_2 shows the black square in the middle of the image which mean a high pass filter.



3.

Equation #1: h3 (convolution) input = distorted

Is equivalent with

Equation #2: FTh3 (multiply) FTinput = FTdistorted

(Fourier Transform: h3 -> FTh3; distorted -> FTdistorted)

We can recover the original “cameraman.tif” image from the distorted image by using Fourier Transform and Inverse Fourier Transform(matlab command: fft2 and ifft2).

Inverse Fourier Transform: FTinput -> input

Which means we can get original image by inverse Fourier transforming the FTinput.

FTinput can be achieved by equation #2: FTinput = FTdistorted / FTh3

However, in this case, FTinput is not exactly what we want because if FTh3 has zeros, then it will result wierd values when we plug in division. Thus, we use offset here, which should be as small as possible. We randomly picked offset value as 10^-16. Now, new equation will be:

Equation #3: FTinput = FTdistorted / (FTh3 + offset)

Notice that matrix dimensions must agree when multiplication. Since FTdistorted is dimension of (258 x 258), we changed dimension of FTh3 by using matlab code of “fft2(h3, 258, 258)”.

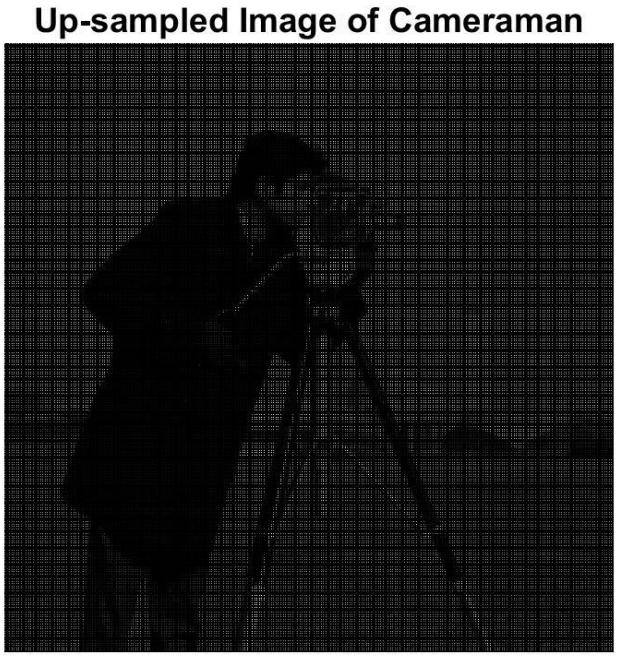
Finally, we can compute FTinput and then use inverse Fourier Transform on it in order to recover original image back.

Part Ⅲ

1.

MATLAB COMMAND:

transpose(upsample(transpose(upsample(IMAGE,L)),L))



Size of the up-sampled image: 768 * 768

2.

Up-sampled Image with Filter Function

The image obtained in (2) by using a function “ZOH filter” is zoomed in with much better quality compared to the image obtained in (1), which means by using a function “ZOH filter” the original image is zoomed in without losing too much data.

3.

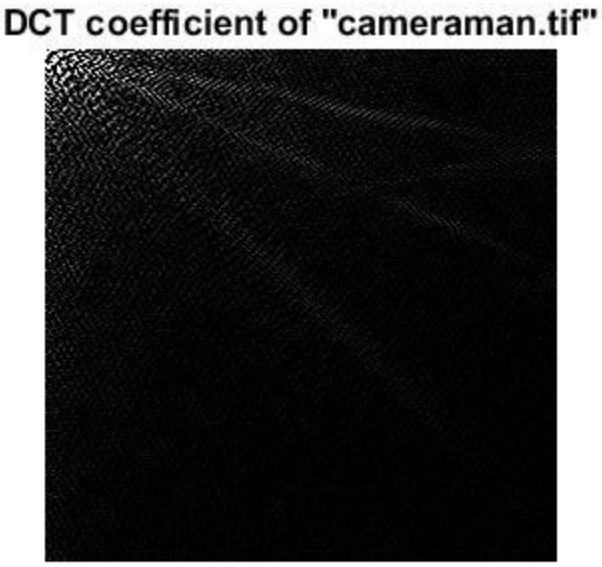
Comparing output images obtained in (1) and (2), the output images obtained in (3) have better quality. They're much clearer and overcome blurring effects from (1) and (2). Furthermore, although it is not easy to distinguish easily, the output image obtained by using bicubic interpolation is brighter and clearer than the output image obtained from using bilinear interpolation.

Resize with Bilinear

Resize with Bicubic

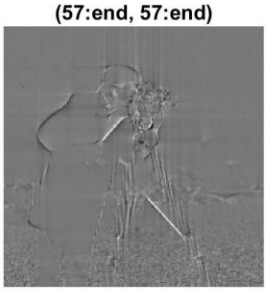
Part Ⅳ

1.



In the image, the white part in the left-upper corner represents low frequency and dark part represents high frequency. The left-upper corner represents background of the original image and some white lines spreading from the left-upper corner represent edges of the original image.

2.




```
 size(Cam_dct1)

ans =

   200   200

size(Cam_dct2)

ans =

   200   200
```

The image obtained from the inverse DCT of A(1:200, 1:200) is closer to “cameraman.tif” than A(57:end, 57:end), because the inverse DCT of A(1:200, 1:200) includes left-upper corner which is low frequency, so that it does not lose too much data, whereas the inverse DCT of A(57:end, 57:end) lose the data of low frequency and includes high frequency more which is black. Therefore, the inverse DCT of A(57:end, 57:end) lose important information which is included in low frequency.

3.

(1:50, 1:50)

(1:150, 1:150)

(1:200, 1:200)

Compression ratio of A(1:50, 1:50) is (256x256)/(50x50) = 26.2144

Compression ratio of A(1:150, 1:150) is (256x256)/(150x150) = 2.9127

Compression ratio of A(1:200, 1:200) is (256x256)/(200x200) = 1.6384

A(1:50, 1:50) has the worst quality; A(1:150, 1:150) has better quality than A(1:50, 1:50); A(1:200, 1:200) has the best quality among these three images.

We can see that as the compression ratio decreases, the quality of image increases. Thus, when we increase the compression ratio, we get the smaller size of the image so that we can save the memory, but we lose a lot of information as well as the quality of the image. Furthermore, by increasing the compression ratio, we lose high frequency components.

Matlab Codes

```
%Part 1: Step 1: #1
close all;
clear all;
clc;

figure, subplot(1,2,1), imshow('Lena.jpg'), title('Lena.jpg')
subplot(1,2,2), imshow('cameraman.tif'), title('cameraman.tif')

%Part 1: Step 1: #2
R_Lena = imread('Lena.jpg');
size(R_Lena)
R_Cameraman = imread('cameraman.tif');
size(R_Cameraman)

%Part 1: Step 1: #3
a = im2double(R_Lena);
b = log(1+abs(fftshift(fft2(a))));
figure, imshow(b, []), title('Frequency Transform of Lena.jpg')

%Part 1: Step 2: #1
A = im2double(R_Cameraman);
h1 = (1/100)*ones(10);
h2 = (1/2500)*ones(50);
Lena_conv_h1 = conv2(a, h1);
figure, subplot(2,2,1), imshow(Lena_conv_h1), title('Lena convolution with h1')
Lena_conv_h2 = conv2(a, h2);
subplot(2,2,2), imshow(Lena_conv_h2), title('Lena convolution with h2')
```

```
Cam_conv_h1 = conv2(A, h1);
subplot(2,2,3), imshow(Cam_conv_h1), title('Cameraman convolution with h1')
Cam_conv_h2 = conv2(A, h2);
subplot(2,2,4), imshow(Cam_conv_h2), title('Cameraman convolution with h2')

%Part 1: Step 2: #2
size(Cam_conv_h2)
size(R_Cameraman)



%Part 2: #1
h3 = [ 1 1 1; 1 -8 1; 1 1 1 ];
Lena_conv_h3 = conv2(a, h3);
figure, subplot(1,2,1), imshow(Lena_conv_h3), title('Lena convolution with h3')
Cam_conv_h3 = conv2(A, h3);
subplot(1,2,2), imshow(Cam_conv_h3), title('Cameraman convolution with h3')

%Part 2: #2
h1_fft2 = fft2(h1);
h1_shift = fftshift(h1_fft2);
h1_freq = abs(h1_shift);
figure, subplot(1,2,1), imshow(h1_freq, []), title('Frequency Transform of h1')
h3_fft2 = fft2(h3);
h3_shift = fftshift(h3_fft2);
h3_freq = abs(h3_shift);
subplot(1,2,2), imshow(h3_freq, []), title('Frequency Transform of h2')

%Part 2: #3
Cam_conv_h3_fft2 = fft2(Cam_conv_h3);
```

```
h3_fft2_fixedDimension = fft2(h3, 258, 258);
offset = 10^-16;
recover_fft2 = Cam_conv_h3_fft2 ./ (h3_fft2_fixedDimension + offset);
recover = ifft2(recover_fft2);
figure, imshow(recover ,[]), title('Recovered image from distorted image')


%Part 3: #1
Upsample_Cam = transpose(upsample(transpose(upsample(R_Cameraman,3)),3));
figure, imshow(Upsample_Cam), title('Up-sampled Image of Cameraman')
size(Upsample_Cam)

%Part 3: #2
filter_function = ZOH_filter(R_Cameraman, 3);
figure, imshow(filter_function, []), title('Up-sampled Image with Filter Function')

%Part 3: #3
Bilinear_Resize_Cam = imresize(R_Cameraman, 3, 'bilinear');
figure, subplot(1,2,1), imshow(Bilinear_Resize_Cam), title('Resize with Bilinear')
Bicubic_Resize_Cam = imresize(R_Cameraman, 3, 'bicubic');
subplot(1,2,2), imshow(Bicubic_Resize_Cam), title('Resize with Bicubic')


%Part 4: #1
Cam_dct = dct2(A);
figure, imshow(Cam_dct), title('DCT coefficient of "cameraman.tif"')

%Part 4: #2
Cam_dct1 = idct2(Cam_dct(1:200, 1:200));
```

```
figure,  subplot(1,2,1), imshow(Cam_dct1, []), title('(1:200, 1:200)')
Cam_dct2 = idct2(Cam_dct(57:end, 57:end));
subplot(1,2,2), imshow(Cam_dct2, []), title('(57:end, 57:end)')
size(Cam_dct1)
size(Cam_dct2)

%Part 4: #3
out1= idct2(Cam_dct(1:50, 1:50));
out2= idct2(Cam_dct(1:150, 1:150));
out3= idct2(Cam_dct(1:200, 1:200));
figure, imshow(out1, []), title('(1:50, 1:50)')
figure, imshow(out2, []), title('(1:150, 1:150)')
figure, imshow(out3, []), title('(1:200, 1:200)')
```