Xi'an Jiaotong-Liverpool University
西交利物浦大学

EEE330 - Image Processing
Assessment 3

Student Name: Wang Xiaoyang
Student ID: 1405888

May 3, 2018

Contents

1 Task 1

This section is to do discrete Fourier (DFT) and cosine transforms (DCT) separately on a 1-D linear signals as shown in Fig 1. DFT and DCT aim to transform signal from spatial or time domain to frequency domain to illustrate the frequency information. For example, Fig.2(a) shows signal spectrum from DFT where we can see the frequency distribution of time signal. Compare with DFT, DCT contains only real parts after transformation so it seems easier to recover without considering effect of magnitude and phase.

One important session is to evaluate inverse transform of both DCT and DFT with part of high frequency components filtered in frequency domain. For signal reconstruction, high frequency components are quite demanding to recover boundary discontinuity. Therefore, filtering those components will have negative effect on reconstruction result. In Fig.3, the inverse DFT performs poorly in line reconstruction even in the first stage with most frequency components remained. The starting and ending points are distorted to a large extent. However, DCT has good performance in approximating boundary discontinuity with few coefficients compared to DFT. The blocking artifacts are less pronounced in DCT as shown in Fig.4. The lines are quite straight even with least frequency components. The comparison in PSRN is shown in Table 1.

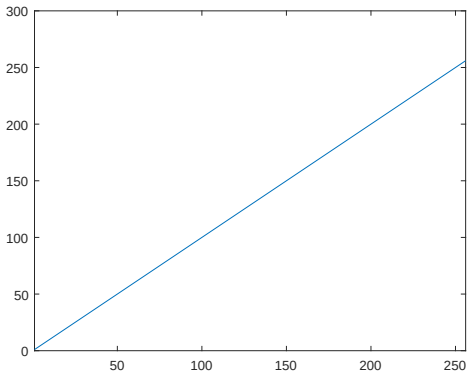
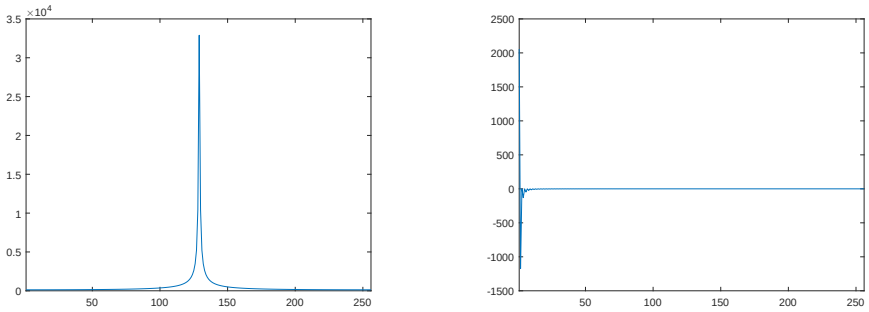


Figure 1: Linear Signal

Table 1: PSNR of Reconstructed Line

$PSNR\ (dB)$	32/256	64/256	96/256	128/256	160/256	192/256	224/256
DFT	-32.404	-32.441	-32.453	-32.455	-32.461	-32.471	-32.514
DCT	57.880	48.320	42.090	36.836	31.584	25.355	15.804



(a) DFT result (b) DCT result

Figure 2: Evaluations of DFT and DCT on signal

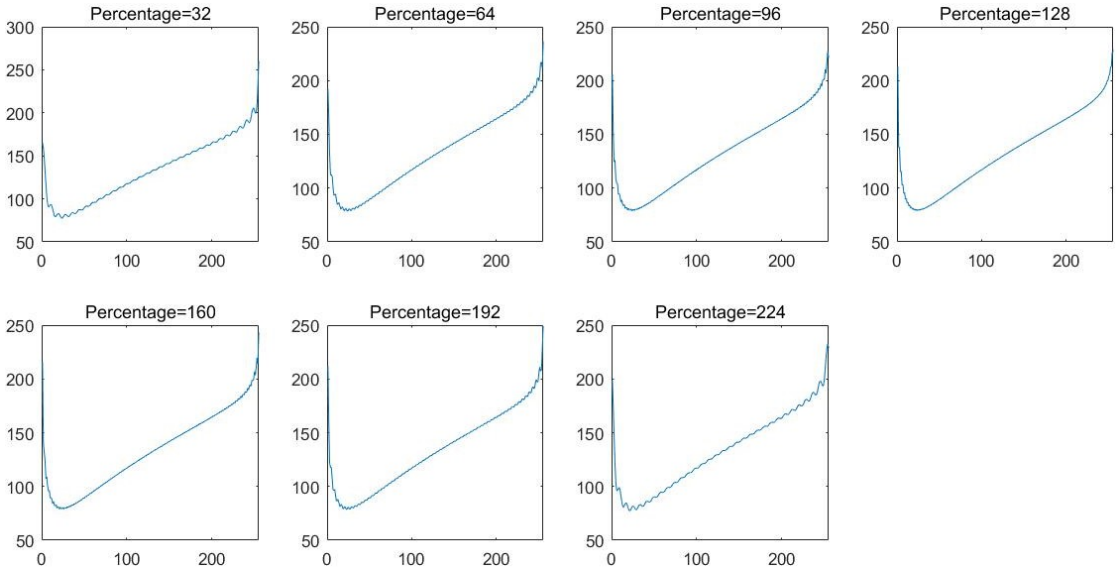


Figure 3: Reconstructed lines from DFT filtered by 7 proportions

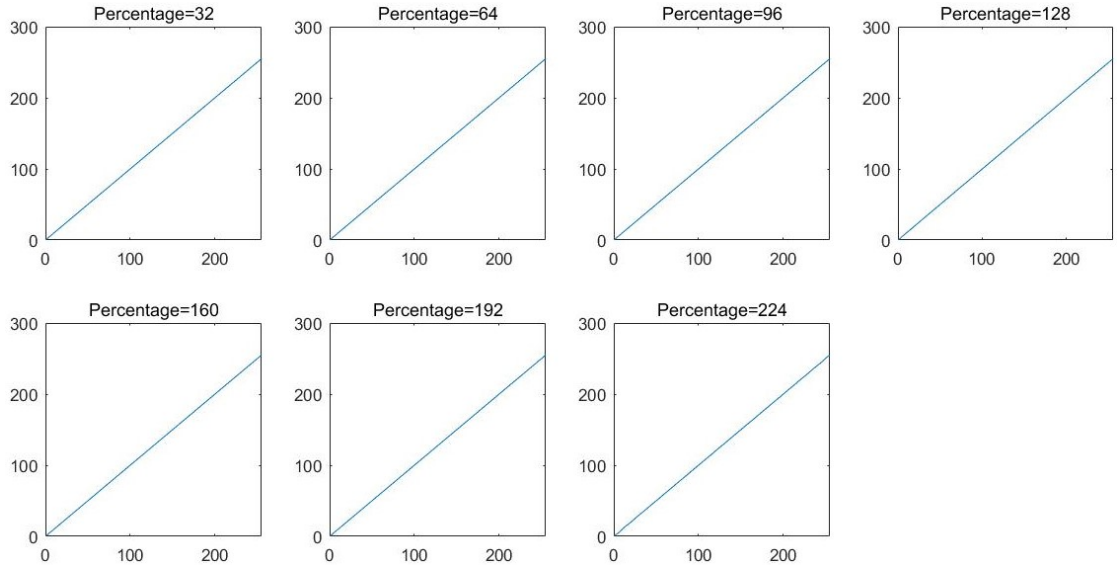


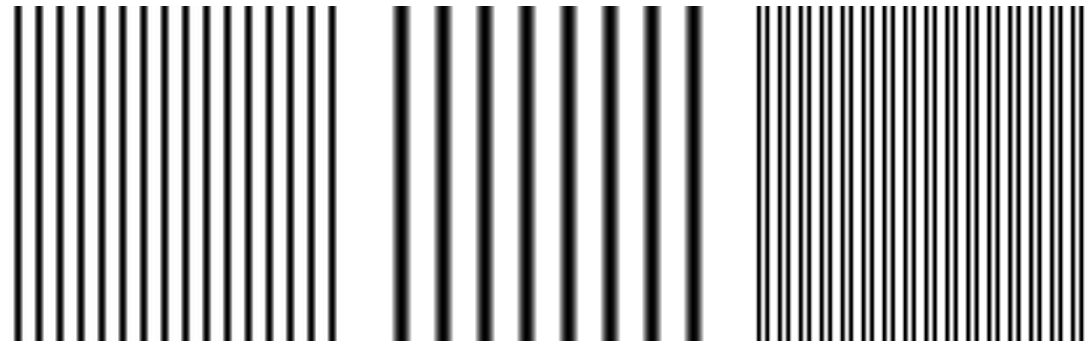
Figure 4: Reconstructed lines from DCT filtered by 7 proportions

2 Task 2

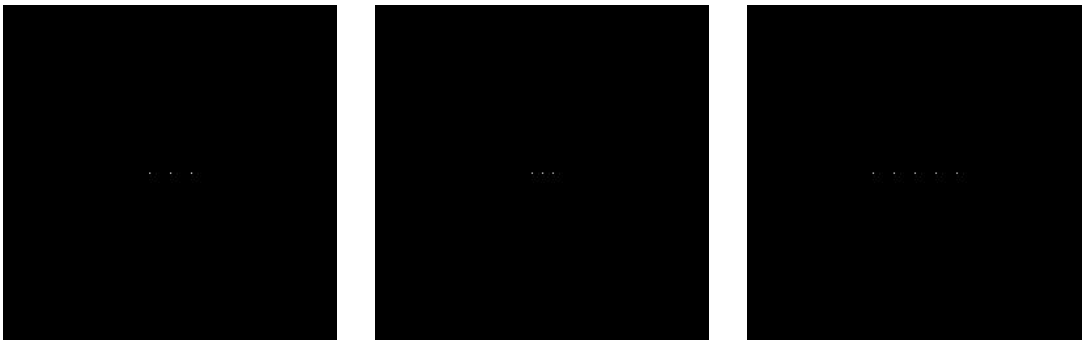
In this session, we start to do 2-D DFT on images. The plotting of original image is shown in listing 1. All plotting results are shown in Fig. 5 with clear label. The first column in Fig. 5 are the plots in spacial and frequency domain when $N = width/16$. The spacial plot is periodically repeated fringes with single frequency of change. Then, in the frequency domain shown in Fig. 5(d), there is only one point to each side of central point referring to 1 frequency component. Note that before doing $fftshift$ the white points in spectrum are located in up corners. After we change the bar width a little wider as shown in (b), which also means a lower frequency, the spectrum in (e) shows a points closer to center. In the last step, we add another frequency components and it leads to a more complex spacial plots pattern containing two changing frequency. Meanwhile, two points appears in each side of center in spectrum.

Listing 1: Plotting of original image

```
width = 256;
N=width/16;
% N = width/8;
n=[1:width]-1;
x = 1+cos(2*pi*n/N);
x = 1+cos(2*pi*n/N)+cos(4*pi*n/N);
% im = ones(width, 1)*x;
figure; imshow(im);
```



(a) Spatial N=width/16 (b) Spatial N=width/8 (c) Spatial extra frequency



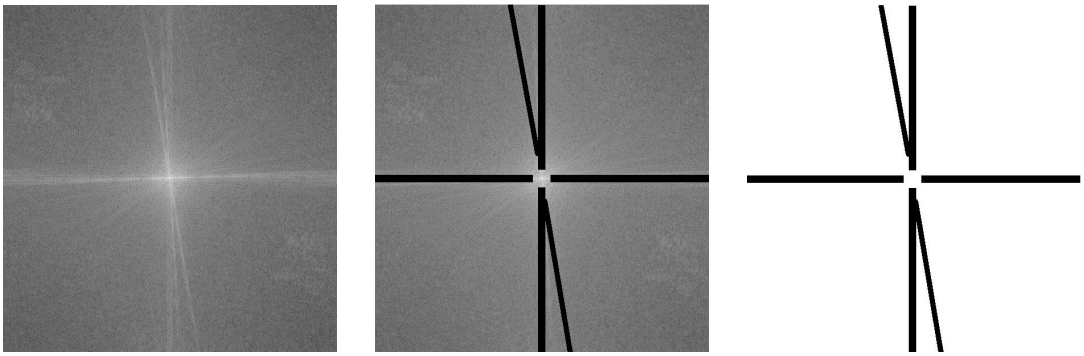
(d) Spectrum N=width/16 (e) Spectrum N=width/8 (f) Spectrum extra frequency

Figure 5: Plots in spatial and frequency domain for Task 2

3 Task 3

In this section, we will use frequency filter to get rid of fences in an image shown in Fig. 7. This is an RGB image with three channels so first step is to separate them. Then do dct for each of channel and get the spectrum shown in Fig. 6(a). Actually the three channels are almost the same so we just pick red channel for illustration. Note that elements in images are orthogonal between spacial and frequency domains. Then the horizon line in spectrum is relevant to vertical fences in original image. It clear to see there are an vertical bright line and another line with some degree to the vertical direction. This is because the horizontal fences are not fully horizontal but with an angle of slope.

The steps to design a filter are shown in Fig. 6(b) and (c). We start with red channel (no difference to choose other channels) spectrum and use it to locate fence information as shown in (b). After that, we transfer that information to build a filter as in (c). The results are illustrated in Fig. 7(b). The fences are blurred to a large extent which means the filter really works.



(a) Red Channel Spectrum (b) Detemine filter shape (c) Mannully obtained filter

Figure 6: Process to determine filter

(a) Original image with fence (b) Filter result with blurred fence

Figure 7: Filter result compared to original image

4 Task 4

This section is to resize image by zero-padding in frequency domain and then transforming back. The implementation details are shown in listing 2 with comments. Fig. 8 illustrates all results by written function and build-in method of *nearest neighbor* and *lanczos3*. It can be observed intuitively that the results are very close to original image while in Table 2 the differences are clear. Zero-padding method performs best for both two images and *lanczos3* is second best and *nearest neighbor* ranks the last place.

Listing 2: Resize by zero-padding

```
function [new_im]=zeroresize(org_im, newsize)
% Args:
%   org_im is the image to be resized
%   newsize is an array containing desired size
imsize = size(org_im);
% Obtain number of zeros to fill in
padnum_dim1 = floor((newsize(1) - imsize(1))/2);
padnum_dim2 = floor((newsize(2) - imsize(2))/2);
% Determine the normalization factor
scale_factor=(newsize(1)*newsize(2))/(imsize(1)*imsize(2));
org_dft = fftshift(fft2(org_im));
% Do zero-padding in frequency domain
new_dft = padarray(org_dft,[padnum_dim1,padnum_dim2]);
new_im = scale_factor*(ifft2(ifftshift(new_dft)));
new_im = uint8(abs(new_im));
end
```

(a) im420 by zeropadding

(b) im420 by NN

(c) im420 by lanczos3

(d) im440 by zeropadding

(e) im440 by NN

(f) im440 by lanczos3

Figure 8: Resize result comparison

Table 2: Comparison on PSNR Results

Type	$im420$	$im440$
zero-padding	32.334	36.811
nearest	28.287	32.752
lanczos3	29.915	34.415

5 Task 5

This section is to investigate principle of 2D-DCT. The first step is to generate bases for 4×4 images. The relation between image and its bases can be expressed as in eq. 1 where $F(k,l)$ is the coefficient of relevant base. Then, I notice that the inverse DCT transform in eq. 2 has the similar pattern and can be rearranged to eq. 3. Note that (m,n) is the spacial location of image pixel and (k,l) is the base index. The $F(k,l)$ is coefficient and relevant base is the part in the square bracket. Then, base expression can be extracted to be eq. 4. To get a certain base is just to keep (k,l) unchanged and traverse all (m,n) values. The Matlab code to generate all the bases is shown in appendix listing 5. The generated 16 bases are shown in Fig. 9(a). The first column is all the frequencies in vertical direction and the first row is those for horizontal. The other components are the superposition of these basic frequencies.

$$\text{image} = \sum F(k,l) \times \text{base}(k,l) \tag{1}$$

$$x(n,m) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \alpha(k)\alpha(l)F(k,l)cos\left[\frac{(2n+1)k\pi}{2N}\right] cos\left[\frac{(2m+1)l\pi}{2N}\right] \tag{2}$$

$$\text{image} = \sum_{k,l} F(k,l) \times \left[\sum_{m,n} \alpha(k)\alpha(l)cos\left[\frac{(2n+1)k\pi}{2N}\right] cos\left[\frac{(2m+1)l\pi}{2N}\right]\right] \tag{3}$$

$$\text{base}(k,l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \alpha(k)\alpha(l)cos\left[\frac{(2n+1)k\pi}{2N}\right] cos\left[\frac{(2m+1)l\pi}{2N}\right] \tag{4}$$

The next step is to project certain image on DCT bases. If we transform an image by DCT, we will get an image of the same size. In the new image, each pixel value no longer represents for color information but the coefficient of relevant base. The general idea to project image is that we use nonzero pixel values in frequency image and return the index of its relevant base. The Matlab code is shown in listing 3.

The final step is to do recovery from frequency domain to spatial domain. The concept has been introduced in eq. 1. The general idea is to utilize the information from function *projection* and sum up all bases multiplied with coefficients. The Matlab code is shown in listing 4. The transform result is shown in Fig. 9(b). The first row of it are original images and the second row are DCT results. The third row are those by inverse transform using function *recover_from_dct* and the images are almost fully recovered despite some very small noises (approx. e^{-15}).

Listing 3: Projection of an image on bases

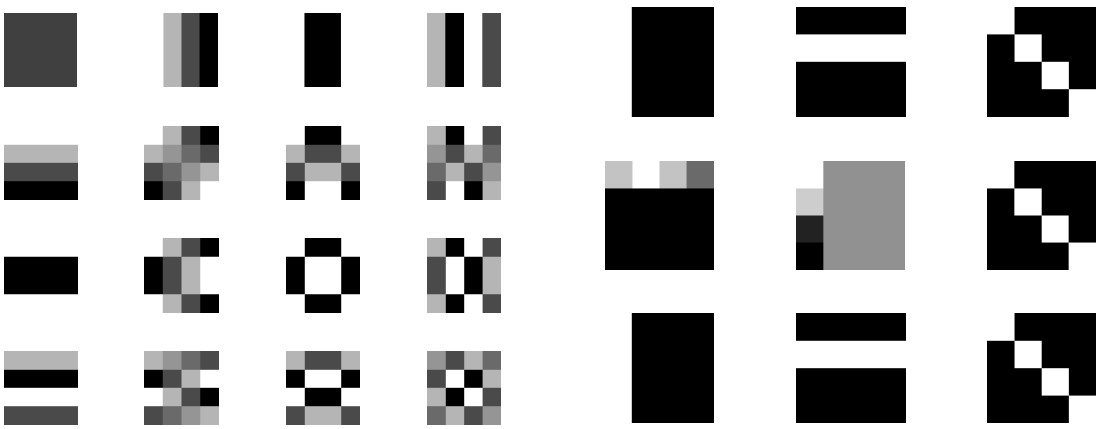
```
function [im_DCT, DCT_bases,indexCnt] = projection(im)
% args:
%   im is the original image
% Return:
%   im_DCT is the transformed image
%   DCT_bases is indices of base with nonzero coeffiecient
%   indexCnt is the count of useful bases

im_DCT = dct2(im);
[M,N] = size(im);
DCT_bases = zeros([M*N,2]);
threshold = 0.001;
indexCnt = 0;
for m=1:1:M
    for n=1:1:N
        if abs(im_DCT(m,n))>threshold
            DCT_bases(indexCnt+1,:)=[m,n];
            indexCnt = indexCnt+1;
        end
    end
end
DCT_bases = DCT_bases(1:indexCnt,:);
end
```

Listing 4: Recover image according to base information

```
function [im_recover] = recover_from_dct(im_dct,dct_bases,
    bases)
% This function is to do inverse dct on image
% args:
%    im_dct is the transformed image
%    dct_bases are the indices of bases with none zero factor
%    bases are the generated bases in a 4-D matrix

size_im = size(im_dct);
im_recover = zeros(size_im);
num_indices = size(dct_bases,1);
for i=1:1:num_indices
    base_factor = im_dct(dct_bases(i,1),dct_bases(i,2));
    current_base = bases(:,:,dct_bases(i,2),dct_bases(i,1));
    im_recover = im_recover + base_factor.*current_base;
end
end
```



(a) Generated 16 bases for DCT

(b) DCT and inverse DCT results

Figure 9: Results for Task 5

Appendices

Listing 5: base_generation

```
function [bases] = base_generate(MtxSize,plot)
% This functio is used to generate DCT basis of size MtxSize
% Only SUQARE images are surpported.
% Args:
%   MtxSize: function will return MtxSize*MtxSize basis
%   (n,m) are the spacial coordination
%   (k,l) are base coordination
%   if plot=1, basis will be ploted

if nargin<2
    plot = 1;
end

N = MtxSize;
baseCnt = 1;
bases = zeros([N,N,N,N]);
bases_norm = zeros([N,N,N,N]);
for k=0:1:N-1
    for l=0:1:N-1
        for m=0:1:N-1
            for n=0:1:N-1
                if(k==0)
                    a_k = sqrt(1/N);
                end
                if(l==0)
                    a_l = sqrt(1/N);
                else
                    a_k = sqrt(2/N);
                    a_l = sqrt(2/N);
                end
                bases(m+1,n+1,k+1,l+1) = a_k*a_l*...
                cos((2*n+1)*k*pi/(2*N))*...
                cos((2*m+1)*l*pi/(2*N));
            end
        end
        bases_norm(:,:,k+1,l+1) = mat2gray(bases(:,:,k+1,l
           +1));
    end
end
```

```
39
40 if plot==1
41     figure;
42     for k = 1:1:N
43         for l = 1:1:N
44             subplot(N,N,baseCnt);
45             imshow(bases_norm(:,:,l,k));
46             baseCnt = baseCnt + 1;
47         end
48     end
49 end
50 end
```

Listing 6: Task 1

```
1  %%
2  x = [1:256];
3  plot(x);
4  xlim([1,256]);
5  x_dft = fft(x);
6  x_dct = dct(x);
7  x_dft_tmp = x_dft;
8  x_dct_tmp = x_dct;
9  dft_shift = fftshift(x_dft);
10 figure;
11 plot(abs(dft_shift));
12 xlim([1,256]);
13 pctg = [32,64,96,128,160,192,224];
14 psnr_dft = zeros([1,7]);
15 psnr_dct = zeros([1,7]);
16 %%
17 figure;
18 for i=1:1:7
19     index = 256-(pctg(i)-1);
20     % DFT
21     x_dft = x_dft_tmp;
22     x_dft(index:256) = 0;
23     x_idft = ifft(x_dft);
24     subplot(2,4,i);
25     plot(abs(x_idft));
26     xlim([0,255]);
27     title(['Percentage=' num2str(pctg(i))]);
28     psnr_dft(i) = psnr(abs(x_idft),x);
29 end
```

```
%%
figure;
for i=1:1:7
    index = 256-(pctg(i)-1);
    % DCT
    x_dct = x_dct_tmp;
    x_dct(index:256)=0;
    x_idct = idct(x_dct);
    subplot(2,4,i)
    plot(abs(x_idct));
    xlim([0,255])
    title(['Percentage=' num2str(pctg(i))]);
    psnr_dct(i) = psnr(x_idct,x);
end
```

Listing 7: Task 2

```
%%
width = 256;
N=width/16;
n=[1:width]-1;
%x = 1+cos(2*pi*n/N);
x = 1+cos(2*pi*n/N)+cos(4*pi*n/N);
im = ones(width, 1)*x;
figure; imshow(im);
%%
im_dft = fft2(im);
figure; imshow(log(abs(im_dft)));
im_dft_shift = fftshift(im_dft);
im_dft_final = log(abs(im_dft_shift));
figure; imshow(im_dft_final);
```

Listing 8: Task 3

```
%%
im = imread('fence.jpg');
% Separate channels
redIm = im(:,:,1);
greenIm = im(:,:,2);
blueIm = im(:,:,3);
% Do dft for each channel
red_dft = fftshift(fft2(redIm));
green_dft = fftshift(fft2(greenIm));
blue_dft = fftshift(fft2(blueIm));
```

```
% Filter images
task3_filter;
red_filt = red_dft.*test_filter;
green_filt = green_dft.*test_filter;
blue_filt = blue_dft.*test_filter;
% Inverse transform on each channel
red_re = uint8(abs(ifft2(ifftshift(red_filt))));
green_re = uint8(abs(ifft2(ifftshift(green_filt))));
blue_re  = uint8(abs(ifft2(ifftshift(blue_filt))));
% Stack 3 channels back to recover
im_re = cat(3, red_re, green_re, blue_re);
imshow(im_re);
```

Listing 9: Task 3 Filter_generation

```
%%
filter=ones([611,588]);
filter(299:311,:) = 0;
filter(:,288:300) = 0;
filter(290:320,279:309)=1;
% imshow(filter);
%%
test_filter = imread('test_filter.bmp');
test_filter = test_filter(:,:,1);
for i=1:1:611
    for j=1:1:588
        if test_filter(i,j)~=0
            test_filter(i,j) = 1;
        end
    end
end
test_filter = double(test_filter);
```

Listing 10: Task 4

```
%%
im_440 = imread('lenna_ds440.bmp');
im_420 = imread('lenna_ds420.bmp');
im_ref = imread('lenna512.bmp');
%%
im_420_zero = zeroresize(im_420,[512 512]);
im_440_zero = zeroresize(im_440,[512 512]);
%%
im_420_nearest = imresize(im_420,[512 512],'nearest');
```

```
im_440_nearest = imresize(im_440,[512 512],'nearest');
%%
im_420_lanczos3 = imresize(im_420,[512 512],'lanczos3');
im_440_lanczos3 = imresize(im_440,[512 512],'lanczos3');
%%
psnr_420_zero = psnr(im_420_zero,im_ref);
psnr_440_zero = psnr(im_440_zero,im_ref);
psnr_420_neareast = psnr(im_420_nearest,im_ref);
psnr_440_neareast = psnr(im_440_nearest,im_ref);
psnr_420_lanczos3 = psnr(im_420_lanczos3,im_ref);
psnr_440_lanczos3 = psnr(im_440_lanczos3,im_ref);
```

Listing 11: Task 5

```
%% Initialize images to be tested
im1 = [1,0,0,0;1,0,0,0;1,0,0,0;1,0,0,0];
im2 = [0,0,0,0;1,1,1,1;0,0,0,0;0,0,0,0];
im3 = [1,0,0,0;0,1,0,0;0,0,1,0;0,0,0,1];
%% Base generation and prjection
[bases] = base_generate(4,1);
[im1_dct, dct_bases1,indexCnt1] =
   projection_an_image_on_its_DCT_bases(im1);
[im2_dct, dct_bases2,indexCnt2] =
   projection_an_image_on_its_DCT_bases(im2);
[im3_dct, dct_bases3,indexCnt3] =
   projection_an_image_on_its_DCT_bases(im3);
%% Image recovery from frequency domain
[im1_recover] = recover_from_dct(im1_dct,dct_bases1,bases);
[im2_recover] = recover_from_dct(im2_dct,dct_bases2,bases);
[im3_recover] = recover_from_dct(im3_dct,dct_bases3,bases);
%% Image Refine (Set very small number to 0)
im1_re = im_refine(im1_recover);
im2_re = im_refine(im2_recover);
im3_re = im_refine(im3_recover);
```