Report of Coursework Assignment

Jingran Fu

# Overview

This report will cover the solutions of question 2.1. There are six sub-questions. The first three are related to Discrete Cosine Transform(DCT), where DCT is used to process the original pictures. After this procedure, a threshold is set to filter the matrix so that some high frequency values can be replaced with “0”. To compress the picture the filtered matrix will be coded by Run Length Encoding(RLE).

Question 4,5 are related with Gaussian and Laplacian Pyramid, which can also be used to compress the pictures. Similarly, after a filtered and coded by RLC, the processed pictures will be compressed. Question 6 will compare the performance of these two compression methods through the rate-distortion curves. Besides, some critical analysis will be given at this part.

Among all m files, the Question12, Question3 and Question45 are the main function of this course work. At the beginning of these three files, a workpath is given, which should be modified before running. Besides some detail explanations are given in the code.

# Method2.1.1

## Question1 & Question2

**Explanation of the method**

In Question1, DCT is used to process the original pictures. Discrete Cosine Transform(DCT) can be describe as below:





In this equation, N is the size of the picture (or a N x N Blocks) that need to be transformed. From this equation we can tell that a DCT will put low frequency information at top-left area of the new matrix while the high frequency information will be gathered in bottom right corner, which will be helpful for smoothing or filtering. In Matlab, this equation can be transferred as a matrix form:





Obviously, the backward DCT will be:



**DCT Solution:**

function D=dctforward(I)

% estimate the size of input picture "I"

sz=size(I);

m=sz(1);

DM=zeros(m,m);

%build dct matrix

for i = 0:m-1

for j = 0:m-1

if i == 0

DM(i+1,j+1) = ((1/m)^0.5)\*cos(((2\*j+1)\*i\*pi)/(2\*m));

else

DM(i+1,j+1) = ((2/m)^0.5)\*cos(((2\*j+1)\*i\*pi)/(2\*m));

end

end

end

%process dct transform.

D=DM\*I(:,:,1)\*DM';

end

function D=dctbackward(I)

%estimate the size of input picture.

sz=size(I);

m=sz(1);

DM=zeros(m,m);

for i = 0:m-1

for j = 0:m-1

if i == 0

DM(i+1,j+1) = ((1/m)^0.5)\*cos(((2\*j+1)\*i\*pi)/(2\*m));

else

DM(i+1,j+1) = ((2/m)^0.5)\*cos(((2\*j+1)\*i\*pi)/(2\*m));

end

end

end

%reconstruct the picture

D=DM'\*I(:,:,1)\*DM;

end

“question12.m” gives a solution of taking a 8x8 block from input picture and process the DCT forward and backward process. Before this file is run, the path should be modified as described in the code. And the value of N can be saved in “size”. Through adding or modifying the value in “size” this method can work for blocks of arbitrary size. The result of DCT with 8x8, 16x16 and 64x64 blocks are given at the end of this section.

**The main function of Question1&2**

addpath('C:\Users\Admin\Desktop\DA\Image and Video Analysisi\coursework\data\');

% the path should be modified to real path.

image=imread('barbara.bmp');

image=im2double(image);

% give a list of the size of block.

size=[8,16,64];

len=length(size);

for i=1:length(size)

% take nxn blocks from input picture.

n=size(i);

imageblock=image(1:n,1:n);

%dct transform

imagedct=dctforward(imageblock);

%dct reconstructed

imagedctback=dctbackward(imagedct);

subplot(2,len,i);

imshow(imagedct);

title('dctforward');

subplot(2,len,len+i);

imshow(imagedctback);

title('dctbackward');

end

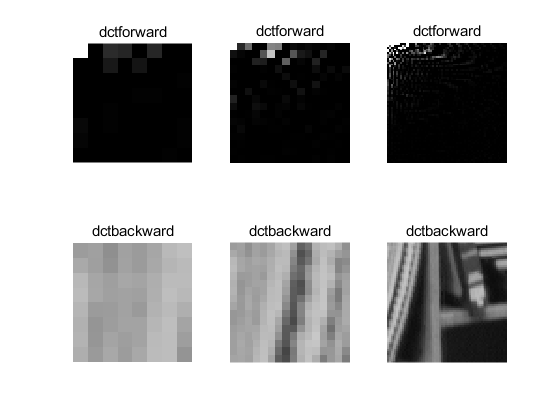


Figure 1 the result of DCT with 8x8 16x16 64x64 blocks

## Question 3

**Explanation of the method**

Firstly, the input pictures need to be DCT transformed. To reduce the amount of calculation, pictures will be divided into several NxN blocks before DCT. In this part “blkproc(mage,[n,n],fun1)” is applied, which can dived the original pictures and each blocks will be processed by “fun1”, and finally returns a matrix with all blocks. Since human eyes are not very sensitive to high frequency information, through setting a threshold to filter these kind of information the input picture can be compressed at a higher rage without reducing much quality.

The filtered data will be coded in RLE to complete the whole process of compression. RLE is a simple form of lossless data compression, which will consist of two lists, one of them is used to record the value in the matrix and each of them is 8bits, and the other one record the run length of each value. The Whole length of these two lists represent the number of compressed data. Hence, the compressed number of bits equals to the length x 8bits.

**The solution of setting thresholding**

%I means the input image, s means the value of threshold

function D=comthresholding(I,s)

%estimate the size of input image

sz=size(I);

newI=zeros(sz(1),sz(2));

%if the abs value of input image is no more than s, it will be set as "0"

for i=1:sz(1)

for j=1:sz(2)

if I(i,j)>=s || I(i,j)<=0-s

newI(i,j)=I(i,j);

else

newI(i,j)=0;

end

end

end

%return the new matrix

D=newI;

end

**The solution of DLE**

function D=Dle2(Image)

%estimate the size of input image

length=size(Image);

% store all data in a 1 demension list

len=length(1)\*length(2);

Im=zeros(1,len);

in=1;

for m=1:length(1)

for n=1:length(2)

Im(in)=Image(m,n);

in=in+1;

end

end

j=1;

run(1)=1;

for i=1:(len-1)

if Im(i)~=Im(i+1)

data(j)=Im(i);

j=j+1;

run(j)=1;

else

run(j)=run(j)+1;

end

end

data(j)=Im(len);

%stroe run length in list "run", store data in list "data".

D{1}=run;

D{2}=data;

end

**The method of Evaluation**

There are two main indicators to evaluate the compression result, Signal to Noise Rate(SNR,dBs) and compression rate(bites per pixel). The equation of SNR is shown below:

In this equation f is the original picture and g is the compressed picture. SNR gives the distortion rate after compressions. The compression rate is:

In this part a loop is used to set different threshold from 0.15 to 1 with a step of 0.05 to calculate different SNR and CR and plot a curve to describe the relationship of these two indicators.

**The main function of question 3**

addpath('C:\Users\Admin\Desktop\DA\Image and Video Analysisi\coursework\data\');

inputimage=imread('lena.bmp');

% judge if the picture is gray, if not, convert input picture to gray

% version.

if ndims(inputimage)==3

inputimage=rgb2gray(inputimage);

else

inputimage=inputimage;

end

im=im2double(inputimage);

%------------------------------------------------------------------------------

b=1;

[m,n]=size(im);

%set different thresholding values

for i=0.15:0.05:1

%DCT transform

imdct=blkproc(im,[8,8],'dctforward');

%filtering with thresholds

imcom=comthresholding(imdct,i);

%reconstruction

imre=blkproc(imcom,[8,8],'dctbackward');

% calculate the SNR

snr(b)=ImageSnr(im,imre);

%DLE

imcode=Dle2(imcom);

codelen=length(imcode{1})+length(imcode{2});

%calculate the compression rate

cr1(b)=(codelen\*8)/(m\*n);

b=b+1 ;

end

figure; plot(cr1,snr);

title('Rate-Distortion Curve, lena')

xlabel('CR(bpps)')

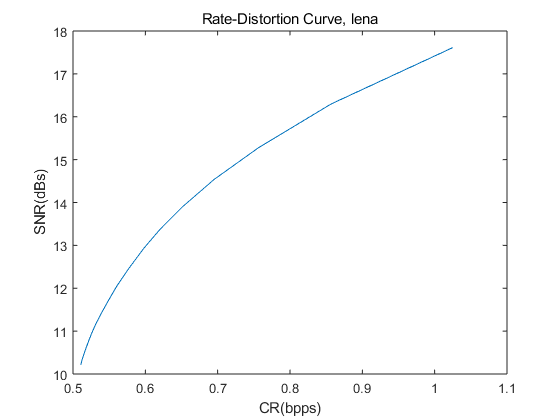
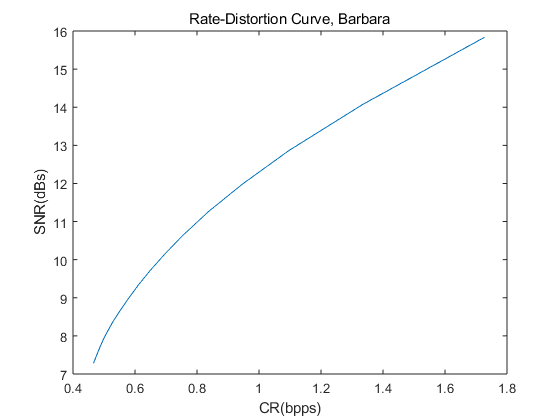
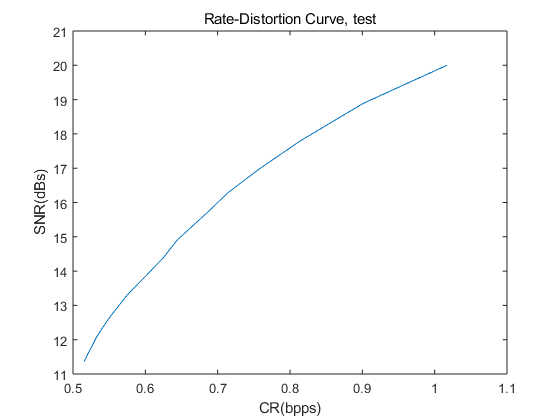
 ylabel('SNR(dBs)')

Figure 2 Rate-Distortion Curves

## Question4 & Question5

**explanation of the method**

For a Gaussian Pyramid, we chose a 2D Gaussian function convolution mask and apply it to the image. The 2D Gaussian kernel can be described as below:





In this quotation (u, v) are coordinates of Gaussian coefficients, σ is the variance. This method applies a 5x5 Gaussian kernel and σ=1. After sub-sampling with a factor of 2, the result will become the ﬁrst level of the pyramid. The next level can be created by ﬁltering this image with the same convolution kernel and subsampling again to generate the 2nd level, and so on. To super-sample, the higher level need to convolution the same kernel to expand with a factor of 2.

The Laplacian Pyramid is easy to come from Gaussian Pyramid. The method is to expand the next Gaussian level and take it away from the previous level the result is the Laplacian Pyramid layer.



Therefore, through reverse the process of building Laplacian Pyramid we can reconstructed the original image.

**Gaussian pyramid**

%I is the input image, G is the Gauss kernel

function R=Gaussianp(I,G)

sz=size(I);

N=log2(sz(1));

Gpyramid{1}=I;

r=I;

for index=2:N

[m,n]=size(r);

%apply Gausse smmothing

smoothimage=conv2(r,G);

smoothimage=smoothimage(3:m+2,3:n+2);

%subsamping

newsz=m/2;

r=zeros(newsz,newsz);

for i=1:newsz

for j=1:newsz

r(i,j)=smoothimage(2\*i-1,2\*j-1);

end

end

Gpyramid{index}=r;

end

R=Gpyramid;

end

**Laplacian pyramid**

function lappy=LaplacianP(gpy,G)

% gpy=R

len=length(gpy);

for i=len-1:-1:1

%ExpandG is a expand function

exp=ExpandG(gpy{i+1},G);

ladata{i}=gpy{i}-exp;

end

lappy=ladata;

lappy{len}=gpy(len);

end

**Reconstruction Function**

% this function is used to reconstruct the original image

function [laplacere]=laplacere(Gauss,Laplace,G)

len=length(Gauss);

imre=Gauss{len};

for i=len:-1:2

imre=ExpandG(imre,G);

imre=Laplace{i-1}+imre;

end

% add Laplace cth a expanded Gaussian pyramid layer

laplacere=imre;

end

Question5 is similarly with question3, the same filter and encoding method(RLE) are utilized in this part to compress the input picture. The result of Question4&5 are given at the end of solution.

**The main function of Question4&5**

addpath('C:\Users\Admin\Desktop\DA\Image and Video Analysisi\coursework\data\');

inputimage=imread('Lena.bmp');

%if the input image is not gray version transfer it into gray version.

if ndims(inputimage)==3

inputimage=rgb2gray(inputimage);

else

inputimage=inputimage;

end

im=im2double(inputimage);

%calculate the value of Gauss kernel, the radius is 2 and the sigma is 1.2.

G=Gauss21(2,1.2);

gsspy=Gaussianp(im,G);

lplpy=LaplacianP(gsspy,G);

lpre=laplacere(gsspy,lplpy,G);

%print the Gaussian and Laplace Pyramid.

for i=1:3

figure;imshow(lplpy{i});

title('Laplacian Pyramid');

figure;imshow(gsspy{i});

title('Gaussian Pyramid');

end

figure;imshow(lpre);

title('reconstruction');

%calculate the value of SNR and compression rate under different threshold.

a=1;

for threshold=0.05:0.025:1

[cr(a),comladata]=lacomrate(im,lplpy,threshold);

lapre=laplacere(gsspy,comladata,G);

snr(a)=ImageSnr(im,lapre);

a=a+1;

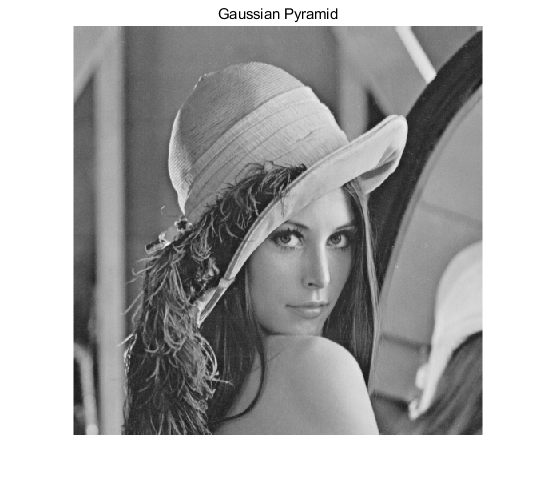
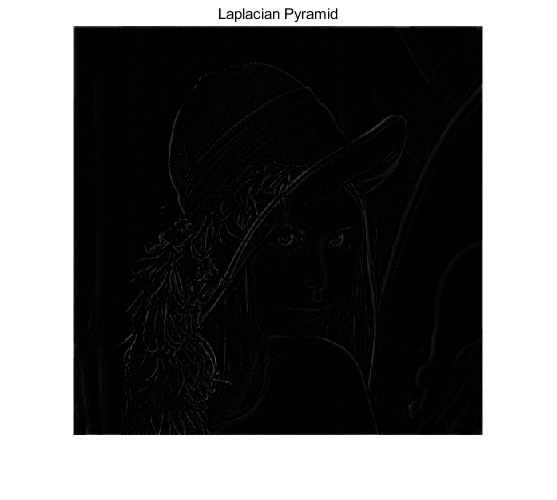
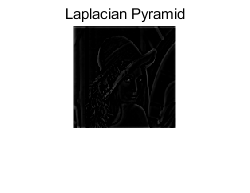
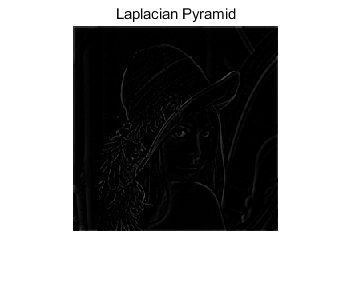
end

figure; plot(cr,snr)

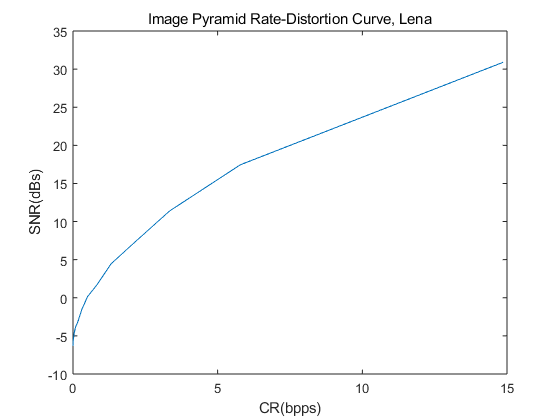
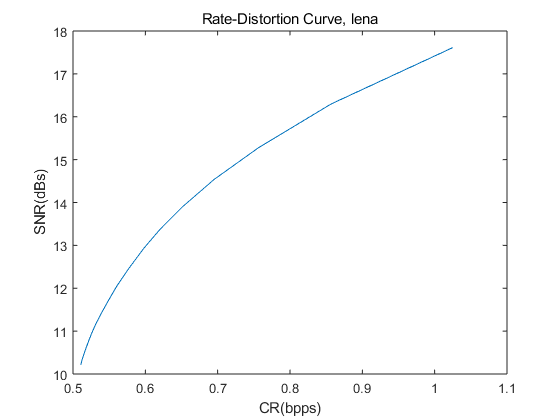
title('Rate-Distortion Curve, Lena')

xlabel('CR(bpps)')

ylabel('SNR(dBs)')c



Question6



These two rate-distortion curves show that the value of SNR will rise with the growth of CR, Which can be explained. When the value of CR increase, every pixel has more bits which means a lower threshold is applied and more information is maintained. Obviously, the value of SNR will be larger.

Compared with Image pyramid, DCT can give a better performance in compression when using the same threshold and encoding method. These two plots illustrate that when have the same CR, DCT has a larger SNR, which means under the same compression rate, DCT method can provide more information. The reason is that after DCT transform, the low frequency information can gather around top right corner of each block. After filtered, DLE can work more efficient.

# Conclusion

This coursework helps me get a better understanding of DCT, LOG, DOG, and other relevant knowledge. While, with the limitation of time, I just give a basic solution of these questions, the performance of my method still can be improved. For instance, when I use DLE, my solution encodes the data in a row and column order, in fact, if using the “Z" path the DLE will be more efficient in DCT method.

The number of words : 1021