实验4-有损图像压缩系统设计

1 实验目的

（1）熟悉传统图像有损压缩的基本原理和开发流程；

（2）了解和掌握变换编码方法，如DCT变换等对信号熵的影响；

（3）了解量化的原理，以及量化等级与信号失真程度之间的关系；

（4）掌握常见的图像压缩性能评价指标，如PSNR和SSIM。

2. 实验环境

（1）硬件环境：PC；

（2）软件环境：Windows 10、MATLAB R2017a

3 实验内容

3.1 离散余弦变换

离散余弦变换（DCT）能够以数据无关的方式去除输入信息之间的相关性，在图像压缩标准中得到了广泛应用。在图像压缩中，通常是对固定的$M \times N$的图像块f做二维DCT变换，以获得$M \times N$大小的DCT变换系数矩阵F。对应的公式定义如下:

$$F(u,v)=\frac{2C(u)C(v)}{\sqrt{MN}}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}B(u,v)f(i,j), \tag{1}$$

其中$i,u=0,\ldots,M-1$，$j,v=0,\ldots,N-1$。常数$C(u)$和$C(v)$由下式得出:

$$C(\xi)=\begin{cases}\frac{\sqrt{2}}{2}, & \xi=0\\ 1, & \xi\neq 0.\end{cases} \tag{2}$$

对应的二维DCT变换的基函数$B(u,v)$定义为:

$$B(u,v)=\cos\frac{(2i+1)\cdot u\pi}{2M}\cdot\cos\frac{(2j+1)\cdot v\pi}{2N}. \tag{3}$$

需要注意的是，基函数$B(u,v)$是一个尺寸为$M \times N$的矩阵。

下面给出了matlab语言实现的`plotdct2base`代码，该代码块可完成指定基函数$B(u,v)$的定义和可视化。

```matlab
%%file plotdct2base.m
function [dct2base,dct2baseplot]=plotdct2base(u,v,M,N,flag)
dct2base=zeros(M,N);
%base function for index (u,v)
for i=0:M-1
    for j=0:N-1
```

```
        dct2base(i+1,j+1)=cos((2*i+1)*u*pi/M/2)*cos((2*j+1)*v*pi/N/2);

    end
end

%将MxN的二维基函数扩充为（wsizeM）x（wsizeN）大小 for better illustration
%可视化上的操作
if flag
    wsize=32;
    wsizearray=ones(wsize);
    dct2baseplot=kron(dct2base,wsizearray);
    %imshow(dct2baseplot)
else
    dct2baseplot = 0;
end
```

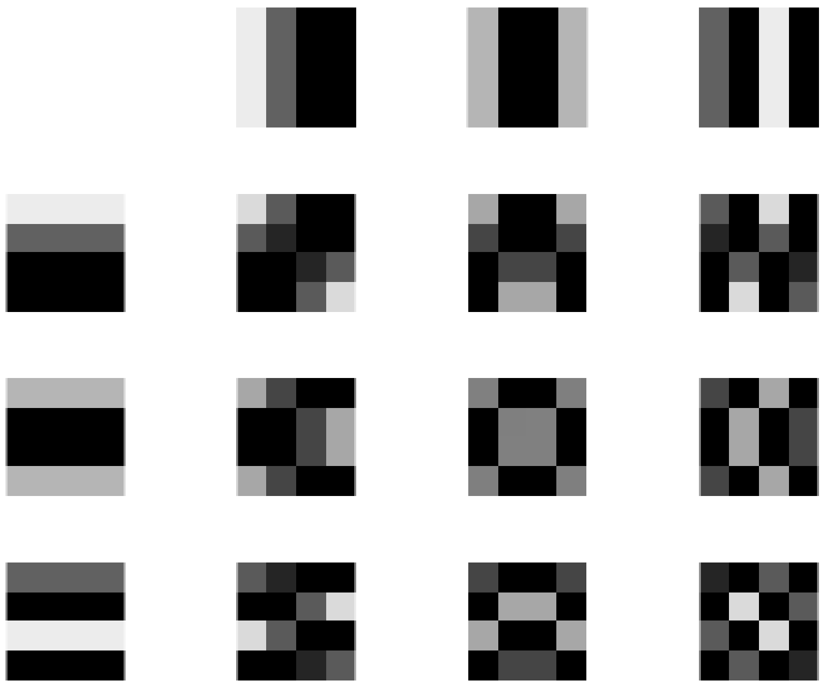
下面给出了matlab语言实现的`plotidct2base`代码，该代码块可完成指定逆变换中基函数$B(u,v)$的定义和可视化。

```
%%file plotidct2base.m
function [idct2base,idct2baseplot]=plotidct2base(i,j,M,N,flag)
idct2base=zeros(M,N);
C = ones(max(M,N),1);
C(1) = sqrt(2)/2;
%base function for index (u,v)
for u=0:M-1
    for v=0:N-1
        scale = 2*C(u+1)*C(v+1)/sqrt(M*N);

 idct2base(u+1,v+1)=cos((2*i+1)*u*pi/M/2)*cos((2*j+1)*v*pi/N/2)*scale;

    end
end
```

假设$M = N = 4$,我们可通过执行下述的示例代码`Lec5Exp1`，获得相应的基函数的值，以及可视化的效果。

```
%% Lec4Exp1: Visualize the 2D-DCT base function
%%function plotdct2baseall
clear all, close all
M=4;
N=4;
flag=1;
for u=0:M-1
    for v=0:N-1
        [dct2base,dct2baseplot]=plotdct2base(u,v,M,N,flag);
        subplot(M,N,u*N+v+1),imshow(dct2baseplot)
    end
end
```

以下为代码执行后的输出结果：



给定$M = N = 8$,对8×8大小的图像块，每次2D DCT/IDCT变换，需要做64次迭代计算。执行下述的示例代码`Lec5Exp2`，不难看出DCT变换是可逆变换。

```matlab
%% Lec4Exp2: 2D DCT/IDCT Transform
clear all, close all
data = [202 205 189 188 189 175 175 175;
        200 203 198 188 189 182 178 175;
        203 200 200 195 200 187 185 175;
        200 200 200 200 197 187 187 187;
        200 205 200 200 195 188 187 175;
        200 200 200 200 200 190 187 175;
        205 200 199 200 191 187 187 175;
        210 200 200 200 188 185 187 186]

M=8;
N=8;
C = ones(max(M,N),1);
C(1) = sqrt(2)/2;
flag=0;
dct_2D_coef=zeros(M,N);
coef=zeros(M,N);

s = 0;
%2D DCT Transform
for u=0:M-1
     for v=0:N-1
         dct2base=plotdct2base(u,v,M,N,flag);
         scale = 2*C(u+1)*C(v+1)/sqrt(M*N);
         tic
         dct_2D_coef(u+1,v+1) = sum(sum(data.*dct2base))*scale;
         s = s+toc;
     end
end

%2D Inverse DCT Transform
for i=0:M-1
     for j=0:N-1
```

```
        idct2base=plotidct2base(i,j,M,N,flag);
        tic
        data_rec_2D(i+1,j+1) = sum(sum(dct_2D_coef.*idct2base));
        s = s+toc;
    end
end
disp('The reconstructed version of original data for 2D DCT/IDCT Transform:')
disp(floor(data_rec_2D+0.5))
disp(sprintf('Runtime is %0.6f s for %d x %d 2D-DCT/IDCT Transform', s, M, N))

```

以下为代码执行后的输出结果：

```
data =

   202   205   189   188   189   175   175   175
   200   203   198   188   189   182   178   175
   203   200   200   195   200   187   185   175
   200   200   200   200   197   187   187   187
   200   205   200   200   195   188   187   175
   200   200   200   200   200   190   187   175
   205   200   199   200   191   187   187   175
   210   200   200   200   188   185   187   186

The reconstructed version of original data for 2D DCT/IDCT Transform:
   202   205   189   188   189   175   175   175
   200   203   198   188   189   182   178   175
   203   200   200   195   200   187   185   175
   200   200   200   200   197   187   187   187
   200   205   200   200   195   188   187   175
   200   200   200   200   200   190   187   175
   205   200   199   200   191   187   187   175
   210   200   200   200   188   185   187   186

Runtime is 0.004568 s for 8 x 8 2D-DCT/IDCT Transform
```

如果利用前面获得二维DCT基函数对图像块进行DCT变换，需要大量重复的计算。由于二维基函数中关于u和v的两个分量相互独立，故二维DCT变换可以转化为两个一维DCT变换。该变换过程可表示为矩阵乘法：

$$F = TfT', \tag{4}$$

其对应的逆变换矩阵实现为

$$f = T'FT. \tag{5}$$

T为$N \times N$大小的DCT矩阵，其定义为：

$$T[i,j] = \begin{cases} \frac{1}{\sqrt{N}}, & i = 0 \\ \sqrt{\frac{2}{N}} \cdot \frac{\cos(2j+1)\cdot i\pi}{2N}, & i > 0. \end{cases} \tag{6}$$

以$M = N = 8$为例，执行案例`Lec5Exp2`，对固定的矩阵`data`进行DCT变换和反变换。不难发现，DCT变换是可逆无损的。且随着迭代次数的减少，运算速度得到了显著提升。

```
%% Lec4Exp3: Example of DCT Matrix
T = zeros(N,N);
T(1,:) = 1/sqrt(N);
for i = 1 : N-1
    for j = 0 : N-1
        T(i+1, j+1) = sqrt(2/N) * cos((2*j+1)*i*pi/(2*N));
    end
end

tic
dct_coef = (T * data * T');
data_rec_matrix = floor(T' * dct_coef * T+0.5);
s = toc;
disp('The reconstructed version of original data for 2D DCT Matrix:')
disp(floor(data_rec_matrix+0.5))
disp(sprintf('Runtime is %0.6f s for %d x %d 2D-DCT/IDCT Matrix Operation', 
s, N, N))
```

以下为代码执行后的输出结果：

```
The reconstructed version of original data for 2D DCT Matrix:
   202   205   189   188   189   175   175   175
   200   203   198   188   189   182   178   175
   203   200   200   195   200   187   185   175
   200   200   200   200   197   187   187   187
   200   205   200   200   195   188   187   175
   200   200   200   200   200   190   187   175
   205   200   199   200   191   187   187   175
   210   200   200   200   188   185   187   186

Runtime is 0.000431 s for 8 x 8 2D-DCT/IDCT Matrix Operation
```

3.2 DCT变换系数的信息熵

3.2.1 中宽量化器的影响

对DCT变换系数直接进行量化（中宽），可以重建信号的误差为代价，显著的降低信源的信息熵，达到图像信号压缩的目的。下面的matlab函数`dct_entropy_demo`提供了简单的中宽量化的演示功能。可以通过改变2D-DCT变换核的尺寸和中宽量化器的量化步长，在图像信号的信息熵和重建质量两个指标之间取得平衡。

```
%%file dct_entropy_demo.m
function dct_entropy_demo(filename, blksize, q)
% close all; clear all
% filename = 'data\Lenna.png';
% blksize = [8 8];
% q= 3;
disp(sprintf('The block size of DCT is %d', blksize(1)))
disp(sprintf('The step size of midtread quantizer is %d', q))
img = imread(filename);
img_yuv = double(rgb2ycbcr(img));
img_yuv_offset = img_yuv-128;
img_yuv_rec = zeros(size(img_yuv));
img_dct2 = @(block_struct) dct2 (block_struct.data);
```

```
img_idct2 = @(block_struct) idct2 (block_struct.data);

for chl= 1:3
    x = blockproc(img_yuv_offset(:,:,chl),blksize,img_dct2);

    x_h = floor(x/q+0.5); % 中宽量化器
    [Height,Width] = size(x_h);

    [p_h,Binx_h] = hist(x_h(:),max(x_h(:))-min(x_h(:))+1);% compute the histogram.
    p_h = p_h/(Height*Width);% normalize the histogram counts.

    H_h = sum(-p_h.* log2(p_h+1e-08));% compute the entropy % to avoid log(0), add a small positive value to p.
    disp(sprintf('Entropy of input image for channel %d = %6.2f',chl,H_h))
    %subplot(2,3,chl),plot(Binx_h,p_h,'k')
    %xlabel('Pixel value'), ylabel('relative count');
    %subplot(2,3,chl+3),imshow(x_h),title(sprintf('channel = %d', chl))

    y_h = x_h * q;
    img_yuv_rec(:,:,chl) = blockproc(y_h, blksize, img_idct2) +128;
end

img_yuv_rec = uint8(img_yuv_rec);
img_rec = ycbcr2rgb(img_yuv_rec);

%mse_yuv= sum((double(img_yuv(:))-double(img_yuv_rec(:))).^2)/numel(img_yuv);
mse_rgb= sum((double(img(:))-double(img_rec(:))).^2)/numel(img);
psnr_rgb = 10*log10(255^2/mse_rgb);
ssim_rgb = ssim(img, img_rec);

figure
subplot(121),imshow(img),title('original iamge')
subplot(122),imshow(img_rec), title('reconstructed image')
disp(sprintf('SSIM: %6.4f; MSE: %6.2f, PSNR: %6.2f dB', ssim_rgb, mse_rgb, psnr_rgb))
```

将2D-DCT的变换核尺寸设为4×4,同时中宽量化器的步长设为50。通过下面的代码块`Lex5Exp3`调用`dct_entropy_demo`，通过代码执行中的打印信息不难看出，即使对DCT系数直接做比较粗的量化，重建后的画质仍然可以让人接受。同时，Y、Cb和Cr几个通道的信息熵下降明显。

```
%% Lec4Exp4: Influence of kernel size
close all; clear all
filename = 'data\Lenna.png';
blksize = [4 4];
q= 50;
dct_entropy_demo(filename, blksize, q)
```

以下为代码执行后的输出结果：

```
The block size of DCT is 4
The step size of midtread quantizer is 50
Entropy of input image for channel 1 =   0.66
Entropy of input image for channel 2 =   0.38
Entropy of input image for channel 3 =   0.46
SSIM: 0.9621; MSE: 104.19, PSNR:  27.95 dB
```

将2D-DCT的变换核尺寸设为8×8,同时中宽量化器的步长保持不变。可以发现变换核的尺寸增大可以限制降低测试图像的信息熵，同时重建后的画质亦显著提高。

```
blksize = [8 8];
q= 50;
dct_entropy_demo(filename, blksize, q)
```

以下为代码执行后的输出结果：

```
The block size of DCT is 8
The step size of midtread quantizer is 50
Entropy of input image for channel 1 =   0.42
Entropy of input image for channel 2 =   0.18
Entropy of input image for channel 3 =   0.19
SSIM: 0.9720; MSE:  73.80, PSNR:  29.45 dB
```

继续将2D-DCT的变换核尺寸增大为16×16,同时中宽量化器的步长保持不变。可以发现变换核的尺寸增大仍然可以限制降低测试图像的信息熵，同时重建后的画质亦显著提高，但增幅相对变缓。

```
blksize = [16 16];
q= 50;
dct_entropy_demo(filename, blksize, q)
```

以下为代码执行后的输出结果：

```
The block size of DCT is 16
The step size of midtread quantizer is 50
Entropy of input image for channel 1 =   0.35
Entropy of input image for channel 2 =   0.10
Entropy of input image for channel 3 =   0.11
SSIM: 0.9737; MSE:  67.56, PSNR:  29.83 dB
```

3.2.2 JPEG压缩标准中的量化矩阵

JPEG压缩标准根据心理学研究结果，为亮度和色度图分别提供了不同的量化矩阵。不同的频率对应不同的量化矩阵系数值。下面的matlab函数`dct_jpeg_entropy_demo`给出了相应的JPEG量化矩阵的实现。可以通过改变量化步长，在图像信号的信息熵和重建质量两个指标之间取得平衡。

```matlab
%%file dct_jpeg_entropy_demo.m
function dct_jpeg_entropy_demo(filename, q)
% filename = 'data\Lenna.png';
% q= 3;

blksize = [8 8];
img = imread(filename);
img_yuv = double(rgb2ycbcr(img));
img_yuv_offset = img_yuv-128;
img_yuv_rec = zeros(size(img_yuv));
img_dct2 = @(block_struct) dct2 (block_struct.data);
img_idct2 = @(block_struct) idct2 (block_struct.data);

jpgQstepsY = [16 11 10 16 24 40 51 61;
              12 12 14 19 26 58 60 55;
              14 13 16 24 40 57 69 56;
              14 17 22 29 51 87 80 62;
              18 22 37 56 68 109 103 77;
              24 35 55 64 81 104 113 92;
              49 64 78 87 103 121 120 101;
              72 92 95 98 112 100 103 99];

jpgQstepsC = [17 18 24 47 66 99 99 99;
              18 21 26 66 99 99 99 99;
              24 26 56 99 99 99 99 99;
              47 66 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99];
qthread = @(block_struct) block_struct.data./(jpgQstepsY*q);
iqthread = @(block_struct) block_struct.data.*(jpgQstepsY*q);

qthreadC = @(block_struct) block_struct.data./(jpgQstepsC*q);
iqthreadC = @(block_struct) block_struct.data.*(jpgQstepsC*q);
for chl = 1:3
    x = blockproc(img_yuv_offset(:,:,chl),blksize,img_dct2);
    if chl>1
        x_h = floor(blockproc(x,blksize,qthreadC)+0.5);
    else
        x_h = floor(blockproc(x,blksize,qthread)+0.5);
    end
    %x_h = floor(x/q+0.5); % 中宽量化器
    [Height,Width] = size(x_h);

    [p_h,Binx_h] = hist(x_h(:),max(x_h(:))-min(x_h(:))+1);% compute the histogram.
    p_h = p_h/(Height*Width);% normalize the histogram counts.

    H_h = sum(-p_h.* log2(p_h+1e-08));% compute the entropy % to avoid log(0), add a small positive value to p.
    disp(sprintf('Entropy of input image for channel %d = %6.2f',chl,H_h))
%     subplot(2,3,chl),plot(Binx_h,p_h,'k')
%     xlabel('Pixel value'), ylabel('relative count');
%     subplot(2,3,chl+3),imshow(x_h),title(sprintf('channel = %d', chl))

    %y_h = x_h * q;
    if chl>1
        y_h = blockproc(x_h,blksize,iqthreadC);
    else
        y_h = blockproc(x_h,blksize,iqthread);
    end
    img_yuv_rec(:,:,chl) = blockproc(y_h, blksize, img_idct2) +128;
end
img_yuv_rec = uint8(img_yuv_rec);
img_rec = ycbcr2rgb(img_yuv_rec);

mse_yuv= sum((double(img_yuv(:))-double(img_yuv_rec(:))).^2)/numel(img_yuv);
mse_rgb= sum((double(img(:))-double(img_rec(:))).^2)/numel(img);
psnr_rgb = 10*log10(255^2/mse_rgb);

ssim_rgb = ssim(img, img_rec);
figure
subplot(121),imshow(img),title('original iamge')
subplot(122),imshow(img_rec), title('reconstructed image')
disp(sprintf('SSIM: %6.4f; MSE: %6.2f, PSNR: %6.2f dB', ssim_rgb, mse_rgb, psnr_rgb))
```

将量化矩阵的等级设为3。调用`dct_jpeg_entropy_demo`，通过代码执行后的结果不难看出，JPEG量化矩阵能够取得不错的压缩效果。

```
%% Lec4Exp5: Influence of JPEG Quantization Matrix
close all; clear all
filename = 'data\Lenna.png';
q= 3;
dct_jpeg_entropy_demo(filename, q)
```

以下为代码执行后的输出结果：

```
Entropy of input image for channel 1 =   0.45
Entropy of input image for channel 2 =   0.17
Entropy of input image for channel 3 =   0.18
SSIM: 0.9714; MSE:  74.23, PSNR:  29.42 dB
```

3.3 图像有损压缩系统设计

在3.1和3.2的基础上，我们可以考虑针对图像数据，基于2D-DCT变换和JPEG量化矩阵，进行有损压缩系统的设计。`ImgLossyCodec_Encoder_Demo`为一个简单的图像压缩系统的函数实现示例，具体步骤包括：

- Step 1 预处理：读入原始图像数据，并将图像数据从RGB颜色空间转换到YCbCr颜色空间。完成参数设置，如码流文件的存放文件夹检查等。
- step 2 对图像数据的每个通道，利用matlab中的内置函数`blockproc`、`dct2`，完成图像信号的分块二维DCT变换和JPEG量化，调用`Lec4`中的图像单通道熵编码函数`encode_channel`，完成二进制码流的生成和保存。并打印编码时间和压缩倍数。
- step 3 对图像数据的每个通道，调用`Lec4`中的图像单通道熵解码函数`decode_channel`,再通过反量化和分块逆二维DCT变换，完成图像数据的重建，再重新将图像数据转换到RGB颜色空间。并打印解码时间和图像质量。

```
%%file ImgLossyCodec_Encoder_Demo.m
function [bpp, psnr_rgb, ssim_rgb]=ImgLossyCodec_Encoder_Demo(filename,q,flag, plotflag)
%filename = 'data\kodim12.png';
%q= 2;
img = imread(filename);
img_yuv = double(rgb2ycbcr(img));
img_yuv_offset = img_yuv-128;
img_dct2 = @(block_struct) dct2 (block_struct.data);
img_idct2 = @(block_struct) idct2 (block_struct.data);
blksize = [8 8];

jpgQstepsY = [16 11 10 16 24 40 51 61;
              12 12 14 19 26 58 60 55;
              14 13 16 24 40 57 69 56;
              14 17 22 29 51 87 80 62;
              18 22 37 56 68 109 103 77;
              24 35 55 64 81 104 113 92;
              49 64 78 87 103 121 120 101;
              72 92 95 98 112 100 103 99];

jpgQstepsC = [17 18 24 47 66 99 99 99;
              18 21 26 66 99 99 99 99;
              24 26 56 99 99 99 99 99;
              47 66 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99];
qthread = @(block_struct) block_struct.data./(jpgQstepsY*q);
iqthread = @(block_struct) block_struct.data.*(jpgQstepsY*q);

qthreadC = @(block_struct) block_struct.data./(jpgQstepsC*q);
iqthreadC = @(block_struct) block_struct.data.*(jpgQstepsC*q);
[Height,Width,d] = size(img_yuv_offset);
diffmode = 0;
size_dict = 0;
size_bitstream =0;
tic
for chl = 1:d
    x = blockproc(img_yuv_offset(:,:,chl),blksize,img_dct2);
    if chl>1
        x_h = floor(blockproc(x,blksize,qthreadC)+0.5);
    else
        x_h = floor(blockproc(x,blksize,qthread)+0.5);
    end
    htreefile = sprintf('dct_chl%d__htree.bin', chl);
    streamfile = sprintf('dct_chl%d__stream.bin', chl);
    [size_dict_chl, size_bitstream_chl]=encode_channel(x_h, htreefile, streamfile, diffmode);
    size_dict=size_dict+size_dict_chl;
    size_bitstream=size_bitstream+size_bitstream_chl;
end
if flag
    disp(sprintf('Image Encoding Finshed'));
    disp(sprintf('Encoding time:  %6.4f s', toc));
    disp(sprintf('Dict size:  %d Bytes', size_dict));
    disp(sprintf('Code size:  %d Bytes', size_bitstream));
    disp(sprintf('----------------'));
end
tic
img_yuv_rec = zeros(size(img_yuv));
for chl = 1:3
    htreefile = sprintf('dct_chl%d__htree.bin', chl);
    streamfile = sprintf('dct_chl%d__stream.bin', chl);
    x_hd = decode_channel(htreefile, streamfile, diffmode, [Height,Width]);
    if chl>1
        y_h = blockproc(x_hd,blksize,iqthreadC);
    else
        y_h = blockproc(x_hd,blksize,iqthread);
    end
    img_yuv_rec(:,:,chl) = blockproc(y_h, blksize, img_idct2) +128;
end
img_yuv_rec = uint8(img_yuv_rec);
img_rec = ycbcr2rgb(img_yuv_rec);
if flag
    disp(sprintf('Image Decoding Finshed'));
    disp(sprintf('Decoding time:  %6.4f s', toc));
end

mse_yuv= sum((double(img_yuv(:))-double(img_yuv_rec(:))).^2)/numel(img_yuv);
mse_rgb= sum((double(img(:))-double(img_rec(:))).^2)/numel(img);
psnr_rgb = 10*log10(255^2/mse_rgb);

%%Calculation of compression ratio
B0 = numel(img);
B1 = size_dict+size_bitstream;
compressionratio=B0/B1;
bpp = B1/(Height*Width)*8;
if flag
    disp(sprintf('Q-Step: %0.2f; Orginal:  %d Bytes; Compressed:  %d Bytes; Compression ratio:  %6.2f BPP: %0.6f',q, B0, B1, compressionratio,bpp));
end

ssim_rgb = ssim(img, img_rec);
if plotflag
    figure
    subplot(121),imshow(img),title('original iamge')
    subplot(122),imshow(img_rec), title('reconstructed image')
    end
if flag
    disp(sprintf('SSIM: %6.4f; MSE: %6.2f, PSNR: %6.2f dB', ssim_rgb, mse_rgb, psnr_rgb))
end
```

调用下面的示例代码`Lec5Exp5`

```
%% Lec4Exp6: Example of Image Lossy Codec
close all, clear all
filename = 'data\kodim12.png';
q= 2;
flag = 1;
plotflag = 1;
[bpp, psnr_rgb, ssim_rgb]=ImgLossyCodec_Encoder_Demo(filename,q,flag,plotflag);
```

以下为代码执行后的输出结果：

```
Image Encoding Finshed
Encoding time:  6.5933 s
Dict size:  1024 Bytes
Code size:  160650 Bytes
----------------
Image Decoding Finshed
Decoding time:  20.9045 s
Q-Step: 2.00; Orginal:  1179648 Bytes; Compressed:  161674 Bytes; Compression ratio:    7.30 BPP: 3.289266
SSIM: 0.9482; MSE:  40.62, PSNR:  32.04 dB
```

3.4 图像有损压缩系统性能评估

系统性能的评价指标包含多种判断标准，如计算复杂度、压缩率等。本实验中，我们主要考虑压缩率。并将所设计的有损压缩系统与matlab中默认的JPEG的编解码器进行性能比较。具体的，对每种编解码器，我们通过调节量化步长或者质量因子等控制参数，来获取对应的<码率，图像质量>数据。其中码率用位率（Bit Per Pixel，bpp）表示。图像质量用所有RGB分量的SSIM，或者均方差的平均值计算得到的PSNR来表示。最后，绘制出相应的图像质量与位率的关系曲线。通过曲线可以很容易判断出，不同编解码器的性能优劣。

首先执行下面的代码块`Lec5Exp7`中的第一部分，通过调节量化步长q的取值，完成对所提出的编解码器的性能测试。为后续的率失真性能对比提供数据支撑。

```
%%Lec4Exp7
clear all, close all
filename = 'data\kodim12.png';
q = [1, 2, 4, 6];
flag = 1; % 开启/关闭编解码器运行报告
plotflag = 0; % 开启/关闭原始/重建图像效果对比图
for i = 1 : 4
    [bpp, psnr_rgb, ssim_rgb]=ImgLossyCodec_Encoder_Demo(filename,q(i),flag, plotflag);
    bpp_vector(i) = bpp;
    psnr_rgb_vector(i) = psnr_rgb;
    ssim_rgb_vector(i) = ssim_rgb;
end
```

以下为代码执行后的部分输出结果：

```
Q-Step: 1.00; Orginal:  1179648 Bytes; Compressed:  172577 Bytes;
Compression ratio:    6.84 BPP: 3.511088
SSIM: 0.9682; MSE:  22.68, PSNR:  34.57 dB
    ----------------
Q-Step: 2.00; Orginal:  1179648 Bytes; Compressed:  161674 Bytes;
Compression ratio:    7.30 BPP: 3.289266
SSIM: 0.9482; MSE:  40.62, PSNR:  32.04 dB
    ----------------
 Q-Step: 4.00; Orginal:  1179648 Bytes; Compressed:  155305 Bytes;
Compression ratio:  7.60 BPP: 3.159688
SSIM: 0.9095; MSE:  76.40, PSNR:  29.30 dB
    ----------------
Q-Step: 6.00; Orginal:  1179648 Bytes; Compressed:  152805 Bytes;
Compression ratio:    7.72 BPP: 3.108826
SSIM: 0.8791; MSE: 109.49, PSNR:  27.74 dB
```

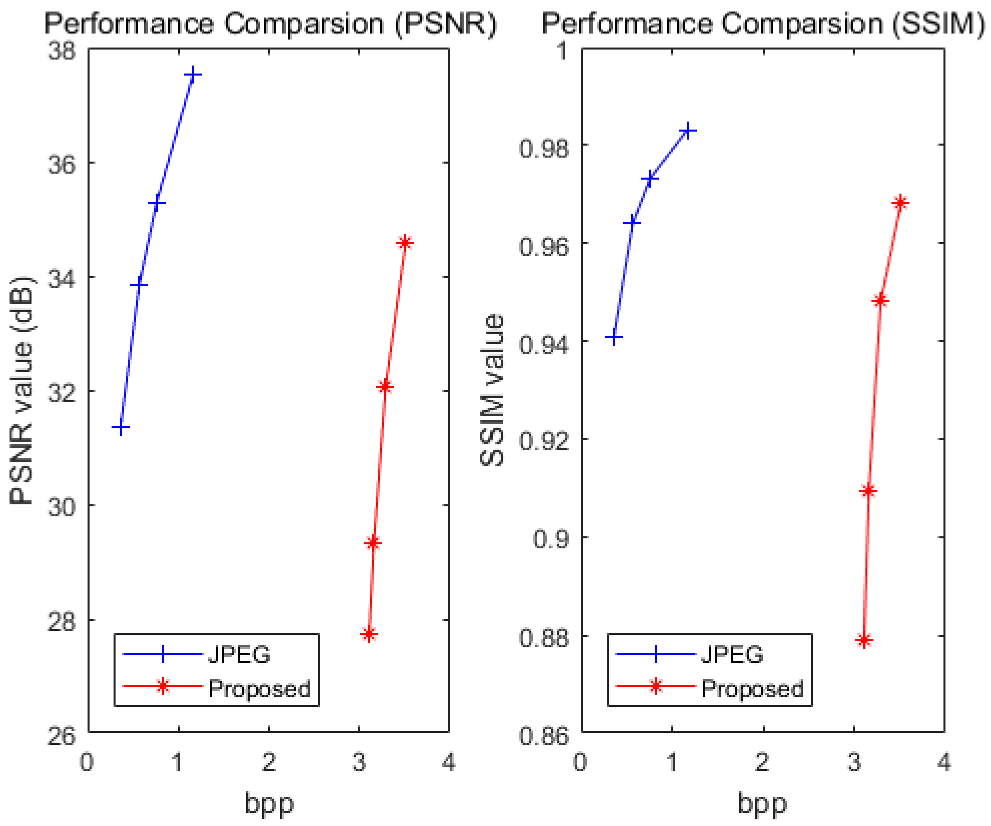
继续执行以下代码，完成matlab中自带的JPEG编解码器的性能测试：

```
img = imread(filename);
[h, w, c] = size(img);
qfactor = [80, 60, 40, 20];
for i = 1 : 4
    imwrite(img, 'test.jpg', 'quality', qfactor(i));
    file = dir('test.jpg');
    bpp_vector_jpeg(i) = file.bytes/(h*w)*8;

    img_rec = imread('test.jpg');
    ssim_rgb_jpeg(i) = ssim(img, img_rec);

    mse_rgb= sum((double(img(:))-double(img_rec(:))).^2)/numel(img);
    psnr_rgb_jpeg(i) = 10*log10(255^2/mse_rgb);
end
```

执行以下代码，画出所设计的编解码器，以及matlab中自带的JPEG编解码器的图像质量-位率曲线：

```
figure
subplot(121),plot(bpp_vector_jpeg,psnr_rgb_jpeg,'+-b'),
hold on
plot(bpp_vector,psnr_rgb_vector,'*-r')
legend('JPEG','Proposed','Location','southwest')
title('Performance Comparsion (PSNR)')
xlabel('bpp'),ylabel('PSNR value (dB)')

subplot(122),plot(bpp_vector_jpeg,ssim_rgb_jpeg,'+-b'),
hold on
plot(bpp_vector,ssim_rgb_vector,'*-r')
legend('JPEG','Proposed','Location','southwest')
title('Performance Comparsion (SSIM)')
xlabel('bpp'),ylabel('SSIM value')
```

以下为代码执行后的输出结果：



从上面的性能比较图中可以看出，matlab中自带的JPEG编解码器的性能比我们所提出的编解码系统，在率失真性能上有明显的优势。比如相同的PSNR/SSIM值下，JPEG编解码器需要更少的位率。

4 课后习题

- 1）3.3.节中的`ImgLossyCodec_Encoder_Demo`中还存在很多需要优化的地方,比如量化后的DCT系数的熵编码的过程。JPEG编码标准中针对DC和AC系数的特点，分别采用了DPCM，Z字形扫描和游程编码（Run Length Coding, RLC）等技术以消除频率系数间的空间相关性。再次基础上，为了减少赫夫曼编解码中的符号表过大的问题，设计了赫夫曼编码的变体方案，进一步降低了统计冗余，提高了压缩效率。请尝试在代码基础上，去复现JPEG编码标准中的相应的技术。
- 2）3.4节中采用了PSNR/SSIM两个指标来评估压缩重建后的图像画质，请查阅文献，探索一下图像质量评价领域中的其他平均指标。另外，3.4节中只给出了定性的分析方法，如何定量分析不同的编解码器性能的优劣呢？请以BD-PSNR或者BD-Rate为关键词来寻找相应的解决方案。