实验3-无损图像压缩系统设计

1 实验目的

（1）掌握信号信息熵的计算方法，了解数据表示方式对信息熵的影响。

（2）熟悉熵编解码器，如huffman和算术编解码器等的使用流程和码流读写方法。

（3）熟悉图像无损压缩系统设计的基本流程。

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 图像信号的信息熵

待处理图像信号的信息熵反映了信号无损压缩的难易程度。假设8位灰度图像，假设图像信息的直方图的灰度分布呈均匀分布，即对于任意的$i \in [0, 255]$，$p_i = \frac{1}{256}$,则该图像的熵为：

$$\eta = \sum_{i=0}^{255} \frac{1}{256} \cdot \log_2 256 = 8 \tag{1}$$

不难证明，图像的熵的上界为8。

以下将讨论图像的数据表示方法，以及差分预测模式等对图像数据的信息熵的影响。

3.1.1 RGB颜色模型

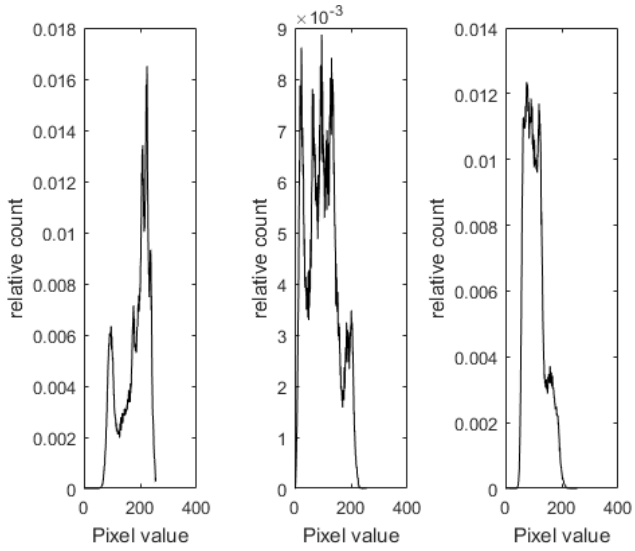
以`Lenna.png`为例，执行案例`Lec3Exp1`以分析该图像在RGB颜色模型下的信息熵情况：matlab内置函数`imhist`可用于统计uint8图像数据的每个通道的直方图。通过实验可以发现，RGB颜色模型下3个通道的信息熵差别不大，都很接近上界8。

```
%plot inline
%% Lec3Exp1: Computes the entropy of the image data in RGB Color Space
close all; clear all
format compact

filename = 'data\Lenna.png';
img = imread(filename);
for chl= 1:3
    x = img(:,:,chl);
    [Height,Width] = size(x);
    [p,Binx] = imhist(x);% compute the histogram with 256 bins.

    p = p/(Height*Width);% normalize the histogram counts.

    H = sum(-p.* log2(p+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))
    subplot(1,3,chl),plot(Binx,p,'k')
    xlabel('Pixel value'), ylabel('relative count');
end

```

以下为代码执行后的输出结果：

```
Entropy of input image for channel 1 =   7.25
Entropy of input image for channel 2 =   7.59
Entropy of input image for channel 3 =   6.97
```



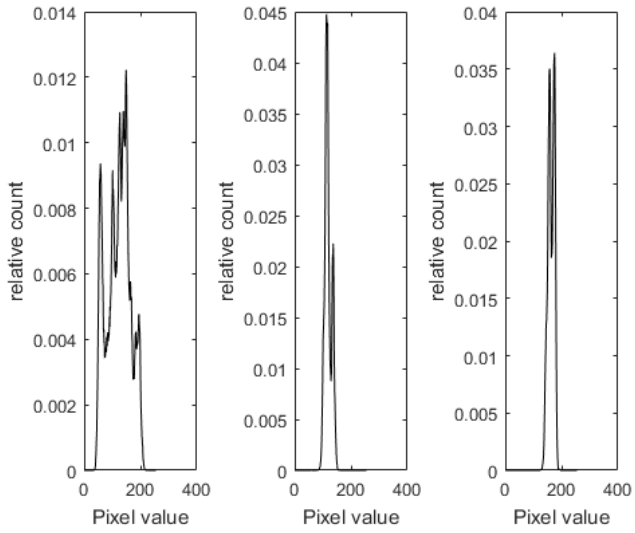
3.1.2 YCbCr颜色模型

YCbCr颜色模型包含一个亮度通道和两个色差通道。执行案例 `Lec4Exp2` 以分析该图像在YCbCr颜色模型下的信息熵情况：图像数据通过matlab内置函数 `rgb2ycbcr` 进行转换，再重新计算各通道的信息熵。不难发现，Y亮度通道的信息熵和RGB颜色模型中的信息熵接近。但是Cb和Cr两个色差通道的信息熵显著下降。即无损压缩难度降低。

```
%% Lec3Exp2: Computes the entropy of the image data in YCbCr Color Space
img_yuv = rgb2ycbcr(img);
for chl= 1:3
    x = img_yuv(:,:,chl);
    [Height,Width] = size(x);
    [p,Binx] = imhist(x);% compute the histogram with 256 bins.

    p = p/(Height*Width);% normalize the histogram counts.
    %sprintf('Sum of hist. values = %g',sum(p))% verify that sum(p) = 1
    H = sum(-p.* log2(p+1e-08));% compute the entropy
    disp(sprintf('Entropy of input image for channel %d = %6.2f',chl,H))
    subplot(1,3,chl),plot(Binx,p,'k')
    xlabel('Pixel value'), ylabel('relative count');
end
```

以下为代码执行后的输出结果：

```
Entropy of input image for channel 1 =   7.23
Entropy of input image for channel 2 =   5.47
Entropy of input image for channel 3 =   5.42
```



3.1.3 图像的差分编码

记原始图像$\mathbf{img}$的通道d的像素位置(i,j)的灰度值为$\mathbf{img}(i,j,d)$，则其水平方向的差分图像$\mathbf{diff}_h$的像素值定义为:

$$\mathbf{diff}_h(i,j,d)=\mathbf{img}(i,j,d)-\mathbf{img}(i,j-1,d) \tag{2}$$

垂直方向的差分图像$\mathbf{diff}_v$的像素值定义为:

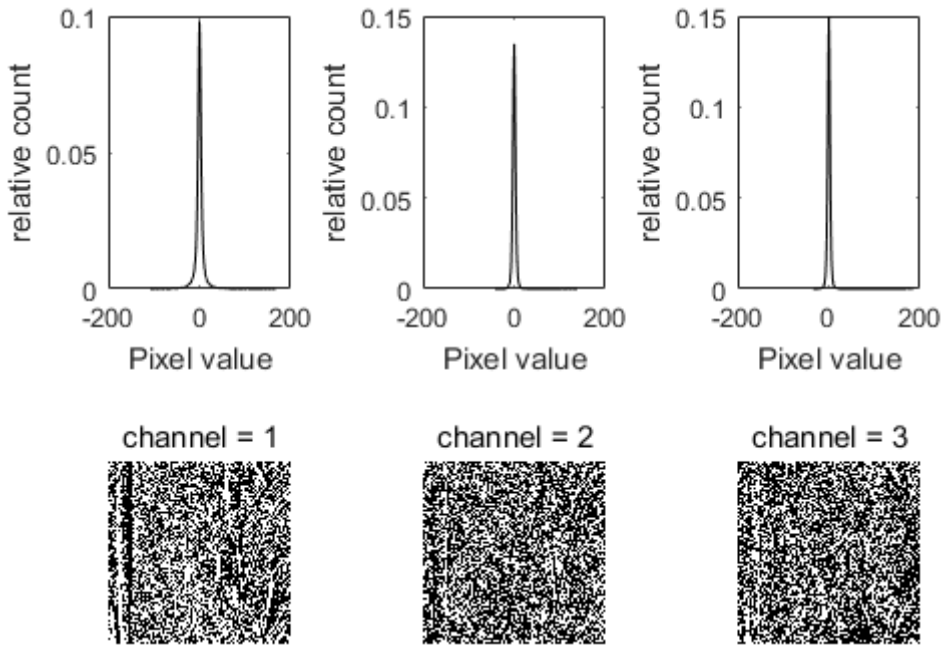
$$\mathbf{diff}_v(i,j,d)=\mathbf{img}(i,j,d)-\mathbf{img}(i-1,j,d) \tag{3}$$

注意：水平（垂直）方向的差分图像第一列（行）的数据直接复制原图像的对应列（行）。该差分图像的值域范围的上界为$[-255,255]$。需要注意的是，此时由于值域范围已超出uint8的表示范围，需要采用matlab内置函数`hist`来进行直方图的统计。同时，需要根据图像数据的灰度值分布范围来确定huffman tree中的符号表取值范围: $range=[\min:1:\max]$。执行案例`Lec4Exp3`和`Lec4Exp4`,观察得出，对于`Lenna.png`,对应两个色差通道，其水平差分图像的信息熵明显低于垂直分类图像的信息熵，即水平相关性比垂直相关性更强。

```
%% Lec3Exp3: Computes the entropy of the horizontal difference image data
for chl= 1:3
    img_chl = double(img_yuv(:,:,chl));
    [Height,Width] = size(img_chl);
    x_h = img_chl;
    x_h(:,2:Width) = img_chl(:,2:Width)-img_chl(:,1:Width-1);

    [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 horizontal difference 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 ))
end
```

以下为代码执行后的输出结果：

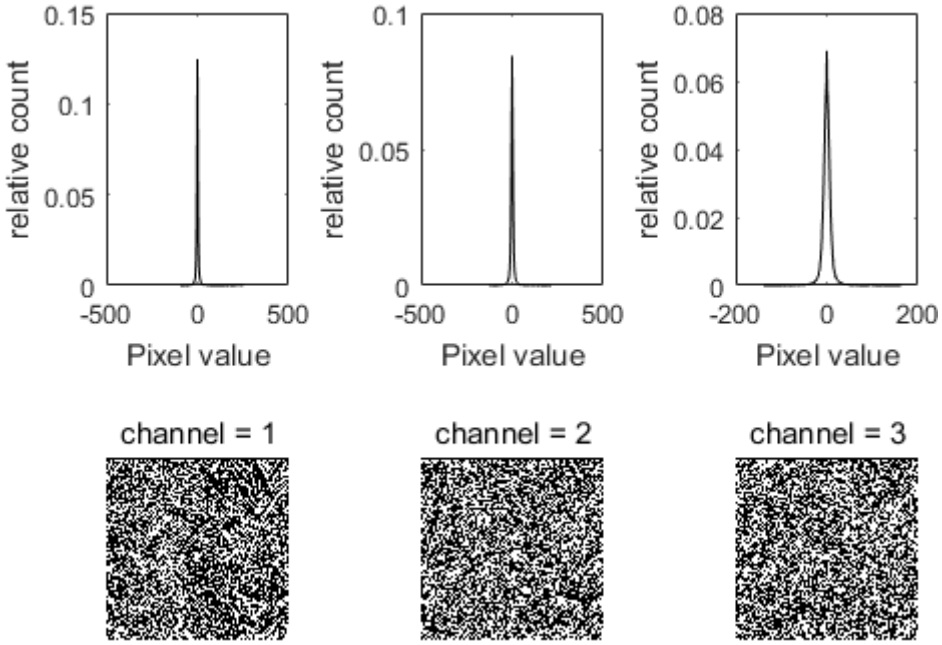
```
Entropy of input horizontal difference image for channel 1 =   4.85
Entropy of input horizontal difference image for channel 2 =   3.89
Entropy of input horizontal difference image for channel 3 =   3.78
```




```
%% Lec3Exp4：Computes the entropy of the vertical difference image data
for chl= 1:3
    img_chl = double(img(:,:,chl));
    [Height,Width] = size(img_chl);
    x_v = img_chl;

    x_v(2:Height,:) = img_chl(2:Height,:)-img_chl(1:Height-1,:);
    [p_v,Binx_v] = hist(x_v(:),max(x_v(:))-min(x_v(:))+1);% compute the
histogram.
    p_v = p_v/(Height*Width);% normalize the histogram counts.

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

以下为代码执行后的输出结果：

```
Entropy of input vertical difference image for channel 1 =   4.50
Entropy of input vertical difference image for channel 2 =   5.02
Entropy of input vertical difference image for channel 3 =   5.15
```



3.2 熵编解码器

3.2.1 Huffman 编解码器

matlab中的Communication Toolbox，提供了Huffman编解码器的内置函数，分别是`huffmandict`,`huffmanenco`和`huffmandeco`。在代码块`lec4exp5`中提供了Huffman编解码的一个简单示例demo。

- `huffmandict`：输入为信源的符号集合`symobols`和对应的概率值`prob`，输出为Huffman树`dict`,包含每个`symobol`对应的`code`。
- `huffmanenco`：输入为Huffman字典`dict`和待压缩信号`sig`,输出为对应的二进制码流`hcode`。
- `huffmandeco`：输入为Huffman字典`dict`和待解码的二进制码流`hcode`,输出为重建的信号`dhsig`。

需要注意的是，在实际的编解码压缩系统中，编码器和解码器是放在不同的终端上的。编码器一般部署在服务端，输出信号的二进制码流表示。服务端将Huffman字典`dict`和二进制码流`hcode`打包成文件，再以网络传输等各种形式发送给客户端。客户端收到码流文件后，需要解析码流文件，还原出Huffman字典`dict`和二进制码流`hcode`，再调用解码器进行解码，完成信号重建。由于huffman编码属于非定长编码，代码块`lec4exp5`中分别调用了4个函数，分别完成Huffman字典`dict`和二进制码流`hcode`的保存和读取解析。

- `savehtree`：保存Huffman字典`dict`
- `savestreamfile`：保存二进制码流`hcode`
- `readhtree`：读取Huffman字典`dict`
- `readstreamfile`：读取Huffman字典`hcode`

huffman tree在matlab中实现的数据结构为Len(symbols)x2大小的元胞cell。故文件保存时按`<symbol_i> <code_i>`的方式成对顺序写入，并添加空格字符作为分隔符，以方便读取时的数据解析。以下代码块`savehtreex`和`readhtree`为相应的读写函数实现方式。

```
%%file savehtree.m
function size_dict = savehtree(htreefile, dict)
tb = fopen(htreefile,'w');
size_dict = 0;
for i=1:size(dict,1)
    size_dict=size_dict+fprintf(tb,'%s',num2str(dict{i,1}));
```

```
    size_dict=size_dict+fprintf(tb,'%c',' ');
    size_dict=size_dict+fprintf(tb,'%s',string(dict{i,2}));
    if i < size(dict,1)
        size_dict=size_dict+fprintf(tb,'%c',' ');
    end
end
fclose(tb);
end
```

```
%%file readhtree.m
function dict_d = readhtree(htreefile)
fid = fopen(htreefile,'r');
tmpd = fscanf(fid,'%c');
fclose(fid);
tmp = split(tmpd);
for i = 1:size(tmp)/2
    dict_d{i,1} = str2num(tmp{i*2-1});
    tmp_array = [];
    for j = 1:size(tmp{i*2},2)
        tmp_array=[tmp_array str2double(tmp{i*2}(j))];
    end
    dict_d{i,2} = tmp_array;
end
end
```

`hcode`为二值一维数组。文件保存时，每八位合成一个`uint8`,写入文件。由于非定长编码，若最后不足8位时，在末尾补 $\mod(Code_Len,8)$个0。并追加一个字节，其值为 $\mod(Code\_Len,8)$。为了提高读写效率，采用matlab内置按位操作函数`bitset`和`bitget`对`hcode`向量化操作。

BIT	8	7	6	5	4	3	2	1
Code	x	x	x	0	0	0	0	0
$\mod(Code\_Len,8)$	0	0	0	0	0	1	0	1

以下代码块`savestreamfile`和`readstreamfile`为相应的读写函数实现方式:

```
%%file savestreamfile.m
function count_stream = savestreamfile(streamfile, bitstream)
count_stream = 0;
stream_len = numel(bitstream);
padding_len = 8-mod(stream_len,8);
all_len = stream_len+padding_len;
bitstream_temp = zeros(all_len,1);
bitstream_temp(1:stream_len)=bitstream;
bitstream_array = reshape(bitstream_temp, 8, floor(all_len/8));
buffer = uint8(zeros(floor(all_len/8),1));
for BIT = 1:8
    buffer = bitset(buffer, 8-BIT+1, bitstream_array(BIT,:)');
end
tstream = fopen(streamfile,'w');
count_stream = fwrite(tstream,buffer,'uint8');
count_stream = count_stream+fwrite(tstream,padding_len,'uint8');
fclose(tstream);
end
```

```
%%file readstreamfile.m
function bitstream_d  = readstreamfile(streamfile)
fid = fopen(streamfile,'r');
buffer = fread(fid,'uint8');
fclose(fid);
count_stream = numel(buffer);
all_len_byte = count_stream - 1;
padding_len = buffer(count_stream);
stream_len = all_len_byte*8-padding_len;
bitstream_array = zeros(8, count_stream);
for BIT = 1:8
    bitstream_array(BIT, :) = bitget(buffer, 8-BIT+1)';
end
bitstream_array = bitstream_array(:);
bitstream_d = bitstream_array(1:stream_len)';
end
```

执行代码块`lec3exp5`：

```
%function lec3exp5
close all, clear all
%addpath('./script')
symbols = [1:5];
prob = [0.6, 0.1, 0.1, 0.1, 0.1];
sig_len = 10000;
sig = randsrc(1, sig_len, [symbols; prob]);

%%setting
streampath = 'stream';
if ~isdir(streampath)
    mkdir(streampath);
end
htreefile  = sprintf('%s\\huffman_htree.bin',streampath);  %压缩后需要保存的huffman tree文件名
streamfile = sprintf('%s\\huffman_stream.bin',streampath); %压缩后需要保存的二进制码流文件名

tic
disp(sprintf('Building Huffman tree\r\n ----------------'));
dict = huffmandict(symbols,prob);
disp('Symbol: Code');
for i = 1:size(dict,1)
    disp(sprintf('   %s:    %s',num2str(dict{i,1}),num2str(dict{i,2})))
end
disp(sprintf('----------------'));

%%huffman encoding
hcode = huffmanenco(sig,dict);

%%save the huffman tree and stream file
size_dict = savehtree(htreefile, dict);
size_bitstream = savestreamfile(streamfile, hcode);
disp(sprintf('Huffman Encoding Finshed'));
disp(sprintf('Encoding time:  %6.4f s', toc));
disp(sprintf('htreefile size:  %d Bytes', size_dict));
disp(sprintf('code size:  %d Bytes \r\n ----------------', size_bitstream));
```

```
tic
%%read the huffman tree and stream file
dict_d = readhtree(htreefile);
hcode_d  = readstreamfile(streamfile);

%%huffman decoding
dhsig = huffmandeco(hcode_d,dict);
disp(sprintf('Huffman Decoding Finshed'));
disp(sprintf('Decoding time:  %6.4f s', toc));
if isequal(sig,dhsig);
    disp('Huffman Lossy Compression Success!')
else
    disp('Huffman Lossy Compression failed!')
end

%% Performance discussion
H = sum(-prob.* log2(prob));% compute the entropy
%disp(sprintf('Entropy:%6.2f; Theortic Average code length:%6.2f',H,
numel(hcode)/numel(sig)));
disp(sprintf('Entropy:%6.2f; Average code length:%6.2f',H,
(size_dict+size_bitstream)*8/numel(sig)));
```

以下为代码执行后的输出结果：

```
Building Huffman tree
 ----------------
Symbol: Code
   1:    0
   2:    1  0  1
   3:    1  0  0
   4:    1  1  1
   5:    1  1  0
----------------
Huffman Encoding Finshed
Encoding time:  0.1210 s
htreefile size:  27 Bytes
code size:  2248 Bytes
 ----------------
Huffman Decoding Finshed
Decoding time:  0.1101 s
Huffman Lossy Compression Success!
Entropy:  1.77; Average code length:  1.82
```

3.2.2 算术编解码器

matlab中的Communication Toolbox中自带了算术编解码器的内置函数，分别是`arithenco`,`arithdeco`。

- `arithenco`：输入为待压缩信号`sig_mapped`和对应符号计数表`prob`,输出为对应的二进制码流`accode`。
- `arithdeco`：输入为符号计数表`prob`和待解码的二进制码流`accode`和需要重建的信号长度,输出为重建的信号`sig_mapped_d`。

需要注意的是，matlab中的算法编码器，只支持数值符号集合（从1开始计数），故对于任意符号集，需要建立一个映射表，将原始待压缩信号`sig`通过自定义的映射函数`arith_sig_mapping`，映射为从1开始计数的数值型信号`sig_mapped`。并将映射表和相应的计数表`prob`，以及信号长度信息均保存在头文件中。客户端解码后再通过自定义的逆映射函数`arith_sig_invmapping`,完成信号的重建。

执行代码块`lec3exp6`：

```
%lec3exp6
%%setting
actablefile  = sprintf('%s\\ac_table.bin',streampath);  %压缩后需要保存的概率表文件
acstreamfile = sprintf('%s\\ac_stream.bin',streampath); %压缩后需要保存的二进制码流文件名
prob = prob*sig_len;

tic
%%arithmetic encoding
sig_mapped = sig;
%[sig_mapped, map] = arith_sig_mapping(sig)
accode = arithenco(sig_mapped,prob);

%%write the AC symbos mapping table and stream file
size_actable = 0;
len_sig = numel(sig_mapped);
% size_actable = savetable(actablefile, map, prob_d, len_sig);
size_acbitstream = savestreamfile(streamfile, accode);

disp(sprintf('AC Encoding Finshed'));
disp(sprintf('Encoding time:  %6.4f s', toc));
disp(sprintf('table size:  %d Bytes', size_actable));
disp(sprintf('code size:  %d Bytes \r\n ----------------', size_acbitstream));

tic
%%read the AC symbos mapping table and stream file
% [map, prob_d, len_sig] = readtable(tablefile);
accode_d = readstreamfile(streamfile);

%% arithmetic decoding
sig_mapped_d = arithdeco(accode_d,prob,len_sig);
%dacsig = arith_sig_invmapping(sig_mapped_d, map)
dacsig = sig_mapped_d;

disp(sprintf('AC Decoding Finshed'));
disp(sprintf('Decoding time:  %6.4f s', toc));
if isequal(sig,dacsig)
    disp('AC Lossy Compression Success!')
else
    disp('AC Lossy Compression failed!')
end

%% Performance discussion
disp(sprintf('Entropy:%6.2f; Average code length:%6.2f',H,(size_dict+size_bitstream)*8/numel(sig)));
```

以下为代码执行后的输出结果：

```
AC Encoding Finshed
Encoding time:  0.1645 s
table size:  0 Bytes
code size:  2213 Bytes
 ----------------
AC Decoding Finshed
Decoding time:  0.1748 s
AC Lossy Compression Success!
Entropy:  1.77; Average code length:  1.82
```

3.3 图像无损压缩系统设计

在3.1和3.2的基础上，我们可以考虑针对图像数据，进行无损压缩系统的设计。`ImgCodec_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`hist`，完成图像信号的概率统计分析，调用自定义单通道编码函数`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，调用自定义单通道解码函数`decode_channel`,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。并打印解码时间和判断图像数据是否无损。

以下分别为单通道编码函数`encode_channel`和单通道解码函数`decode_channel`的函数实现示例。其中，为了消除空间冗余，引入水平差分预测方法，并用参数`diffmode`来进行控制。当`diffmode`为0时关闭差分预测模式。

```
%%file encode_channel.m
function [size_dict, size_bitstream]=encode_channel(img, htreefile,
streamfile, diffmode)
%% 信号源的统计特性分析
if diffmode
    img = double(img);
    [Height,Width] = size(img);
    sig = img;
    sig(:,2:Width) = img(:,2:Width)-img(:,1:Width-1);
    sig = sig(:);
else
    sig = double(img(:));
end
[prob,symbols] = hist(sig,[min(sig):1:max(sig)]);% compute the histogram.
prob = prob/numel(sig);

%% 赫夫曼编码
dict = huffmandict(symbols,prob); %huffman tree
bitstream = huffmanenco(sig,dict);

%% 保存为码流文件（注意文件尺寸）
size_dict = savehtree(htreefile, dict);
size_bitstream = savestreamfile(streamfile, bitstream);
end
```

```
%%file decode_channel.m
function img = decode_channel(htreefile, streamfile, diffmode, imgsize)
%% 读取码流文件
```

```
dict_d = readhtree(htreefile);
bitstream_d = readstreamfile(streamfile);

dhsig = huffmandeco(bitstream_d,dict_d);
diffimg = reshape(dhsig, imgsize(1), imgsize(2));

%% 差分预测模式重建
if diffmode
    img = zeros(imgsize);
    img(:,1) = diffimg(:,1);
    for w = 2: imgsize(2)
        img(:,w) = diffimg(:,w)+img(:,w-1);
    end
else
    img = diffimg;
end
end
```

```
%%file ImgCodec_Encoder_Demo.m
function ImgCodec_Encoder_Demo(filename, ratio, diffmode)
%close all, clear all
%% 读入待压缩图像原始数据
%filename = 'data\\Lenna.png';
%filename = 'data\\Weeki_Wachee_spring.jpg';
img_rgb = imread(filename);
img_rgb = imresize(img_rgb, 1/ratio);
[h, w, d] = size(img_rgb);
if d > 1
    img_yuv = rgb2ycbcr(img_rgb);
end

% 码流存放路径
stream_path = 'stream';
if ~isdir(stream_path)
    mkdir(stream_path);
end

% 文件名解析
[pathstr, name, ext] = fileparts(filename);

% 预测模型参数设置：0/1：不开启/开启差分模式
%diffmode = 0; %
if diffmode
    disp(sprintf('Difference prediction mode is ON! \r\n--------------------'));
else
    disp(sprintf('Difference prediction mode is OFF! \r\n-------------------'));
end
%% 对图像的各个通道进行单独编码，每个通道输出htree.bin和stream.bin两个文件。
tic
size_dict=0; % huffman tree的文件的大小(Byte)
size_bitstream=0; % 生成的码流文件的总大小(Byte)
if d == 1
    htreefile = sprintf('%s\\%s_m%d_htree.bin',stream_path,name,diffmode);
    streamfile =
sprintf('%s\\%s_m%d_stream.bin',stream_path,name,diffmode);
```

```
    [size_dict, size_bitstream]=encode_channel(img_rgb, htreefile,
streamfile, diffmode);
else
    for chl=1:d
        htreefile =
sprintf('%s\\%s_chl%d_m%d_htree.bin',stream_path,name,chl,diffmode);
        streamfile =
sprintf('%s\\%s_chl%d_m%d_stream.bin',stream_path,name,chl,diffmode);
        [size_dict_chl,
size_bitstream_chl]=encode_channel(img_yuv(:,:,chl), htreefile, streamfile,
diffmode);
        size_dict=size_dict+size_dict_chl;
        size_bitstream=size_bitstream+size_bitstream_chl;
    end
end
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));

%%Calculation of compression ratio
B0 = numel(img_rgb);
B1 = size_dict+size_bitstream;
compressionratio=B0/B1;
disp(sprintf('Orginal:  %d Bytes; Compressed:  %d Bytes; Compression ratio:
%6.2f \r\n ----------------',B0, B1, compressionratio));

%% 对图像的各个通道进行单独解码，每个通道首先解析相应的htree.bin文件，生成Huffman
TreeB0,再对stream.bin文件进行解码和重建。
tic
% size_dict=0; % 读取的huffman tree的文件大小(Byte)
% size_bitstream=0; % 读取的码流文件的总大小(Byte)
img_yuv_rec = zeros([h,w,d]);
if d == 1
    htreefile = sprintf('%s\\%s_m%d_htree.bin',stream_path,name,diffmode);
    streamfile =
sprintf('%s\\%s_m%d_stream.bin',stream_path,name,diffmode);
    img_rec = decode_channel(htreefile, streamfile, diffmode, [h, w]);
else
    for chl=1:d
        htreefile =
sprintf('%s\\%s_chl%d_m%d_htree.bin',stream_path,name,chl,diffmode);
        streamfile =
sprintf('%s\\%s_chl%d_m%d_stream.bin',stream_path,name,chl,diffmode);
        img_yuv_rec_chl = decode_channel(htreefile, streamfile, diffmode,
[h, w]);
        img_yuv_rec(:,:,chl) = img_yuv_rec_chl;
    end
    img_yuv_rec = uint8(img_yuv_rec);
    img_rec = ycbcr2rgb(img_yuv_rec);
end
disp(sprintf('Image Decoding Finshed'));
disp(sprintf('Decoding time:  %6.4f s', toc));
mse_yuv= sum((double(img_yuv(:))-
double(img_yuv_rec(:))).^2)/numel(img_yuv);
mse_rgb= sum((double(img_rgb(:))-double(img_rec(:))).^2)/numel(img_rgb);

if mse_yuv
```

```
    disp('Image Lossy Compression Failed!')
else
    disp('Image Lossy Compression Success!')
end
subplot(121),imshow(img_rgb), title(sprintf('original: %s', name))
subplot(122),imshow(img_rec), title(sprintf('MSE YUV: %6.2f; RGB: %6.2f',
mse_yuv, mse_rgb))

```

读入图像数据，开启差分预测模式，执行以下代码：

```
%lec3exp7
close all, clear all
%% 读入待压缩图像原始数据
filename = 'data\\Lenna.png';
%filename = 'data\\Weeki_Wachee_spring.jpg';
ratio = 1;
ImgCodec_Encoder_Demo(filename, ratio, 1)
```

以下为差分预测模式开启后的输出结果：

```
Difference prediction mode is ON!
-------------------------
Image Encoding Finshed
Encoding time:  10.4095 s
Dict size:  22275 Bytes
Code size:  412967 Bytes
Orginal:  786432 Bytes; Compressed:  435242 Bytes; Compression ratio:
1.81
 ----------------
Image Decoding Finshed
Decoding time:  360.4501 s
Image Lossy Compression Success!
```

关闭差分预测模式，执行以下代码：

```
ImgCodec_Encoder_Demo(filename, ratio, 0)
```

以下为差分预测模式关闭后的输出结果：

```
Difference prediction mode is OFF!
------------------------
Image Encoding Finshed
Encoding time:  10.4310 s
Dict size:  5990 Bytes
Code size:  596815 Bytes
Orginal:  786432 Bytes; Compressed:  602805 Bytes; Compression ratio:
1.30
 ----------------
Image Decoding Finshed
Decoding time:  326.5554 s
Image Lossy Compression Success!
```

从上面的实验结果可以看出，差分预测模式的开启，可以大大提高压缩性能。但由于字典变大，字典文件所需要的存储空间显著增大。同时，解码复杂度也会提升。

4 课后习题

- 1）3.3.节中的`ImgCodec_Encoder_Demo`的熵编解码器是基于huffman编码实现的。请在3.2.2节的基础，完成基于算术编码的图像编解码器的设计。包括相关符号表映射`arith_sig_mapping`/逆映射函数`arith_sig_invmapping`的实现。以及头信息（映射表和相应的计数表`prob`，以及信号长度信息）的保存/读入解析函数的实现。
- 2) 3.3.节中的`ImgCodec_Encoder_Demo`中还存在很多需要优化的地方,包括但不限于：1）编解码器的复杂度优化：huffman编解码器的C/C++代码实现效率较高，可以在Matlab中以MEX方式调用相应的动态链接库，需要注意matlab mex方式的接口设计要求。2）编码性能的提升：是否有其他可以更好的预测编码方式或者数据表示方法?