图像处理

勾天润 2020012321 无 03

August 2022

目录

1 基础知识

1.1

使用 help images 指令查看图像处理相关函数。

1.2 画圆、棋盘

在 help images 给出的函数中，发现 viscircles 可用来在坐标区内画圆。使用这一函数，给定圆心、半径、颜色，画好后获取句柄，使用 getframe 获取 frame，再使用 frame2im 获取像素矩阵。

```
r=min(w/2,h/2);
fig=figure;imshow(hall_color);
viscircles([w/2 h/2],r,'color','r');%w、h为提前获取的宽度、高度
frame = getframe(fig);
circle = frame2im(frame);
```

但这样操作的问题在于，生成的 circle 像素矩阵和 hall_color 不一致。原因在于 getframe 是从屏幕截取图片，和原来的大小无关。
于是改换方法，直接针对像素矩阵进行操作，把要画圆的像素点的 R 设为 255，G、B 设为 0 即可。在此过程中，合理使用 logical、拼接函数 cat，进行矩阵索引，可以规避循环的使用。

```
circle=hall_color;
r=min(w/2,h/2);
if_red=(x-w/2).^2+(y-h/2).^2≤r*r&(x-w/2).^2+(y-h/2).^2≥0.9*r*r;
%是否涂红的逻辑矩阵
draw_red=cat(3,if_red,false(h,w),false(h,w));
draw_gb=cat(3,false(h,w),if_red,if_red);
circle(draw_red)=255;
circle(draw_gb)=0;%逻辑值索引，进行赋值
figure;imshow(circle);
```

画棋盘与画圆类似，以 8 为单位分格，获取是否涂黑的逻辑矩阵，以此对像素矩阵进行赋值。

```
chessboard=hall_color;
if_black=mod(floor(x/8)+floor(y/8),2)==0;%横纵坐标加起来的数的奇偶性
draw_black=cat(3,if_black,if_black,if_black);
chessboard(draw_black)=0;%利用逻辑值索引、赋值
figure;imshow(chessboard);
```

最后使用 imwrite 函数将图片保存下来。

```
imwrite(circle,'circle.png');
imwrite(chessboard,'chessboard.png');
```

这一部分主要学习了逻辑值进行索引的强大之处。以前我只会使用正整数索引，为了发挥矩阵运算的强大之处，逻辑索引和正整数索引一样重要。也学习了 imshow、imwrite 用法。

图 1: 画圆

图 2: 棋盘

2 图像压缩编码

2.1

可以在变换域进行。

在变换域进行的意思是对未预处理直接 DCT 变换得到的系数矩阵 C 操作。

$$c = Dp$$

$$c = D(p + 128ones(N)) - 128D * ones(N)$$

所以，如果要在变换域给灰度减 128，只需要给 c 减去一个矩阵 128 倍的全一矩阵。

```
load('hall.mat');
image=double(hall_gray(108:115,100:107));
```

```
C=dct2(image-128);
C2=dct2(image)-dct2(128*ones(8,8));
```

打开工作区变量，二者结果相同

2.2 自编程实现 DCT2

```
function C = dct2_my(x)
%DCT2_MY 自己实现的二维DCT函数
N=size(x,1);
A=zeros(N)+[0:N-1]';
B=zeros(N);
B(1,:)=1:2:2*N-1;

D=sqrt(2/N)*cos(A*B*pi/2/N);
D(1,:)=sqrt(1/N);

C=D*double(x)*D';
end
```

第一次没有最后的 double 转换时，报错“错误使用 * MTIMES (*) 不完全支持整数类。至少一个参数必须为标量。”查阅官方文档的 MTIMES “If one of A or B is an integer class (int16, uint8, ⋯), then the other input must be a scalar. Operands with an integer data type cannot be complex.”像素矩阵的数据类型为 uint8，所以要转为 double。
用自己编写的函数，对同一个一个 8×8 矩阵操作。用 immse 算均方误差结果。

```
C_my=dct2_my(image-128);
err=immse(C,C_my)

err =

    2.5854e-27
```

可知编写正确。

2.3

2.4 DCT 系数操作对图像影响

通过阅读材料，了解到 DCT 系数左上角代表直流分量，右上方是横向变化强度，左下角是纵向变化强度，右下角是两个方向都变化的强度。常见的景观图片纹理变化缓慢，所以 DCT 系数左上方大，右下方小。
猜想:

1. 将右 4 列置 0，会使图像横向变化纹理减弱。
2. 将左 4 列置 0，会使整体亮度变暗，纵向变化纹理减弱。
3. 转置：根据
$$D^T C^T D = (D^T C D)^T = P^T$$
相当于把图像块转置。
4. 旋转 90°：将左上方较大的系数移到了左下方，增强了纵向变化纹理强度。
5. 旋转 180°：将左上方较大系数移到了右下方，增强了横纵向变化纹理强度

验证猜想

```
load('hall.mat');
image=double(hall_gray(108:115,100:107));
C=dct2(image-128);

right=C;
right(:,5:8)=0;
left=C;
left(:,1:4)=0;
tp=C';
r90=rot90(C,1);
r180=rot90(C,2);%rot90函数可对array进行旋转操作
```

将操作后的 DCT 系数作逆变换，得到**图 3**
可以观察到，与原图相比，右侧置 0 后图片横向变化缓和了。左侧置 0 使整体亮度减弱。转置后图片也跟着转置。转 90° 使纵向纹理增强，180° 横纵纹理都增强了。

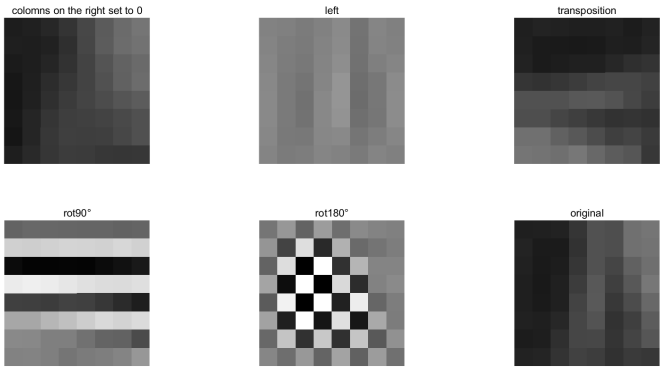

图 3: 对图像块进行不同操作后的结果

2.5 差分系统

绘制频率响应如**图 4**

```
1 freqz([-1 1],[1]);
```

观察到它是高通滤波器。先差分编码再熵编码，说明 DC 系数的高频分量更多。

2.6

利用误差计算 catgory 的方法：

$$Category = \lceil log_2|\hat{c}| \rceil$$

先对误差取绝对值，然后取以 2 为底的对数，最后向上取整。

2.7 ZIG-ZAG

matlab 有强大的矩阵运算功能。起初我尝试规避循环语句，使用矩阵运算，但没能尝试成功。最终采用循环语句实现了 zigzag 算法。思路如**图 5**：先扫红色，左上角奇数次，再扫蓝色，左上角偶数次，在扫右下角奇数次，右下角偶数次。代码操作上，右下角的扫描可以，将矩阵旋转 180°，扫完倒转向量。拼接向量，得到最终结果。代码如下。

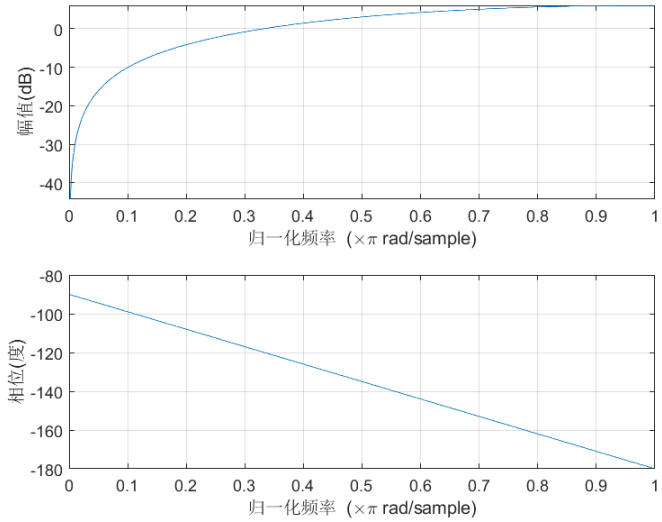

图 4: 差分系统频率响应

```
function y = zigzag(x)
%ZIGZAG 自己完成的zigzag扫描算法
N=length(x);
y1=zeros(1,N*(N+1)/2);
y2=zeros(1,(N-1)*N/2);
x_=rot90(x,2);
for i=1:2:N%左上角奇数次扫描
    idx=i*(i-1)/2;
    for j=1:1:i
        y1(idx+j)=x(i+1-j,j);
    end
end
for i=2:2:N%左上角偶数次扫描
    idx=i*(i-1)/2;
    for j=1:1:i
        y1(idx+j)=x(j,i+1-j);
    end
end
for i=1:2:N-1%右下角奇数次扫描
    idx=i*(i-1)/2;
    for j=1:1:i
        y2(idx+j)=x_(i+1-j,j);
    end
```

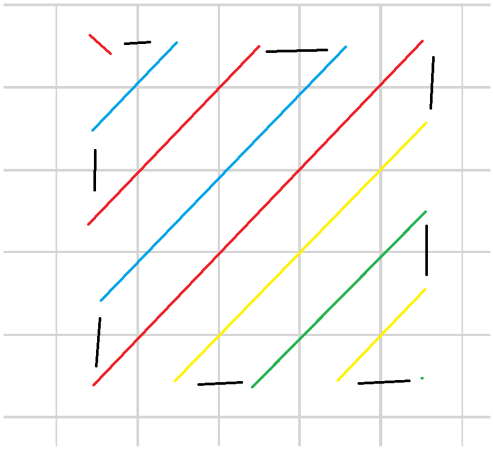

图 5: zigzag

```
24  end
25  for i=2:2:N-1%右下角偶数次扫描
26      idx=i*(i-1)/2;
27      for j=1:1:i
28          y2(idx+j)=x_(j,i+1-j);
29      end
30  end
31  y=[y1,rot90(y2,2)]';%拼接
32  end
```

对设计的 zigzag 函数测验

```
1  >> e=[1:1:16];
2  >> e=reshape(e,[4 4]);
3  >> zigzag(e)'
4
5  ans =
6
7        1     5     2     3     6     9    13    10     7     4     8   ...
            11    14    15    12    16
```

输出结果正确。

2.8 分块、DCT、量化

```
load('hall.mat');
load('JpegCoeff.mat');

W=ceil(size(hall_gray,2)/8);
H=ceil(size(hall_gray,1)/8);%/横纵块数。ceil为向上取整。

extend=padarray(hall_gray,[H*8-size(hall_gray,1),W*8-size(hall_gray,2)],...
    'replicate','post');
%如果原图像长宽不被8整除，需要补像素。注意参数设置。
extend=double(extend)-128;

c=zeros(8,8,H*W);%三元组，用于存储dct系数
for i=1:1:H
    for j=1:1:W
        t=extend(8*i-7:8*i,8*j-7:8*j);
        c(:,:,W*(i-1)+j)=round(dct2(t)./QTAB)%分块dct、量化;
    end
end

m=zeros(64,H*W);
for i=1:1:H*W
    m(:,i)=zigzag(c(:,:,i));
end
%每一列为每个块zigzag所得列矢量，第一行为DC系数。
```

2.9 JPEG 编码实现

2.9.1 DC 系数

```
cd_dif=zeros(1,H*W);
cd_dif(1)=m(1,1);
for i=2:1:H*W
    cd_dif(i)=m(1,i-1)-m(1,i);
end %以上为差分编码，查分结果存储在cd_dif中
DC_code=dc_code(cd_dif);
```

dc_code 函数内部:

```
function DC = dc_code(x)
%DC_CODE
load('JpegCoeff.mat');
DC=[];
Cat=uint8(floor(log2(abs(x)))+1)
%获取Category,如果没有uint8, 0会应对应到-inf
for i=1:1:length(x)
    category=Cat(i);
    L=DCTAB(category+1,1);%category对应huffman编码码长
    mag=buma1(x(i));%获取1-补码
    DC=[ DC DCTAB(category+1,2:L+1) mag ];
end
end
```

buma1 函数内部:

```
function y = buma1(x)
    if(x>0)
        y=double(dec2bin(x))-48;
    elseif(x<0)
        y=49-double(dec2bin(-x));
    else
        y=[];
    end
end
```

2.9.2 AC 系数

```
AC_code=[];%用于存储AC编码的数组。
for i=1:1:size(m,2)
    code=ac_code(m(2:end,i));%ac编码
    AC_code=[AC_code code];
end
```

ac_code 函数内容:

```
function AC = ac_code(x)
load('JpegCoeff.mat');
idx=1:1:length(x);
notzero=(x≠0);
idx=idx(notzero);%idx存储非0系数的下标
```

```
if length(idx)>0
    woc=idx(1);%第一个非0系数的run就是自己
    run=[woc diff(idx)]-1;%其余非0系数的run要对idx做差分-1
else run=[];
end
m=x(notzero);%m存储非0系数的值
Size=uint8(floor(log2(abs(m)))+1);%获取size
AC=[];
for i=1:1:length(idx)
    F_0=floor(run(i)/16);%ZRL个数
    rest_0=mod(run(i),16);%插完ZRL后还剩下多少个0
    L=ACTAB(rest_0*10+Size(i),3);%run/size的huffman编码长度
    r_s=[repmat([1 1 1 1 1 1 1 1 0 0 1],1,F_0),...
        ACTAB(rest_0*10+Size(i),4:L+3)];
    amp=buma1(m(i));%amplitude编码
    AC=[AC r_s amp];
end
AC=[AC 1 0 1 0];%接上EOF
end
```

最后将长宽、编码进行保存

```
height=H*8;width=W*8;
save('jpegcodes.mat','DC_code','AC_code','height','width');%保存结果
```

2.10 压缩比

输入所需 bit 数：长 × 宽 ×8。长 × 宽为像素数。每个像素取值范围 0-255，需要 8bit。
输出所需 bit 数：码流长度

```
load('jpegcodes.mat');
input_l=height*width*8;
output_l=length(DC_code)+length(AC_code);
rate=input_l/output_l
```

```
>> solution10

rate =

```

```
5       6.4247
```

计算得压缩比约为 6.4247

2.11 解码

2.11.1 熵解码

DC 解码所用函数

```
function DC = dc_decode(x,w,h)
%dc_decode
%    x为dc码流 w为横向块数 h为纵向块数
load('JpegCoeff.mat');
DC=[];
dc_todo=x;%用于存储尚未解码的部分
while ~isempty(dc_todo)
    for i=1:1:size(DCTAB,1)%在DCTAB中查找dc_todo最前面的huffman编码
        L=DCTAB(i,1);
        if(dc_todo(1:L)==DCTAB(i,2:L+1))
            category=i-1;
            dc_todo(1:L)=[];%将已经解码的category从dc_todo中删掉
            break;
        end
    end
    mag=dc_todo(1:category);
    dc_todo(1:category)=[];%将已经解码的magnitude从dc_todo中删除
    DC=[DC ibuma1(mag)];%由补码得到十进制数并拼接
end
end
```

在解码后再进行反差分。

```
for i=2:1:length(DC_decode)
    DC_decode(i)=DC_decode(i-1)-DC_decode(i);
end
```

AC 解码所用函数

```
function AC = ac_decode(x,w,h)
%AC_DECODE
%    x为ac码流 w为横向块数 h为纵向块数
```

```
load('JpegCoeff.mat');
ZRL=[1 1 1 1 1 1 1 1 0 0 1];
EOB=[1 0 1 0];

num=w*h;
AC=zeros(8*8-1,num);
ac_todo=x;
for j=1:1:num
    shit=[];
    while(¬isequal(ac_todo(1:4),EOB))%判断是否遇到EOB终止符
        while(ac_todo(1:11)==ZRL)%处理ZRL
            shit=[shit zeros(1,16)];
            ac_todo(1:11)=[];
        end

        for i=1:1:size(ACTAB,1)%在ACTAB中匹配run/size编码
            L=ACTAB(i,3);
            if(L≤length(ac_todo))%ac_todo较短后,
                %下面的if语句可能索引超出数组长度(L过大)
                if(ac_todo(1:L)==ACTAB(i,4:L+3))
                    run=ACTAB(i,1);
                    Size=ACTAB(i,2);
                    ac_todo(1:L)=[];
                    amp=ac_todo(1:Size);
                    ac_todo(1:Size)=[];
                    shit=[shit zeros(1,run) ibuma1(amp)];
                    break;
                end
            end
        end
    end
    ac_todo(1:4)=[];
    shit=[shit zeros(1,63-length(shit))];
    AC(:,j)=shit';
end
end
```

AC 解码的过程，由于存在判断 EOB、ZRL 的过程，较为复杂。主要有三个循环。

1. 判断当前未处理的编码开头是不是 EOB
2. 将当前编码开头的 ZRL 都处理掉
3. 将当前编码开头的 run/size 编码在 ACTAB 中进行匹配。

其中，2、3 在 1 的内部，处于并列结构。

2.11.2 反量化、DCT 逆变换、拼接

```
code=[DC_decode;AC_decode];
recover=uint8(zeros(height,width));
for i=1:1:H
    for j=1:1:W
        c=QTAB.*izigzag(code(:,W*(i-1)+j));%反量化
        p=idct2(c);%DCT逆变换
        p=p+128;
        recover(8*i-7:8*i,8*j-7:8*j)=p;%安置这一像素块
    end
end
```

2.11.3 编码效果

```
load('hall.mat');
image_initial=padarray(hall_gray,[H*8-size(hall_gray,1),W*8-size(hall_gray,2)],...
    'replicate','post');
mse=immse(image_initial,recover);
PSNR=10*log10(255^2/mse);
```

计算 PNSR 结果为 31.1874dB
显示两个图片 从观感上来说可以感受到，JPEG 的这种编码方式失真很小。如果仔细观察右图，可以发现较明显的 “分块”。该现象源自我们是分块处理的原图像。

2.12 QTAB/2

将 QTAB 变为原来 1/2, 再计算一次压缩比，结果为 4.4097，相比原来的 QTAB 有所减小。由于 QTAB 减小，得到的 DC、AC 系数更大，在编码时 magnitude、amplitude 更长，于是减小了压缩比。
计算 PSNR，结果为 34.2067dB，有所增大。QTAB 减小，就更接近于全一矩阵，接近于零失真，误差小，PSNR 大。

图 6: 处理前后的灰度大礼堂

2.13 美丽的雪花

改为处理 snow 变量。求得压缩比 = 3.6450;PSNR = 22.9244。与大礼堂相比，压缩比、PSNR 都更低。

雪花图像横纵纹理变化明显，主要为高频分量。观察量化后熵编码前的系数矩阵，可知 AC 系数中 0 较少，大多为需要编码的非零系数。这也使得 AC 编码长度达到了 40000 多，从而压缩比较低；而失真较为明显的原因，在于该图片高频分量占比较大，高频分量的量化系数都比较大，使得高频分量损失较多。

3 信息隐藏

3.1 空域

在空域隐藏中，我们将每个像素块的值末尾比特置为信息值。对新的像素矩阵，进行 jpeg 编解码，从新图像中提取出信息，检查信息失真程度。

```
1 load('hall.mat');
2 load('JpegCoeff.mat');
3 info_image=dec2bin(hall_gray);%将像素值转为二进制码
4 info=randi([0 1],[size(info_image,1),1]);%随机引入待传输信息
```

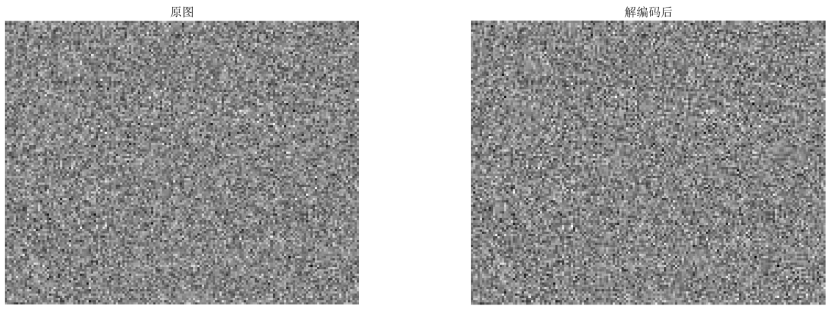

图 7: 雪花

```
info_image(:,8)=48+info;%将信息写入像素矩阵
info_image=bin2dec(info_image);%转回十进制
info_image=reshape(info_image,[size(hall_gray,1) size(hall_gray,2)]);
%变回原图像尺寸
%……报告中略去了jpeg解编码
get_info=dec2bin(recover);
get_info=get_info(:,8);
get_info=get_info-48;%从解码后的图像中获取得到的信息

imshow(recover);
immse(get_info,info)
```

JPEG 编解码部分没有写在报告里。此代码重点在于第一部分：二进制末位的代换、最终对信息提取准确率的评估。末位代换用到了 dec2bin 函数，信息提取准确率可以使用 immse。从 immse 的定义可知，其计算结果正好是获取信息后，错误的比特占总比特的比例。

```
for t=1:1:10
run('solu3_1.m');
rate=rate+wucha;
end
rate=rate/10

rate =
```

```
8
9       0.4985
```

如上，将该程序运行了十次，正确率接近一半，和概率意义下的结果一致，假如我瞎猜传过来的编码，正确率也差不多是 50%。根据信息论对信息的定义 “信息是减少不确定性的东西”，可知 jpeg 的解编码使得空域法失去了传达信息的功能。查看该方法输出的**图 8**，和原图差别较大，可见该方法隐蔽性较差。

图 8: 空域隐藏大礼堂结果

3.2 DCT 域信息隐藏技术

三种方法的共同点是在量化后的 DCT 系数中藏入信息，然后 jpeg 编解码。为了方便，把第二节 JPEG 中的一些步骤封装成函数，以便调用。

1. dct_lh.m 对像素矩阵做 dct、量化
2. code.m 对 zigzag 后的每个块做编码
3. decode.m 解码、反 zigzag、反量化、反 dct 等，获取图像

下面对各种方法中隐藏信息、获取信息部分的关键代码进行解释

3.2.1 方法一：代换全部 DCT 系数

信息隐藏函数

```
function y = info_hide1(x,c)
% x是待隐藏的信息 c是量化后的DCT系数
h=size(c,1);
w=size(c,2);
y=zeros(h,w);%用于存储加入隐藏信息的DCT系数，作为返回值
for i=1:1:h
    for j=1:1:w
        bin=buma1(c(i,j));
        if(isempty(bin))
            if(x(i,j)==1)
                bin=1;
            else bin=[];
            end
        else
            bin(end)=x(i,j);
        end
        y(i,j)=ibuma1(bin);
    end
end
```

由于使用的是 1-补码，0 在 magnitude/amplitude 编码中为空。所以代换最后一位时，0 必然变为 1、-1。为了提高压缩比，就要尽量让 0 多一些。所以“改换二进制末位”时，如果本来该 DCT 系数为 0，改换规则为

1. 信息为 1：系数变为 1
2. 信息为 0：系数还为 0，而不是-1

信息提取函数

```
function y = info_get1(c)
% 输入为量化后的DCT系数矩阵 输出y为信息
h=size(c,1);
w=size(c,2);
y=zeros(h,w);
for i=1:1:h
    for j=1:1:w
        bin=buma1(c(i,j));
        if(isempty(bin))
            y(i,j)=0;
        else
            y(i,j)=bin(end);
        end
    end
```

```
15 end
```

3.2.2 方法二：代换部分 DCT 系数

我选取的代换方法是，隔一个系数代换一次。代码与方法一类似，只需要在循环时将步长改为 2，同时信息的长度缩减为原来的 1/4。

3.2.3 方法三：zigzag 后追加

信息隐藏函数

```
function m= info_hide3(x,m)
% x为待隐藏的信息
% m为zigzag后的数组（每一列是某块的zigzag矢量）
for i=1:1:size(m,2)
    z=1:1:size(m,1);
    no0=m(:,i)≠0;
    z=z(no0);
    if(m(end,i)≠0)%如果最后一位非零，则将信息放在最后一位
        m(end,i)=x(i);
    else
        m(z(end)+1,i)=x(i);%否则追加在最后一个非零位后面
    end
end
```

方法三的信息隐藏发生在 zigzag 之后，而一二是在量化之后，所以方法三的输入和一二不同，输入的是 zigzag 后的数组。
信息提取函数

```
function info = info_get3(m)
info=zeros(1,size(m,2));
idx=m≠0;
for i=1:1:size(m,2)
    lie=m(idx(:,i),i);
    info(i)=lie(end);
end
```

用这三种方法隐藏信息，并计算

1. 错误率。使用 immse，注意第三个方法需要/4，因为 1、0 变为了 1、-1

2. 压缩比

3. PYNR

下面是三种方法在 solu3_2.m 中的完整代码

```
load('hall.mat');
W=ceil(size(hall_gray,2)/8);
H=ceil(size(hall_gray,1)/8);
error=zeros(1,3);
compress_rate=zeros(1,3);
PYNR=zeros(1,3);
c=dct_lh(hall_gray);%量化后矩阵
for k=1:1:10
info1=randi([0 1],H*8,W*8);
info2=randi([0 1],H*4,W*4);
info3=randi([0 1],1,H*W);

%method1
%信源隐藏信息、进行编码
y1=info_hide1(info1,c);
m1=zeros(64,H*W);
for i=1:1:H
    for j=1:1:W
        m1(:,(i-1)*W+j)=zigzag(y1(8*i-7:8*i,8*j-7:8*j));
    end
end

[DC_code_m1,AC_code_m1]=code(m1);
%信宿解码、寻找信息
ima1=decode(DC_code_m1,AC_code_m1,W,H);
c1=dct_lh(ima1);
y1_=info_get1(c1);

error(1)=error(1)+immse(y1_,info1);
compress_rate(1)=...
compress_rate(1)+120*168*8/(length(DC_code_m1)+length(AC_code_m1));
PYNR(1)=PYNR(1)+10*log10(255^2/immse(hall_gray,uint8(ima1)));

%method2
%信源隐藏信息、进行编码
y2=info_hide2(info2,c);
m2=zeros(64,H*W);
for i=1:1:H
    for j=1:1:W
        m2(:,(i-1)*W+j)=zigzag(y2(8*i-7:8*i,8*j-7:8*j));
```

```
41        end
42    end
43    [DC_code_m2,AC_code_m2]=code(m2);
44    %信宿解码、寻找信息
45    ima2=decode(DC_code_m2,AC_code_m2,W,H);
46    c2=dct_lh(ima2);
47    y2_=info_get2(c2);
48    error(2)=error(2)+immse(y2_,info2);
49    compress_rate(2)=...
50    compress_rate(2)+120*168*8/(length(DC_code_m2)+length(AC_code_m2));
51    PYNR(2)=PYNR(2)+10*log10(255^2/immse(hall_gray,uint8(ima2)));
52
53    %method3
54    %信源隐藏信息、进行编码
55    idx=info3==0;
56    info3(idx)=-1;
57    m3=zeros(64,H*W);
58    for i=1:1:H
59        for j=1:1:W
60            m3(:,(i-1)*W+j)=zigzag(c(8*i-7:8*i,8*j-7:8*j));
61        end
62    end
63
64    m3=info_hide3(info3,m3);
65    [DC_code_m3,AC_code_m3]=code(m3);
66    %信宿收到编码、解码、寻找信息
67    ima3=decode(DC_code_m3,AC_code_m3,W,H);
68    c3=dct_lh(ima3);
69    getm3_zigzag=zeros(64,H*W);
70    for i=1:1:H
71        for j=1:1:W
72            getm3_zigzag(:,(i-1)*W+j)=zigzag(c3(8*i-7:8*i,8*j-7:8*j));
73        end
74    end
75    compress_rate(3)=...
76    compress_rate(3)+120*168*8/(length(DC_code_m3)+length(AC_code_m3));
77    PYNR(3)=PYNR(3)+10*log10(255^2/immse(hall_gray,uint8(ima3)));
78    y3_=info_get3(getm3_zigzag);
79    error(3)=error(3)+immse(y3_,info3)/4;
80    end
81
82    error=error/10;
83    PYNR=PYNR/10;
84    compress_rate=compress_rate/10;
```

循环十次，减小偶然误差。最终输出 error、PYNR、compress_rate，查看

图片

```
error =

     0     0     0
PYNR =

   15.2760   20.8864   28.9548
compress_rate =

    2.8691    4.2898    6.1916
```

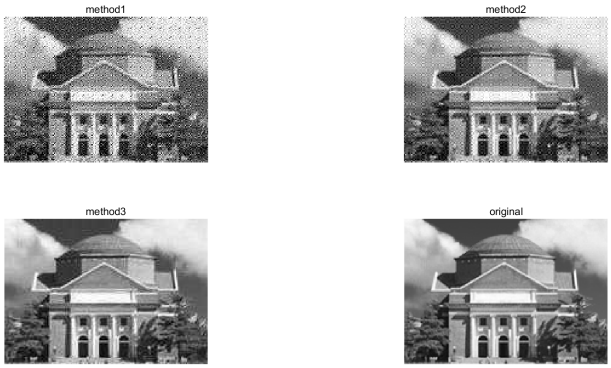

图 9: 三种方法

1. 三种方法都可以无损获得信息，证明其抗 JPEG 编码
2. 第三种方法压缩比最高，第一种最低。原因在于第三种加入的非零系数最少，第一种最多。
3. 第三种方法 PYNR 最高，观感上最贴近原图，其隐秘性最高，质量最好。第一种隐秘性最差，质量最差。
4. 第二种方法处于中间质量、压缩比，推测如果将信息藏在量化矩阵比较小的数的位置，可以让图片质量更高，隐秘性更高。

4 人脸检测

4.1 训练

4.1.1

不需要调整成相同尺寸。向量 $u(R)$ 经过了对总像素数的归一化，是一个概率密度函数，满足

$$\sum_{i=0}^{N-1} f_i(R) = 1$$

4.1.2

```
clear;
file_path = 'Faces\';
img_path_list = ...
    dir(strcat(file_path,'*.bmp'));%在某文件夹中获取特定扩展名文件的方法
len=length(img_path_list);
L_choice=[3 4 5];
V=zeros(3,2^(3*5));
for k=1:1:3
    L=L_choice(k);
    u=zeros(1,2^(3*L));
    for i=1:1:len
        name=img_path_list(i).name;
        I=double(imread(strcat(file_path,name)));
        u=u+u_R(I,L);%u_R用于计算颜色特征矢量
    end
    V(k,1:2^(3*L))=u/len;
    subplot(3,1,k);
    plot(u/len);
end
save('V.mat','V');%将训练好的模型保存，以便后续使用
```

其中的 u_R 函数, 其中一步我在写的时候进行了改进。

```
function u = u_R(I,L)
I=double(I);
[h,w,~]=size(I);
u=zeros(1,2^(3*L));
RGB=floor(I/(2^(8-L)));
```

```
p=RGB(:,:,1)*2^(2*L)+...
  RGB(:,:,2)*2^(L)+...
  RGB(:,:,3);
for i=1:1:h*w%%%%%%%
    u(p(i)+1)=u(p(i)+1)+1;
end
u=u/(h*w);
end
```

上面代码添加了很多% 的那一行，用于得到每种颜色所占据的概率密度。这样计算的复杂度是

$$O(H * W) \tag{法 1}$$

最初我的写法是对每种颜色，计算其在 p 中出现的次数。

```
for i=1:1:2^(3*L)
    u(i)=sum(p==i-1,'all');
end
```

这样的复杂度是

$$O(2^L * H * W) \tag{法 2}$$

法 2 对颜色种类循环，每次循环内部的 sum 相当于都对矩阵逐元素访问了一次，即每次循环都包含了法 1 的 H*W 次循环。法 2 的运行时间足足是法 1 的 $2^{(3*L)}$ 倍。我在后面识别人脸时发现 L=5 的情况耗时过长，返回来改进前面的代码，发现了这一改进之处。

关系

所得 V 长度不同，依此翻 2^3 倍。但是总体的概率密度分布形状应该类似，只是 L 取的越小，峰越少，把近似的颜色归到了一类里。类比《信号与系统》中，在学习有限宽信号和周期信号的傅里叶变换时，后者是前者的频谱不断聚集到谐波频点上，形成冲激，这里更大的 L 的概率密度，就好比周期信号的频谱，峰集中在了点上。如**图 10**

4.2 设计人脸检测算法

我所设计的算法思路为

1. 对一个窗格进行是否为人脸的判断，并以一定步长移动该窗格。

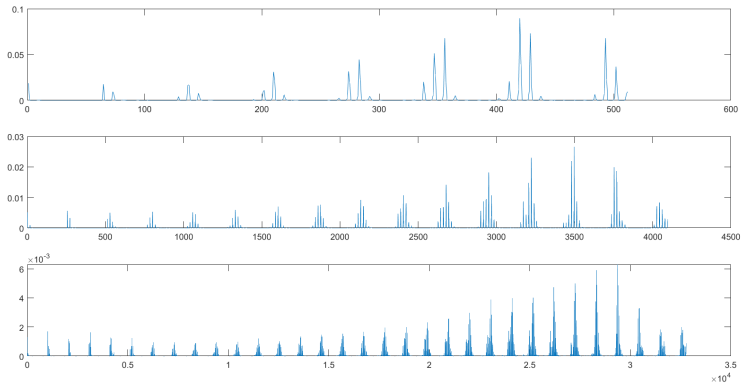

图 10: 概率分布列

2. 将所有为人脸的部分用 bwlabel 找到连通分量，得到每个连通分量的行列边界，画红线

```
load('V.mat');%已经训练好的人脸特征矢量（L=3、4、5）
I=imread('');
[H,W,¬]=size(I);
window=;%每次检测的窗格大小
step=;%窗格移动的步长

for k=3:1:5
    face=false(H,W);
    for i=1:step:H+1-window
        for j=1:step:W+1-window
            img=I(i:i+window-1,j:j+window-1,:);
            u=u_R(img,k);%提取特征
            err=dis(u,V(k-2,1:2^(3*k)));%计算距离
            if(err<  )%取阈值
                face(i:i+window-1,j:j+window-1)=true;
            end
        end
    end

[L,n]=bwlabel(face);%连通分量标签和个数
face=false(H,W);%

for i=1:1:n
    [r,c]=find(L==i);
```

```
25        min_row=min(r);
26        max_row=max(r);
27        min_col=min(c);
28        max_col=max(c);
29      face(min_row,min_col:max_col)=true;
30      face(max_row,min_col:max_col)=true;
31      face(min_row:max_row,min_col)=true;
32      face(min_row:max_row,max_col)=true;%边界设为1，一会儿画线
33  end
34
35  face=cat(3,face,false(H,W),false(H,W));
36  P=I;
37  P(face)=255;
38   figure;
39  imshow(P);
40  end
```

剩下的任务是如何取窗格、步长大小以及阈值。

4.2.1 窗格、步长

起初我想将窗格、步长与图片尺寸联系起来，经过思考我认为这并不合适。可以猜测，合适的参数和人脸像素大小有关，而和图片的尺寸关系不大，因为一张图片中有多少人脸是不确定的。我尝试将 window 设为 20，step 设为 3，对 Beatles 的一张照片进行测试。阈值在这里先设为 0.7。结果如**图 11**

可以看到，L=3、4 的情况检测效果较好，而 L=5 的情况效果较差。推

图 11: window=20 step=3

测是由于 L=5 的情况更加精细，同样的一个像素块，如果越接近人脸，则

误差越小，如果越不像人脸，误差越大。对 L=5 的情况，如果 window 过小，就可能在一些地方误判以为不是人脸，无法形成较大的人脸连通分量。以此为启示，我们尝试将 window 的大小和 L 联系起来，将 window 设置成 20+L, 这样 L 越大, 窗格越大。再运行一次程序，结果如**图 12**

图 12: window=20+L step=3

检测结果好了一些，L=5 的图片也检测出了三位队员的人脸，但还有一位没能成功。这时经过测试，最好不要再调节 window，因为这有可能使得相邻的人脸被检测到一起。于是我们尝试改变阈值。

4.2.2 阈值

现在尝试改变阈值。可以猜测, L 越大, 对人脸的要求越严格, 阈值也应该越大。所以我尝试将阈值也与 L 相关联起来，于是取阈值 =0.6+L*0.03。运行一次，结果如**图 13**

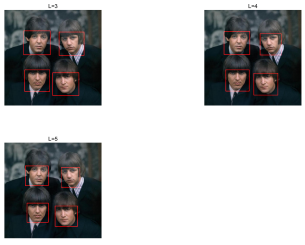

图 13: window=20+L step=3 $\varepsilon = 0.6 + L * 0.03$

这次的检测结果很好，三种 L 的结果很统一。如果三种 L 都取一样的窗格大小和阈值，则 L 越大，越容易漏掉人脸；L 越小，越容易把不是人脸的东

西错判成人脸。

4.3

将上面的人脸检测封装成函数 checkface

```
I=imread('a.jpeg');

%旋转90
I1=imrotate(I,-90);
checkface(I1);
%调整尺寸
I2=imresize(I,[size(I,1) 2*size(I,2)]);
checkface(I2);
%调整颜色
I3=imadjust(I,[.2 .3 0; .6 .7 1],[]);
checkface(I3);
```

检测结果如**图 14 15 16**

图 14: 顺时针 90°

旋转、压缩都依然可以成功检测，但改变颜色后检测效果并不好。改变颜色的参数来自 matlab 官方文档，可以看出这一参数使得图片对比度升高较多，肤色已经偏离了正常人的肤色。我们使用的算法也是根据肤色检测，训练的模型如果针对黄种人，那用黑种人就很难检测成功。但是我们也不可能把所有人种都作为训练样本，训练出一个统一的模型，黑色和白色本身就是两个极端。可见我们在作业中所使用的算法本身有很大局限性。

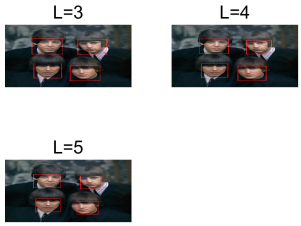

图 15: 宽度 *2

图 16: 调整颜色

4.4

样本应该尽可能去掉头发的影响, 所使用的训练集应尽可能不包含头发。最好给白种人、黄种人、黑种人各训练一个模型。

5 总结与收获

1. 对 matlab 的矩阵运算更加熟悉。在做此次作业之前，我经常使用循环，写的代码不够简洁，但在课程中听到老师说 “matlab 是 matrix lab，其矩阵运算功能十分强大”，而且了解到最早做深度学习的人很多都用 matlab，也是因为其对矩阵运算封装质量高，所以我在做作业过程中尽量思考是否能使用矩阵运算, 而规避循环的使用，我想这样才算理解 matlab 的精髓。

2. 初步预习了 DFT、DCT 等更重要的变换。作业文档以及课本都显示了 DFT 的重要性，其也是 FFT 的基础。《信号与系统》课上讲的知

识还只是信号处理的基础中的基础。

3. 使用 latex 排版。matlab 课程的两次作业报告我都使用 latex 进行排版，这是前所未有的。以前我只是简单学习了基本的操作，但这次我完整排版出了自己的 tex 文档。

4. 老师留作业给予了充足的提示、背景知识，让我体会了从学习到应用，再尝试自我探索突破的过程。在电子系读了两年，大部分作业给的提示较少，我常常参考往年学长的作业，这让我养成了很不好的习惯，我一直体会到这不利于终身学习，也不利于今后的科研道路。但这次作业老师给铺好了很好的台阶，我得以踩在上面，基本独立完成。在此感谢谷老师，虽然您称这是一门很轻松的课，但我这一个假期有很大的收获。