MATLAB 综合实验之图像处理*

聂浩　无 31　2013011280

2015 年 8 月 30 日

1　基础知识

(1)　MATLAB 提供了图像处理工具箱，在命令窗口输入 help images 可查看该工具箱内的所有函数。请阅读并大致了解这些函数的基本功能。

感觉较为常用的函数有:

- image 建立图片对象, 在坐标轴中绘制, 颜色取决于现在的颜色设置
- imshow 显示图片 (按照原来图片的大小)
- imread 读取图片文件
- imwrite 写图片文件
- imabsdiff 比较两张图片的差异
- checkerboard 生成棋盘

(2)　利用 MATLAB 提供的 Image file I/O 函数分别完成以下处理：

(a)　以测试图像的中心为圆心，图像的长和宽中较小值的一半为半径画一个红颜色的圆：

因为这个图像非常小, 所以直接用循环就进行了处理, 没有进行太多优化. 图像如图1, 代码见下一问。

(b)　将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即黑色，“白”则意味着保留原图。用一种看图软件浏览上述两个图，看是否达到了目标。

图像如图2

代码如下 (a3_1.m):

*所有的.m 文件均采用 utf8 编码，windows 版 matlab 中打开可能会出现中文乱码的情况，请用其它编辑器打开

图 1: 绘制红色圆

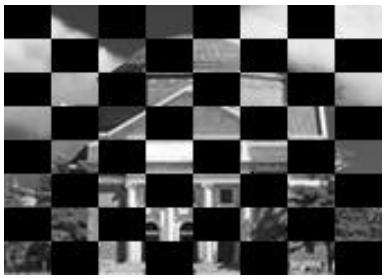

图 2: 绘制黑白格

```
clear;close all;clc;
load('hall.mat');
[m,n,q]=size(hall_color);
r=0.5*min(m,n);
o=0.5*[m n];
circle=hall_color;
board=hall_gray;
step_m=round(m/8);
step_n=round(n/8);
for i=1:m
    for j=1:n
        if(sum(([i j]-o).^2)<=sum(r.^2))
            circle(i,j,1)=255;
            circle(i,j,2)=0;
            circle(i,j,3)=0;
        end
        if(mod(fix((i-1)/step_m),2)==mod(fix((j-1)/step_n),2))
            board(i,j)=0;
        end
    end
end
```

```
imshow(circle);
figure
imshow(board);
imwrite(circle,'circle.bmp');
imwrite(board,'board.bmp');
```

2 图像压缩编码

(1) 图像的预处理是将每个像素灰度值减去 128，这个步骤是否可以在变换域进行？请在测试图像中截取一块验证你的结论。

根据二维 DCT 变换的定义式 $C = DPD^T$, 这是一个线性变换，所以变换前后处理是一致的。

在这里截取了 hall_gray(61:68,81:88), 两种处理次序后的绝对值差在 10^{-12} 数量级，可以认为这只是计算误差，故两者等价。

代码如下 (a3_2_1.m):

```
clc;clear;close all;
load('hall.mat');
in=hall_gray(61:68,81:88);
s1=dct2(in-128);
s2=dct2(in)-dct2(128*ones(size(in)));
e=imabsdiff(s1,s2)
```

(2) 请编程实现二维 DCT，并和 MATLAB 自带的库函数 dct2 比较是否一致。

我直接使用计算 D 矩阵然后相乘的方法进行计算，其计算复杂度为 $O(n^2)$。

系统的 DCT2 函数的调用了两次 DCT 函数，而 DCT 函数则使用了 FFT，因此其计算复杂度为 $O(n\log(n))$。

两者的误差在 10^{-9} 数量级，可以认为这只是计算误差。

在数据较大时，如图3, 系统 DCT2 函数快于我的 my_DCT2 函数。[1]

my_dct2 的代码如下：

```
function [C]=my_dct2(P)
[m n]=size(P);
D1=sqrt(2/m)*[sqrt(0.5)*ones(1,m);cos(kron((1:2:(2*m-1)),(1:m-1)')*pi
    /2/m)];
if m==n
    D2=D1;
else
```

[1]因为计算机差异，具体值可能不同

Profile Summary

Generated 22-Aug-2015 15:30:37 using real time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
my_dct2	1	0.307 s	0.294 s	
dct2	1	0.151 s	0.016 s	
images/private/dct	2	0.135 s	0.135 s	
kron	2	0.013 s	0.013 s	

图 3: 将 hall_gray 重复 100 次后两种 DCT 变换所消耗的时间

```
    D2=sqrt(2/n)*[sqrt(0.5)*ones(1,n);cos(kron((1:2:(2*n-1)),(1:n-1)
        ')*pi/2/n)];
end
C=D1*double(P)*D2';
return;
```

测试代码如下（a3_2_2.m）：

```
clc;clear;close all;
load('hall.mat');
in=repmat(hall_gray,10,10)-128;
profile on;
s1=dct2(in);
s2=my_dct2(in);
profile viewer;
e=imabsdiff(s1,s2);
max(max(e))
```

(3) 如果将 DCT 系数矩阵中右侧四列的系数全部置零，逆变换后的图像会发生什么变化？选取一块图验证你的结论。如果左侧的四列置零呢？

如图4，右侧四列都置零，逆变换后的图像变化不大，因为人眼对高频分量不敏感。当左侧四列都置零，逆变换图片变暗。因为很多低频分量，包括基频被滤掉，导致各点值偏小而发暗。

代码如下（a3_2_3.m）

```
clear;clc;close all;
load('hall.mat');
in=hall_gray;
```

图 4: 原图、左四列与右四列分别清零

```
subplot(1,3,1);
imshow(in);
title('Origin');
C=dct2(in);
[m,n]=size(in);
%左侧
C_l=C;C_l(:,(1:4))=0;
%右侧
C_r=C;C_r(:,(n-3:n))=0;
subplot(1,3,2)
imshow(uint8(idct2(C_l)));
title('Left to zero');
subplot(1,3,3);
imshow(uint8(idct2(C_r)));
title('Right to zero');
```

(4) 若对 DCT 系数分别做转置、旋转 90 度和旋转 180 度操作 (rot90)，逆变换后恢复的图像有何变化？选取一块图验证你的结论。

如图5，转置使得图像沿左上至右下的对角线翻转镜像；旋转 90° 使图像在之前的基础上还出现了黑白条纹；旋转 180° 后图像没有旋转，但是出现了黑白小斑点。

这是因为转置并未改变高低频信息，但两轴被交换，故出现翻转；旋转使得高频和低频分量的信息混淆，故高频相对之前被放大了——旋转 90° 只有一个方向的高频较明显，故为条纹；旋转 180° 则增强了两个方向的高频分量，故为斑点。

代码如下（a3_2_4.m）：

```
clear;clc;close all;
load('hall.mat');
in=hall_gray;
subplot(2,2,1);
imshow(in);
```

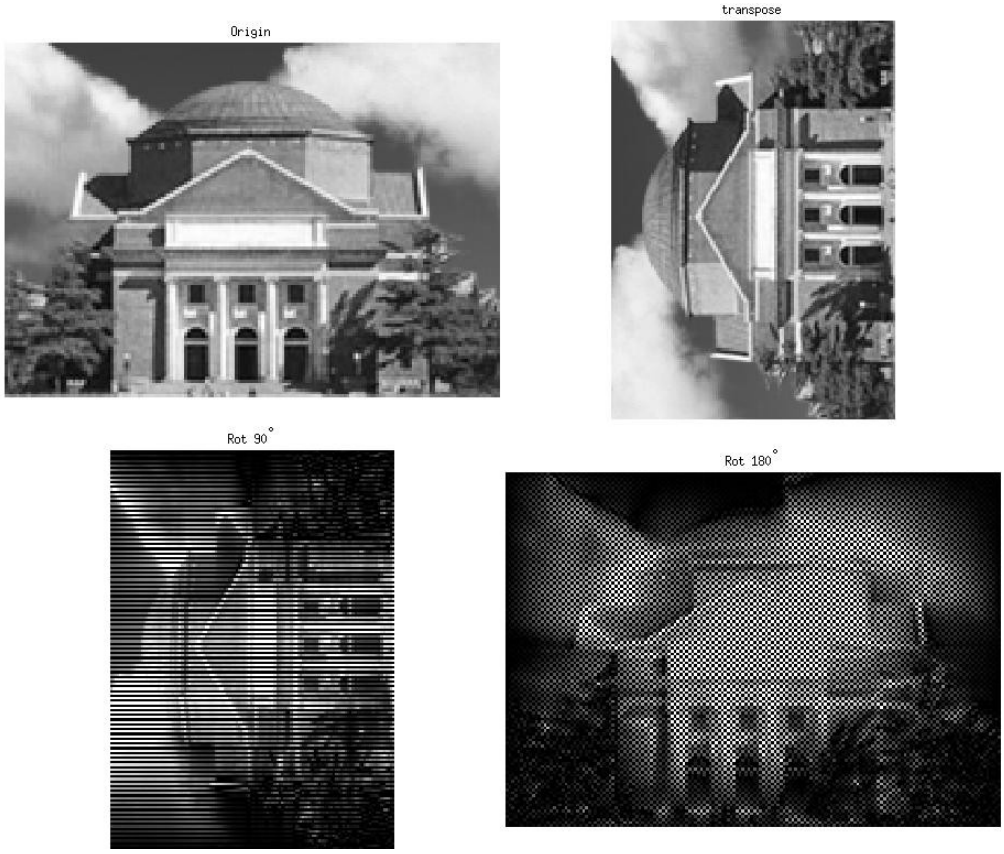

图 5: 左四列与右四列分别清零

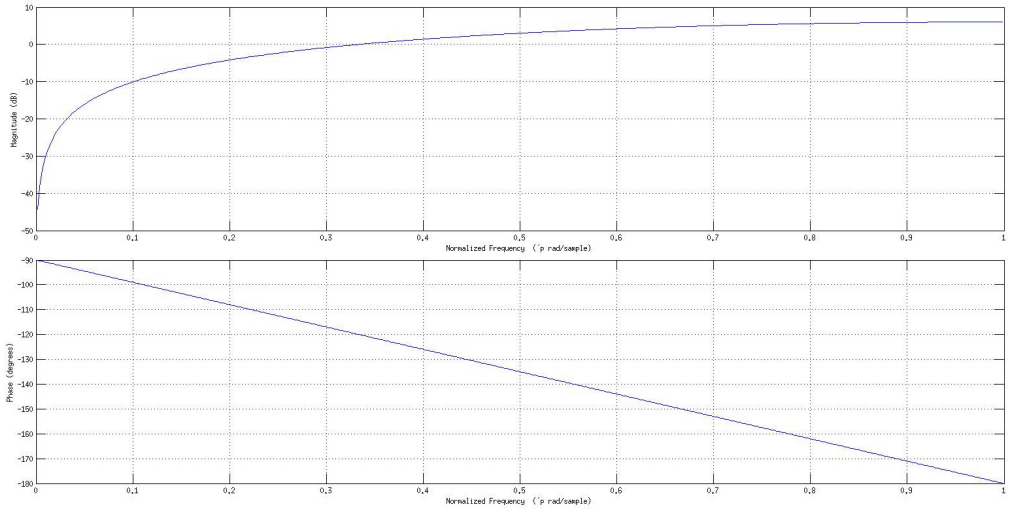

图 6: 差分的频率响应

```
title('Origin');
C=dct2(in);
[m,n]=size(in);
C_tran=C';
C_90=rot90(C);
C_180=rot90(C_90);
subplot(2,2,2)
imshow(uint8(idct2(C_tran)));
title('transpose');
subplot(2,2,3)
imshow(uint8(idct2(C_90)));
title('Rot 90^{\circ}');
subplot(2,2,4);
imshow(uint8(idct2(C_180)));
title('Rot 180^{\circ}');
```

(5) 如果认为差分编码是一个系统，请绘出这个系统的频率响应，说明它是一个 _____(低通、高通、带通、带阻) 滤波器。DC 系数先进行差分编码再进行熵编码，说明 DC 系数的 _____ 频率分量更多。

差分编码的差分方程为 $y(n) = x(n-1) - x(n)$, 其系统函数为

$$H(z) = \frac{1}{z} - 1$$

仿真得到图6, 这是一个高通滤波器。说明 DC 系数的低频分量更多，这样处理可以压缩低频分量。代码如下:a3_2_5.m

```
clc;clear;close all;
b=[-1 1];
a=1;
freqz(b,a);
```

(6) DC 预测误差的取值和 Category 值有何关系? 如何利用预测误差计算出其 Category?

否则 $Category = ceil(log2(abs(\hat{c}_D) + 1))$, 包括 $\hat{c} = 0$ 的情况。

(7) 你知道哪些实现 Zig-Zag 扫描的方法? 请利用 MATLAB 的强大功能设计一种最佳方法。

按照最原始的思路，采用循环的方式，将元素依次放入一数组中，然后利用逻辑判断决定接下来去哪个元素。但是这样速度显然很低。更为直接的思路是利用查表法，因为该图像大小为 8×8, 直接构造一个查表矩阵是最好的，同时，通过查询[2], 将矩阵转换到一维处理是更为简便的方式，不过其 zigzag 的顺序和试验要求有一定出入，简单修改即可。

代码如下 (zigzag.m):

```
function [a]=zigzag(A)
zigtag=[1 ,2 ,9 ,17,10,3 ,4 ,11,18,...
        25,33,26,19,12,5 ,6 ,13,20,...
        27,34,41,49,42,35,28,21,14,...
        7 ,8 ,15,22,29,36,43,50,57,...
        58,51,44,37,30,23,16,24,31,...
        38,45,52,59,60,53,46,39,32,...
        40,47,54,61,62,55,48,56,63,...
        64];
%A变成64x1的矢量
aa = reshape(A',64,1);
a=aa(zigtag);
return;
```

测试该函数的代码 (a3_2_7.m)

```
clc;clear;clear all;
A=magic(8)
zigzag(A)
```

[2]参照http://blog.sina.com.cn/s/blog_54e2ed7b0100mmb7.html

(8) 对测试图像分块 DCT 和量化, 将量化后的系数写成矩阵的形式, 其中每一列为一个块的 DCT 系数 Zig-Zag 扫描后形成的列矢量, 第一行为各个块的 DC 系数。

代码如下（a3_2_8.m）

```
clc;clear;close all;
load('hall.mat');
load('JpegCoeff.mat');
[m n]=size(hall_gray);
%把长宽扩至8的倍数;
M=ceil(m/8);N=ceil(n/8);
hall_gray=double(hall_gray);
if (M*8~=m)
hall_gray=[hall_gray hall_gray(:,n)*ones(1,(M*8-m))];
end
if (N*8~=n)
hall_gray=[hall_gray;ones((N*8-n),1)*hall_gray(m,:)];
end

R=zeros(64,M*N);
hall_gray=hall_gray-128;
for i=1:M
    for j=1:N
        R(:,i*N+j-N)=zigzag(dct2(hall_gray(i*8-7:i*8,j*8-7:j*8)));
    end
end
```

(9) 请实现本章介绍的 JPEG 编码 (不包括写 JFIF 文件), 输出为 DC 系数的码流、AC 系数的码流、图像高度和图像宽度, 将这四个变量写入 jpegcodes.mat 文件。

这里二进制转换时将字符串减去 48（0 的 ascii 码），从而讲 dec2bin 的字符串转化成了数组。

代码如下（a3_2_9.m）：

```
clc;clear;close all;
load('hall.mat');
load('JpegCoeff.mat');
[m,n]=size(hall_gray);
M=ceil(m/8);N=ceil(n/8);
%转化为double以进行计算
hall_gray=double(hall_gray);
%长宽扩至8的倍数
```

```
if (M*8~=m)
    hall_gray=[hall_gray hall_gray(:,n)*ones(1,(M*8-m))];
end
if (N*8~=n)
    hall_gray=[hall_gray;ones((N*8-n),1)*hall_gray(m,:)];
end
m=8*M;
n=8*N;
%C存储DCT基数，R存储zig-tag变换后的值
hall_gray=hall_gray-128;
C=hall_gray;
R=zeros(64,M*N);
for i=1:M
    for j=1:N
        %量化
        C(i*8-7:i*8,j*8-7:j*8)=round(dct2(hall_gray(i*8-7:i*8,j*8-7:j
            *8))./QTAB);
        %zig-zag
        R(:,i*N-N+j)=zigzag(C(i*8-7:i*8,j*8-7:j*8));
    end
end
%DC部分
%差分编码
ERR_DC=[2*R(1,1) R(1,1:N*M-1)]-R(1,:);
%DC部分的编码
DCstream=logical([]);
Category_DC=ceil(log2(abs(ERR_DC)+1));
%AC部分
ACstream=logical([]);
Size_AC=ceil(log2(abs(R)+1));
zero16=[1 1 1 1 1 1 1 1 0 0 1];
%编码
for i=1:M*N
    DCstream=[DCstream DCTAB(Category_DC(i)+1,2:1+DCTAB(Category_DC(i
        )+1,1))];
    if ERR_DC(i)>=0;%减48是因为0的asscii码
        DCstream=[DCstream dec2bin(ERR_DC(i))-48];
    else%一补码
        DCstream=[DCstream ~(dec2bin(-ERR_DC(i))-48)];
```

```
    end

    AC_NONE_ZERO=[1;1+find(R(2:64,i))];
    if length(AC_NONE_ZERO)~=1
        for k=2:length(AC_NONE_ZERO)
            count0=AC_NONE_ZERO(k)-AC_NONE_ZERO(k-1)-1;
            while count0>15
                ACstream=[ACstream zero16];
                count0=count0-16;
            end
            ACstream=[ACstream ACTAB( (count0*10+Size_AC(
                AC_NONE_ZERO(k),i)),...
            4:(3+ACTAB(count0*10+Size_AC(AC_NONE_ZERO(k),i),3)) )];
            if R(AC_NONE_ZERO(k),i)>=0;%减48是因为0的asscii码
                ACstream=[ACstream dec2bin(abs( R(AC_NONE_ZERO(k),i)
                    ))-48];
            else%一补码
                ACstream=[ACstream ~(dec2bin(abs( R(AC_NONE_ZERO(k),i
                    ) ))-48)];
            end
        end
    end
    ACstream=[ACstream 1 0 1 0];
end
save jpegcodes.mat DCstream ACstream m n
```

(10) 计算压缩比 (输入文件长度/输出码流长度), 注意转换为相同进制。

压缩比为

$$\frac{8*m*n}{length(DCstream)+length(ACstream)} = \frac{8\times 120\times 168}{2054+23072} = 6.4188$$

(11) 请实现本章介绍的 JPEG 解码, 输入是你生成的 jpegcodes.mat 文件。分别用客观 (PSNR) 和主观方式评价编解码效果如何。

生成的图像如图7，感觉已经很像原图了。计算得到 PSNR=34.89，根据检查，haffman 编码解无损（R 和 source 一致）。

代码如下 (a3_2_10.m):

图 7: 原图与编码解码后的图像

```
clear;clc;close all;
load('jpegcodes');
load('hall');
load('JpegCoeff');

%判断DC
i=1;
DC=[];
%解码至残差
while(i<=length(DCstream))
    for ca=1:12
        try%可能存在DCstream不够长的情况
            if(DCstream(i:i+DCTAB(ca,1)-1)==DCTAB(ca,2:1+DCTAB(ca,1)));
                i=i+DCTAB(ca,1);
                if(ca==1)
                    DC=[DC 0];
                    i=i+1;
                    break
                elseif(DCstream(i)==1)
                    %此处为ca(Category的序号)与数字位数的对应关系
                    DC=[DC bin2dec(num2str(DCstream(i:i+ca-2)))];
                else
                    DC=[DC -bin2dec(num2str(~DCstream(i:i+ca-2)))];
                end
                i=i+ca-1;
                break;
            end
```

```
28            end
29        end
30    end
31    %恢复至量化
32    for  i=2:length(DC)
33        DC(i)=DC(i-1)-DC(i);
34    end
35    %判断AC
36    i=1;count=1;
37    AC=zeros(63,length(DC));tmp=[];
38    %解码至zig-zag前
39    while(i<=length(ACstream))
40            %判断EOB
41            if(ACstream(i:i+3)==[1  0  1  0])
42                i=i+4;
43                AC(:,count)=[tmp;zeros(63-length(tmp),1)];
44                tmp=[];
45                count=count+1;
46                continue;
47            end
48            %判断ZRL
49            if((i+10)<=length(ACstream))
50                if(ACstream(i:i+10)==[1  1  1  1  1  1  1  1  0  0  1])
51                    i=i+11;
52                    tmp=[tmp;zeros(16,1)];
53                    continue;
54                end
55            end
56            for  j=1:160
57                %避免溢出
58                try
59                    if(ACstream(i:i+ACTAB(j,3)-1)==ACTAB(j,4:3+ACTAB(j,3)
                      ));
60                        i=i+ACTAB(j,3);
61                        Run=fix((j-1)/10);
62                        if(ACstream(i)==1)
63                            %此处为run/size与数字位数的对应关系
64                            tmp=[tmp;zeros(Run,1);bin2dec(num2str(
                                  ACstream(i:i+j-10*Run-1)))];
```

```
                    else
                        tmp=[tmp;zeros(Run,1);-bin2dec(num2str(~ACstream(
                            i:i+j-10*Run-1)))];
                    end
                    i=i+j-10*Run;
                    break;
                end
            end
        end
end
%反zigzag
source=[DC;AC];
%C为最终数据
pic=zeros(m,n);
M=ceil(m/8);N=ceil(n/8);
for i=1:M
    for j=1:N
        %量化
        pic(i*8-7:i*8,j*8-7:j*8)=izigzag(source(:,i*N-N+j));
        pic(i*8-7:i*8,j*8-7:j*8)=idct2(pic(i*8-7:i*8,j*8-7:j*8).*QTAB
            )+128;
        %zig-zag
    end
end
pic=uint8(pic);
subplot(1,2,1)
imshow(hall_gray);
title('origin');
subplot(1,2,2);
imshow(pic);
title('decode');

MSR=1/(m*n)*sum(sum((pic-hall_gray).^2));
PSNR=10*log10(255^2/MSR)
```

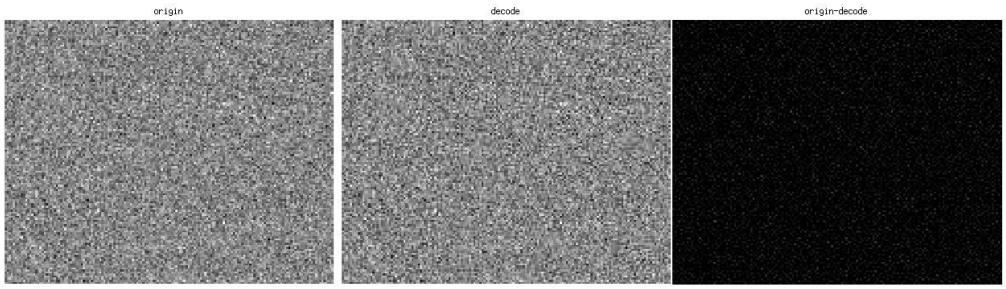

图 8: 雪花原图、编码后的图像和两者的差

图 9: 空域隐藏

(12) 将量化步长减小为原来的一半, 重故编解码。同标准量化步长的情况比较压缩比和图像质量。

使 QTAB=QTAB./2 即可。肉眼难以看到变化，压缩比为 4.4081，PSNR 为 37.32。即图片效果更好了，但压缩比也小了。

(13) 看电视时偶尔能看到美丽的雪花图像 (见 snow.mat), 请对其编解码。和测试图像的压缩比和图像质量进行比较, 并解释比较结果。

如图8, 把 hall_gray 换成 snow 即可，得到压缩比为 3.6407，PSNR=29.5614。两者看起来很像，但编码解码后的图看起来颗粒要大一些,PSNR 也小。这是因为编码过程中滤去了高频分量，而雪花图像中有很强的高频分量。

3 信息隐藏

(1) 实现本章介绍的空域隐藏方法和提取方法。验证其抗 JPEG 编码能力。

[3]

原始信息为传统的 “A quick brown fox jump over the lazy dog。” 通过图9可以看出隐藏很好，同时也能很好的解密信息。但是 jpeg 编码解码后再解密就变成了乱码[4]。

[3]只处理了 8 bit, 也就是只有英文字符，虽然增加每个字符的 bit 数可以加密更多的字符, 但这里涉及字符编码的很多知识, 在此不再进行讨论

[4]乱码形式也和字符编码有关，linux 下和 win 下的乱码不一样，这里就不附了

代码如下 (a3_3_1.m)

```
clear;close all;clc;
load('hall');
%加密,这里只处理了8bit，所以只能加密英文字母
message='A quick brown fox jump over the lazy dog';
ms=reshape((dec2bin(message,8)-48)',1,8*length(message));
%以0000000为结束符，直接从第一个像素开始加
ms=uint8([ms 0 0 0 0 0 0 0 0]);
[m,n]=size(hall_gray);
pic=reshape(hall_gray,1,m*n);
%除以2，fix再乘2等于把最低bit清零，然后加上需要加密的信息
pic(1:length(ms))=fix(pic(1:length(ms))/2)*2+ms;
pic=reshape(pic,m,n);
imshow(pic);

%解密
to_proc=mod(reshape(pic,1,m*n),2);
%偶数自然是0，奇数对应的值为1
for i=1:fix(m*n/8-1)
    if(to_proc(8*i-7:8*i)==[0 0 0 0 0 0 0 0])
        break;
    end
end
dms=to_proc(1:8*i-8);
%二进制转换为字符，因为reshape的特性，需要进行转置
message=char(bin2dec(num2str(reshape(dms,8,i-1)'))')

%jpeg编码解码
[R,M,N]=quan(pic);
[ACstream,DCstream,m,n]=haff(R,M,N);
[source]=dehaff(ACstream,DCstream,m,n);
pic=dequan(source,m,n,pic);
%解密
to_proc=mod(reshape(pic,1,m*n),2);
%偶数自然是0，奇数对应的值为1
for i=1:fix(m*n/8-1)
    if(to_proc(8*i-7:8*i)==[0 0 0 0 0 0 0 0])
        break;
    end
```

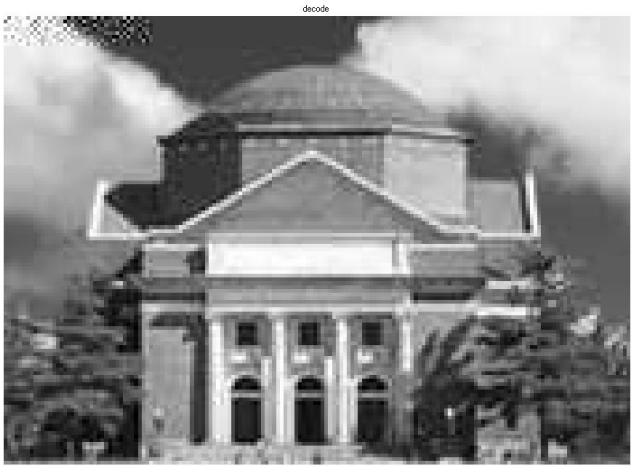

图 10: 类似空域的 DCT 加密

```
end
dms=to_proc(1:8*i-8);
%二进制转换为字符，因为reshape的特性，需要进行转置
message=char(bin2dec(num2str(reshape(dms,8,i-1)'))))'
```

(2) 依次实现本章介绍的三种变换域信息隐藏方法和提取方法，分析嵌密方法的隐蔽性以及嵌密后 JPEG 图像的质量变化和压缩比变化。

[5]

(a) 同空域方法，用信息位逐一替换掉每个量化后的 DCT 系数的最低位，再行熵编码。

信息可以很好的还原。隐藏的图像如图10, 得到 PSNR=34.6，压缩率 6.29。虽然 PSNR 很高，但可以明显看出左上方的噪点，这是改变前面的一部分 DCT 系数导致的。代码如下 (a3_3_2a.m)

```
clear;close all;clc;
load('hall');
pic=hall_gray;
%加密,这里只处理了8bit，所以只能加密英文字母
message='A quick brown fox jump over the lazy dog';
ms=reshape((dec2bin(message,8)-48)',1,8*length(message));
%以0000000为结束符，直接从第一个像素开始加
ms=[ms 0 0 0 0 0 0 0 0];
%解密

```

[5]把 JPEG 编码解码的四个过程——量化和 dct（quan.m）、haffman 编码（haff.m）, 反 haffman（dehaff.m）, 反量化与 idct（dequan.m）封装为四个函数，以方便调用

```
%jpeg 编码解码
%量化
[R,M,N]=quan(pic);
%这里加密用的R是zig-zag后的值显然R是64*len的矩阵
[~,len]=size(R);
%除以2，fix再乘2等于把最低bit清零，然后加上需要加密的信息
to_proc=reshape(R,1,64*len);
to_proc(1:length(ms))=fix(to_proc(1:length(ms))/2)*2+ms;
R1=reshape(to_proc,64,len);

%haffman 编码
[ACstream,DCstream,m,n]=haff(R1,M,N);
[source]=dehaff(ACstream,DCstream,m,n);
%解密
proc=mod(reshape(source,1,64*len),2);
%偶数自然是0，奇数对应的值为1
for i=1:fix(8*len-1)
    if(proc(8*i-7:8*i)==[0 0 0 0 0 0 0 0])
        break;
    end
end
dms=proc(1:8*i-8);
%二进制转换为字符，因为reshape的特性，需要进行转置
message=char(bin2dec(num2str(reshape(dms,8,i-1)'))')

pic=dequan(source,m,n,pic);
```

(b) 同方法 1, 用信息位逐一替换掉若干量化后的 DCT 系数的最低位, 再行熵编码。注意不是每个 DCT 系数都嵌入了信息。

考虑到人眼对于高频分量不敏感，将信息加载在最后两行（zigzag 后的）。产生的图片如图11, 可以看见明显的高频斑点。这是因为量化过程实际上减小了高频分量，这里在高频上增加信息在信息为 1 的部分相当增强了右下角的高频（第 64 个）信号，因此产生了类似棋盘的斑点。虽然能够恢复信息，但是加密效果并不好，图片上太明显了。

容易想到的是，合理选择被替换位可以一定程度上解决这一问题，但是下一问的方法更为合理，这里不再进行调整。PSNR=32.24，压缩比 5.49. 代码如下 (a3_3_2a.m)

```
clear;close all;clc;
load('hall');
pic=hall_gray;
```

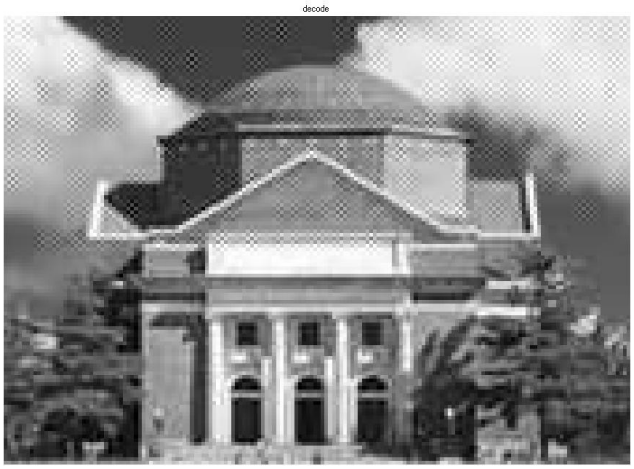

图 11: 特定位 DCT 加密

```
%加密,这里只处理了8bit，所以只能加密英文字母
message='A quick brown fox jump over the lazy dog';
ms=reshape((dec2bin(message,8)-48)',1,8*length(message));
%以0000000为结束符，直接从第一个像素开始加
ms=[ms 0 0 0 0 0 0 0 0];
%解密

%jpeg编码解码
%量化
[R,M,N]=quan(pic);
%这里加密用的R是zig-zag后的值显然R是64*len的矩阵
[~,len]=size(R);
%除以2，fix再乘2等于把最低bit清零，然后加上需要加密的信息
%对R（zigzag所得）的第63\64行（即高频分量）进行处理
to_proc=reshape(R(64:-1:63,:)',1,2*M*N);
to_proc(1:length(ms))=fix(to_proc(1:length(ms))/2)*2+ms;
R(64:-1:63,:)=reshape(to_proc',2,M*N);

%haffman 编码
[ACstream,DCstream,m,n]=haff(R,M,N);
[source]=dehaff(ACstream,DCstream,m,n);
%解密
proc=mod(reshape(source(64:-1:63,:),1,2*M*N),2);
%偶数自然是0，奇数对应的值为1
for i=1:fix(M*N/8)-1
    if(proc(8*i-7:8*i)==[0 0 0 0 0 0 0 0])
```

图 12: 最后一个 DCT 非零系数后加密

```
30            break;
31        end
32    end
33    dms=proc(1:8*i-8);
34    %二进制转换为字符，因为reshape的特性，需要进行转置
35    message=char(bin2dec(num2str(reshape(dms,8,i-1)'))))'
36
37    pic=dequan(source,m,n,pic);
```

(c) 先将待隐藏信息用 1, −1 的序列表示, 再逐一将信息位追加在每个块 Zig- Zag 顺序的最后一个非零 DCT 系数之后; 如果原本该图像块的最后一个系数就不为零, 那就用信像息位替换该系数;

这一问因为是需要对 zigzag 的后的值逐个处理，所以将加密和解密过程分别放入 haffman 编码与解码的循环中。

因为整个图被分为 315 块，所以这种方法只能存储 315bits 的信息，之前的信息有些太长了，这里把信息换为经典的‘You jump I jump’。加密后的图片如11, 肉眼与原图不存在差异，PSNR=34.30。

代码如下 (a3_3_2a.m)

```
1  clear;close all;clc;
2  load('hall');
3  load('JpegCoeff');
4  pic=hall_gray;
5  %加密,这里只处理了8bit，所以只能加密英文字母,因为图像被分为315块,
       所以只能加密315bits
6  %因此改用短一些的句子
7  message='You␣jump␣I␣jump';
8  ms=reshape((dec2bin(message,8)-48)',1,8*length(message));
```

```
%以0000000为结束符
ms=2*[ms 0 0 0 0 0 0 0 0]-1;

%jpeg编码解码
%量化
[R,M,N]=quan(pic);
%haffman 编码
ERR_DC=[2*R(1,1) R(1,1:N*M-1)]-R(1,:);
%DC部分的编码
Category_DC=ceil(log2(abs(ERR_DC)+1));
DCstream=logical([]);
%AC部分
ACstream=logical([]);
Size_AC=ceil(log2(abs(R)+1));
zero16=[1 1 1 1 1 1 1 1 0 0 1];
for i=1:M*N
    DCstream=[DCstream DCTAB(Category_DC(i)+1,2:1+DCTAB(Category_DC(i
        )+1,1))];
    if ERR_DC(i)>=0;%减48是因为0的asscii码
        DCstream=[DCstream dec2bin(ERR_DC(i))-48];
    else%一补码
        DCstream=[DCstream ~(dec2bin(-ERR_DC(i))-48)];
    end

    AC_NONE_ZERO=[1;1+find(R(2:64,i))];
    %%%%%%%%%%%%%%%%%
    %%信息写入
    if(i<=length(ms))
        if(AC_NONE_ZERO(end)==64)
            R(64,i)=ms(i);
            Size_AC(64,i)=1;
        else
            R(AC_NONE_ZERO(end)+1,i)=ms(i);
            AC_NONE_ZERO=[AC_NONE_ZERO;AC_NONE_ZERO(end)+1];
            Size_AC(AC_NONE_ZERO(end),i)=1;
        end
    end
    %%%%%%%%%%%%%%%%%
    if length(AC_NONE_ZERO)~=1
```

```
            for k=2:length(AC_NONE_ZERO)
                count0=AC_NONE_ZERO(k)-AC_NONE_ZERO(k-1)-1;
                while count0>15
                    ACstream=[ACstream zero16];
                    count0=count0-16;
                end
                ACstream=[ACstream ACTAB( (count0*10+Size_AC(
                    AC_NONE_ZERO(k),i)) ,...
                4:(3+ACTAB(count0*10+Size_AC(AC_NONE_ZERO(k),i),3)) )];
                if R(AC_NONE_ZERO(k),i)>=0;%减48是因为0的asscii码
                    ACstream=[ACstream dec2bin(abs( R(AC_NONE_ZERO(k),i)
                        ))-48];
                else%一补码
                    ACstream=[ACstream ~(dec2bin(abs( R(AC_NONE_ZERO(k),i
                        ) ))-48)];
                end
            end
        end
        ACstream=[ACstream 1 0 1 0];
    end

    [ACstream,DCstream,m,n]=haff(R,M,N);
    %解密
    %判断DC
    i=1;
    DC=[];
    %解码至残差
    while(i<=length(DCstream))
        for ca=1:12
            try
                if(DCstream(i:i+DCTAB(ca,1)-1)==DCTAB(ca,2:1+DCTAB(ca,1))
                    );
                    i=i+DCTAB(ca,1);
                    if(ca==1)
                        DC=[DC 0];
                        i=i+1;
                        break
                    elseif(DCstream(i)==1)
                        %此处为ca(Category的序号)与数字位数的对应关系
```

```
                    DC=[DC  bin2dec(num2str(DCstream(i:i+ca-2)))];
                else
                    DC=[DC -bin2dec(num2str(~DCstream(i:i+ca-2)))];
                end
                i=i+ca-1;
                break;
            end
        end
    end
end
%反差分
for  i=2:length(DC)
    DC(i)=DC(i-1)-DC(i);
end
%判断AC,并得到数据串
i=1;count=1;
AC=zeros(63,length(DC));tmp=[];
proc=[];
%解码至zig-zag前
while(i<=length(ACstream))
        %判断EOB
        if(ACstream(i:i+3)==[1  0  1  0])
            i=i+4;
            AC(:,count)=[tmp;zeros(63-length(tmp),1)];
            noz=find(tmp);
            if(noz)
                proc=[proc  tmp(noz(end))];
            end
            tmp=[];
            count=count+1;
            continue;
        end
        %判断ZRL
        if(i+10<=length(ACstream))
            if(ACstream(i:i+10)==[1  1  1  1  1  1  1  1  0  0  1])
                i=i+11;
                tmp=[tmp;zeros(16,1)];
                continue;
            end
```

```
121             end
122             for j=1:160
123                 %避免溢出
124                 try
125                     if(ACstream(i:i+ACTAB(j,3)-1)==ACTAB(j,4:3+ACTAB(j,3)
                        ));
126                         i=i+ACTAB(j,3);
127                         Run=fix((j-1)/10);
128                         if(ACstream(i)==1)
129                             %此处为run/size与数字位数的对应关系
130                             tmp=[tmp;zeros(Run,1);bin2dec(num2str(
                                ACstream(i:i+j-10*Run-1)))];
131                         else
132                             tmp=[tmp;zeros(Run,1);-bin2dec(num2str(~ACstream(
                                i:i+j-10*Run-1)))];
133                         end
134                         i=i+j-10*Run;
135                         break;
136                 end
137             end
138         end
139     end
140
141     %反zigzag
142     source=[DC;AC];
143     %偶数自然是0，奇数对应的值为1
144     %二进制转换为字符，因为reshape的特性，需要进行转置
145     %解密
146     %偶数自然是0，奇数对应的值为1
147     for i=1:fix(M*N/8)-1
148         if(proc(8*i-7:8*i)==([0 0 0 0 0 0 0 0]-1))
149             break;
150         end
151     end
152     dms=uint8((proc(1:8*i-8)+1)/2);
153     %二进制转换为字符，因为reshape的特性，需要进行转置
154     message=char(bin2dec(num2str(reshape(dms,8,i-1)'))))'
155
156     pic=dequan(source,m,n,pic);
```

4 人脸检测

(1) 所给资料 Faces 目录下包含从网图中截取的 28 张人脸, 试以其作为样本训练人脸标准 v

开始时的思路是逐一遍历所有的颜色（c3），然后统计落在该颜色附近（0.5 步长内）的图像点数，从而得到概率密度。但是因为颜色的数量太多，对于 L，这是一个指数型的算法，实在太慢。即使我使用 parpool 这一并行工具箱 4 线程运算[6]，跑完 L=5 的情况还是需要 5 到 6 分钟[7]。在这种情况下，我甚至想到用 GPU 库（CUDA）去进行优化。这部分代码可以参看注释部分。

在与同学讨论这一问题时，受到同学提醒[8]，我发现，可以换一种思路: 寻找图片的每一个点所在的颜色范围，很大程度上化简了运算。这样，计算复杂度就和 L 关系不大了，大大提升了计算速度，L=5 的运算时间缩短到了数秒。而且由于完全是矩阵运算，没有循环，这里也不再需要使用并行算法，也很大程度上减小了程序的复杂度。这里在网上查找了 histc 函数的用法，以对处理过的像素点进行统计。

L=3,4,5 所得值分别存储在 face_3.mat,face_4.mat,face_5.mat 中。

(a) 样本人脸大小不一, 是否需要首先将图像调整为相同大小?

并不需要，因为这里求的是比例，和点数无关。

(b) 假设 L 分别取 3, 4, 5, 所得三个 v 之间有何关系?

其中每个的大小都是前一个的 8（2^3 倍），这一点从所得到的 mat 文件大小也可以看出来, 而相同区间的概率密度和应该是一致的。

代码如下 (a3_4_1.m)

```
%mypool=parpool();
%修改L可以改变采样率
L=5
%c1=2^(7-L)-1:2^(8-L):2^8-1;
v=zeros(2^(3*L),1);
%c3=[reshape(repmat(c1,2^(2*L),1),2^(3*L),1) reshape(repmat(c1,2^L,2^
    L),2^(3*L),1) repmat(c1',2^(2*L),1)];
for j=1:31
    filename=['Faces/' num2str(j) '.bmp'];
    pic=imread(filename);
    v=v+anay_face(pic,L);
end
%delete(mypool);
v=v/31;
save face_5.mat v L
```

[6]这里参考了http://blog.csdn.net/dang_wang/article/details/35553953
[7]core i7 4800mq, 仍然很慢
[8]王璞瑞同学

anay_face.m 代码如下

```
function v=anay_face(pic,L)
%filename   文件名
%f   特征向量的和
%L 每种颜色位数
pic=double(pic);
[x,y,~]=size(pic);
v=zeros(2^(3*L),1);
%除以步长,+2^(7-L)是为了解决最小和最大值的问题,
%避免超出范围
pic=round((pic+2^(7-L))/2^(8-L))-1;
to_proc=reshape(pic,1,x*y,3);
n=2^(2*L)*to_proc(1,:,1)+2^L*to_proc(1,:,2)+to_proc(1,:,3)+1;
A=unique(n);
n1=histc(n,A);
%找到重复的元素和重复次数
v(A)=n1/x/y;

%anay_face非常适合并行处理
%并行处理
%这样比较大小时可以直接求绝对值然后用小于
%pic=pic-0.5;
%step=2^(7-L);
%l=2^(3*L);
%pict1=pic(:,:,1);
%pict2=pic(:,:,2);
%pict3=pic(:,:,3);
%c1=c(:,1);
%c2=c(:,2);
%c3=c(:,3);
%parfor i=1:l;
%    a= (abs(pict1-c1(i))<step)...
%      &(abs(pict2-c2(i))<step)...
%      &(abs(pict3-c3(i))<step);
%     v(i)=sum(sum(a))/x/y;
%end
return;
```

图 13: 识别中间过程

图 14: 长方形边框

(2) 设计一种从任意大小的图片中检测任意多张人脸的算法并编程实现 (输出图像在判定为人脸的位置加上红色的方框)。随意选取一张多人照片 (比如支部活动或者足球比赛), 对程序进行测试。尝试 L 分别取不同的值, 评价检测结果有何区别。

主要的问题是分割。这里的思路是，先在图中找到和之前所得 V 中出现频率最高的数种颜色相近的点，然后从这些点向四周拓展，符合条件就把 index 矩阵中对应的元素变为 1，不符合就把这个点去掉, 更换扩展范围多次重复即可。得到图13。最后对 index 在 x 和 y 方向上分别差分即可得到边框。

这里得到的并不是一个长方形方框，而是把人脸围起来的一个不规则边框。虽然利用 find 函数的线性性[9]可以将其转换为标准的长方型如图14，但是这样降低了准确度。

还有一个问题是开始寻找点时会有很多孤立的点，它们不可能是人脸，虽然后来的变换可以消除其影响，但是太影响运算速度。这里通过在网络上查询，采用 bwareaopen 函数滤掉即可。

通过测试，L=4 时最好的 d 取值为 0.425，得到的图像为图15;L=5 时最好的 d 取值为 0.65，得到的图像为图16。L3 的效果较差，无法避开第三个人的脖子的同时识别后一排的人的脸，可见 pic/ex2_3.jpg 文件，故没有精细调参。

[9]这部分代码见注释, 看了 find 的 help，对此没有很好的解释，但 find 在这里起到了这一作用

图 15: L=4,d=0.425

图 16: L=5,d=0.65

可以看出，L=5 时所得的框要比 L=4 精细很多 (能分辨更小的色差)，能够更好的包围住人物的脸[10]。L=4 要比 L=5 快大概 0.3s（两者都在 1s 以内）。所以认为 L=5 的效果更好。

同时选用另一张照片进行测试，L=5，d=0.65，的到图像如图17, 效果也不错。

代码入下 (a3_4_2.m)

```
clear;close all;clc;
load face_5;
% mypool=parpool();
exam=imread('Faces/example.bmp');
%exam=imrotate(exam,90); %旋转
[m,n,~]=size(exam);
%修改颜色
%exam=imadjust(exam,[.1 0 0;.6 .7 1],[]);
%拉伸
```

[10]中间两人脸右侧多出来的框是因为后面的背景是队列的其他人，色差太小

图 17: L=5,d=0.65

```
% n=2*n;
% exam=imresize(exam,[m n]);
exam=double(exam);
%存储所选的颜色
c1=2^(7-L)-1:2^(8-L):2^8-1;
c3=[reshape(repmat(c1,2^(2*L),1),2^(3*L),1) reshape(repmat(c1,2^L,2^L
    ),2^(3*L),1) repmat(c1',2^(2*L),1)];
[~,list]=sort(v,'descend');

index_find=zeros(m,n,10);
step=8;
for i=1:6
    index_find(:,:,i)=(abs(exam(:,:,1)-c3(list(i),1))<step)...
                &(abs(exam(:,:,2)-c3(list(i),2))<step)...
                &(abs(exam(:,:,3)-c3(list(i),3))<step);
end
index=sum(index_find,3);
%除去单个噪点
index=bwareaopen(index,2);
 for k=[90 80 40 35 ]
     [y,x]=find(index);
     u0=zeros(2^(3*L),1);
     u=zeros(2^(3*L),length(x));
     mk=fix(m/k);nk=fix(n/k);
     for i=1:length(x)index=(index~=0);

         try %去除在图像边缘的点
```

```
36                  pic=exam((y(i)-mk):(y(i)+mk),(x(i)-nk):(x(i)+nk),:);
37                  u(:,i)=anay_face(pic,L);
38                  d=1-sum(sqrt(v.*u(:,i)));
39                  if(d>0.65)
40                      index(y(i),x(i))=0;
41                  else
42                      index(y(i),x(i))=1;
43                  end
44              catch
45                  index(y(i),x(i))=0;
46              end
47          end
48      clear u x y;
49      end
50  [y,x]=find(index);
51
52
53      for i=1:length(x)
54          index(y(i)-mk:y(i)+mk,x(i)-nk:x(i)+nk)=1;
55      end
56  %覆盖区间变为长方形
57  %[y,x]=find(index);
58  %index=zeros(m,n);
59  %index(y,x)=1;
60
61  %差分得到边框
62  index=(index~=0);
63  e_index=2*index-[zeros(m,1) index(:,1:end-1)]-[zeros(1,n);index(1:end
        -1,:)];
64  exam(:,:,1)=exam(:,:,1)+255*(e_index~=0);
65  imshow(uint8(exam));
66  % delete(mypool);
```

(3) 对上述图像分别进行如下处理后再试试你的算法检测结果如何？并分析所得结果

(a) 顺时针旋转 90° (imrotate);

所得图像为18, 可以看出，红框跟着图片一起转了 90°，因为本算法为检验颜色，和旋转角度无关。旋转代码见 a3_4_2.m 第 5 行。

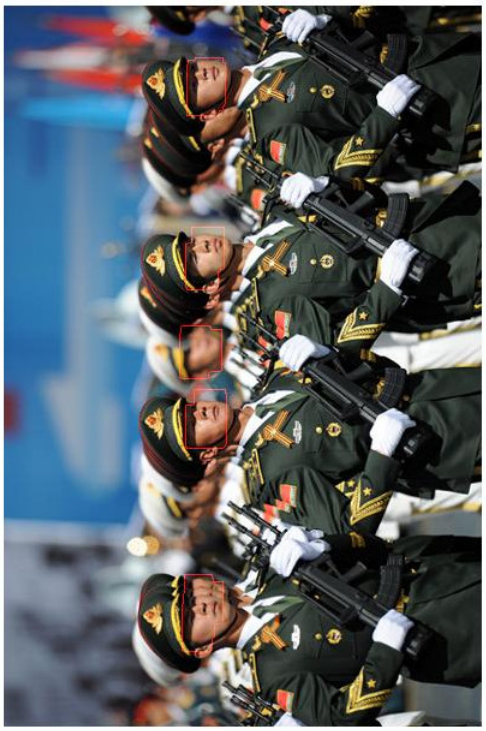

图 18: 旋转 90°

(b) 保持高度不变，长度拉伸为两倍

第一次我得到的图像为19, 第二个人的脸无法被识别，debug 后发现，因为我把图片转换成了 double 型，resize 时其中出现了负数，由此导致了 bug。在 resize 后再转换图片类型就可以解决这一 bug.

修正 bug 后，得到图像20, 中间两人的脖子也被标了出来，这是因为图片被拉伸后脖子相对面积变大导致的。

所用代码为 a3_4_2.m 的 9 至 11 行。

图 19: 错误的拉伸处理

图 20: 正确的拉伸处理

图 21: 改变颜色

(4) 适当改变颜色（imadjust）

得到图21, 可以看到什么都没有被识别，这是因为本算法是基于颜色的，修改颜色后自然不能识别

(5) 如果可以重新选择人脸样本训练标准，你觉得应该如何选取

我觉得首先应该选择单一人种的照片，胡子不能浓密，不应有眼镜、墨镜等装饰，光照应该自然（避免阴阳脸），最好分人种，分光照情况选择判定标准。

5 总结

本实验非常有趣，结合广泛使用的 JPEG 和人脸识别技术使得我更加熟悉 MATLAB 的矩阵运算、图像工具；也让我对图像处理的方法有了一定的认识；还让我明白图像处理多为并行处理，这也是 GPU 出现的原因; 更难能可贵的是人脸识别部分让我理解了算法对性能的极大影响。

人脸识别的发散性题目非常有意思，在其中有过很多思路，也有很多问题和困难，最后的解决方案往往很有创意，令人兴奋。

不过在写的时候，感到自己的 matlab 能力还是不太扎实，一些地方还需要去网上查询，不过，多进行一些类似本实验的训练，我相信自己的能力能够得到更大的提升。