《图像处理导论》

第一次作业

学院：计算机科学与工程学院

班级：计算机1606班

姓名：戚子强

学号：20164625

**作业内容：**

对一张图片使用高斯噪声、椒盐噪声，和均值滤波、中值滤波进行处理。

**结果展示：**



**代码实现：**

Matlab代码如下：

clear;

pic = imread('t3.jpg');

pic1 = rgb2gray(pic);

pic21 = imnoise(pic1,'gaussian',0.01,0);

%目前保持平滑性，仅增加1%扰动，第四个参数为0则不考虑平滑

pic22 = imnoise(pic21,'salt & pepper',0.08);

%第三个参数范围[0,1]，越大噪声越大

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%均值滤波的实现，也可以使用自带函数filter2(fspecial('average',n);,g)/255;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 3; %模板大小

template = ones(n);

[height, width] = size(pic22);

x1 = double(pic22);

x2 = x1;

for i = 1:height-n+1

for j = 1:width-n+1

c = x1(i:i+n-1,j:j+n-1).\*template;

s = sum(sum(c));

x2(i+(n-1)/2,j+(n-1)/2) = s/(n\*n);

end

end

pic23 = uint8(x2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%中值滤波的实现，也可以使用自带函数medfilt2(g,[n2 n2]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[height, width] = size(pic23);

x3 = double(pic23);

x4 = x3;

for i = 1:height-n+1

for j = 1:width-n+1

c = x1(i:i+n-1,j:j+n-1);

e = c(1,:);

for k = 2:n

e = [e, c(k, :)];

end

tmp = median(e);

x4(i+(n-1)/2,j+(n-1)/2) = tmp;

end

end

pic24 = uint8(x4);

figure;

subplot(2,2,1);imshow(pic21);title('1.高斯噪声(基于原图):');

subplot(2,2,2);imshow(pic22);title('2.椒盐噪声(基于图1):');

subplot(2,2,3);imshow(pic23);title('3.均值滤波(基于图2):');

subplot(2,2,4);imshow(pic24);title('4.中值滤波(基于图3):');

《图像处理导论》

第二次作业

学院：计算机科学与工程学院

班级：计算机1606班

姓名：戚子强

学号：20164625

**题目：**

用傅里叶变换实现个人头像的边缘检测。

**结果展示：**

图片包含 屏幕截图

描述已自动生成

**代码：**

//Matlab实现

imo = imread('tx.jpg');

im = rgb2gray(imo);

% 计算频域中的差分算子

nx = size(im, 2);

hx = ceil(nx/2)-1;

ftdiff = (2i\*pi/nx)\*(0:hx);

ftdiff(nx:-1:nx-hx+1) = -ftdiff(2:hx+1); % 共轭对称

g = ifft2( bsxfun(@times, fft2(im), ftdiff) ); % FFT

% Result

figure;

subplot(2,1,1);imshow(imo);title('原图像：');

subplot(2,1,2);imshow(g,[]);title('处理后：');

//OpenCV & C++实现

#include <opencv2/opencv.hpp>

#include <iostream>

#include <math.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace cv;

using namespace std;

int main()

{

//读取图像

Mat src\_image = imread("t2.jpg");

//图像读取出错处理

if (!src\_image.data)

{

cout << "src image load failed!" << endl;

return -1;

}

//显示源图像

namedWindow("原图像", WINDOW\_NORMAL);

imshow("原图像", src\_image);

//此处高斯去燥有助于后面二值化处理的效果

//Mat blur\_image;

//GaussianBlur(src\_image, blur\_image, Size(15, 15), 0, 0);

//imshow("GaussianBlur", blur\_image);

/\*灰度变换与二值化\*/

Mat gray\_image, binary\_image;

cvtColor(src\_image, gray\_image, COLOR\_BGR2GRAY);

threshold(gray\_image, binary\_image, 30, 255, THRESH\_BINARY | THRESH\_TRIANGLE);

//imshow("binary", binary\_image);

/\*形态学闭操作\*/

Mat morph\_image;

Mat kernel = getStructuringElement(MORPH\_RECT, Size(3, 3), Point(-1, -1));

morphologyEx(binary\_image, morph\_image, MORPH\_CLOSE, kernel, Point(-1, -1), 2);

//imshow("morphology", morph\_image);

/\*查找外轮廓\*/

vector< vector<Point> > contours;

vector<Vec4i> hireachy;

findContours(binary\_image, contours, hireachy, CV\_RETR\_EXTERNAL, CHAIN\_APPROX\_NONE, Point());

int l;//目标轮廓索引

//寻找最大轮廓，即目标轮廓

for (size\_t t = 0; t < contours.size(); t++)

{

/\*过滤掉小的干扰轮廓\*/

Rect rect = boundingRect(contours[t]);

if (rect.width < src\_image.cols / 2)

continue;

//if (rect.width >(src\_image.cols - 20))

l = t;//找到了目标轮廓，获取轮廓的索引

}

//画出目标轮廓

Mat result\_image = Mat::zeros(src\_image.size(), CV\_8UC3);

vector< vector<Point> > draw\_contours;

draw\_contours.push\_back(contours[l]);

drawContours(result\_image, draw\_contours, -1, Scalar(255, 255, 255), 1, 8, hireachy);

namedWindow("处理后", WINDOW\_NORMAL);

imshow("处理后", result\_image);

//计算轮廓的傅里叶描述子

Point p;

int x, y, s;

int i = 0, j = 0, u = 0;

s = (int)contours[l].size();

Mat src1(Size(s, 1), CV\_8SC2);

float f[9000];//轮廓的实际描述子

float fd[16];//归一化后的描述子，并取前15个

for (u = 0; u < s; u++)

{

float sumx = 0, sumy = 0;

for (j = 0; j < s; j++)

{

p = contours[l].at(j);

x = p.x;

y = p.y;

sumx += (float)(x \* cos(2 \* CV\_PI \* u \* j / s) + y \* sin(2 \* CV\_PI \* u \* j / s));

sumy += (float)(y \* cos(2 \* CV\_PI \* u \* j / s) - x \* sin(2 \* CV\_PI \* u \* j / s));

}

src1.at<Vec2b>(0, u)[0] = sumx;

src1.at<Vec2b>(0, u)[1] = sumy;

f[u] = sqrt((sumx \* sumx) + (sumy \* sumy));

}

//傅立叶描述字的归一化

f[0] = 0;

fd[0] = 0;

for (int k = 2; k < 17; k++)

{

f[k] = f[k] / f[1];

fd[k - 1] = f[k];

cout << fd[k - 1] << endl;

}

//保存数据

for (int k = 0; k < 16; k++)

{

FILE\* fp = fopen("1.txt", "a");

fprintf(fp, "%8f\t", fd[k]);

fclose(fp);

}

FILE\* fp = fopen("1.txt", "a");

fprintf(fp, "\n");

fclose(fp);

waitKey();

return 0;

}

《图像处理导论》

大作业

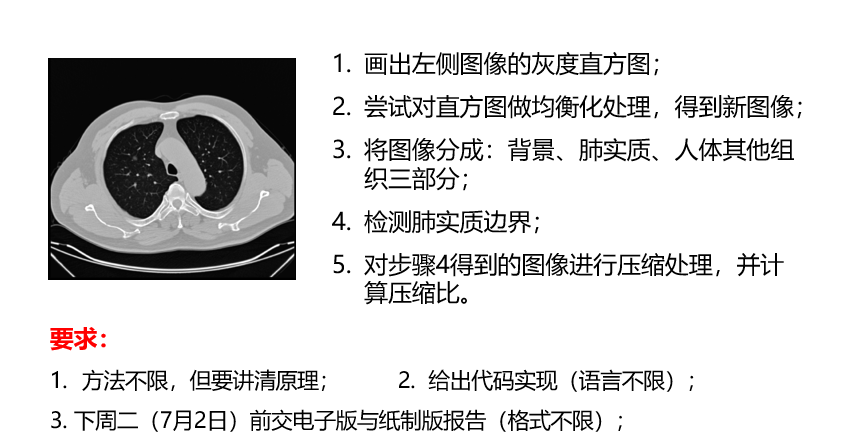
学院：计算机科学与工程学院

班级：计算机1606班

姓名：戚子强

学号：20164625

**一、题目：**



**二、解题思路及结果：**

**2.1 题目1：**

将彩色图像转化成为灰度图像的过程成为图像的灰度化处理。彩色图像中的每个像素的颜色有R、G、B三个分量决定，而每个分量有255中值可取，这样一个像素点可以有1600多万（255\*255\*255）的颜色的变化范围。而灰度图像是R、G、B三个分量相同的一种特殊的彩色图像，其一个像素点的变化范围为255种，所以在数字图像处理种一般先将各种格式的图像转变成灰度图像以使后续的图像的计算量变得少一些。灰度图像的描述与彩色图像一样仍然反映了整幅图像的整体和局部的色度和亮度等级的分布和特征。图像的灰度化处理可用两种方法来实现。

第一种方法使求出每个像素点的R、G、B三个分量的平均值，然后将这个平均值赋予给这个像素的三个分量。第二种方法是根据YUV的颜色空间中，Y的分量的物理意义是点的亮度，由该值反映亮度等级，根据RGB和YUV颜色空间的变化关系可建立亮度Y与R、G、B三个颜色分量的对应：Y=0.3R+0.59G+0.11B，以这个亮度值表达图像的灰度值。

在本题目中，利用Matlab自带的rgb2gray函数，之家将读入的原图进行灰度处理，并使用imhist函数输出其直方图即可。

**2.2 题目2：**

直方图均衡化的总体思想是：首先考虑连续函数并且让变量r代表待增强图像的灰度级，假设被归一化到区间[0,1]，且r=0表示黑色及r=1表示白色。然后再考虑一个离散公式并允许像素值在区间[0,L-1]内。

具体可采用的方法有很多，如灰度级变换方法，保证原图各灰度级在变换后仍保持从黑 到白（或从白到黑）的排列次序，保证变换前后灰度值动态范围的一致性即可。

在本题目中，使用Matlab自带的histeq函数进行处理，得到结果，如下图所示：

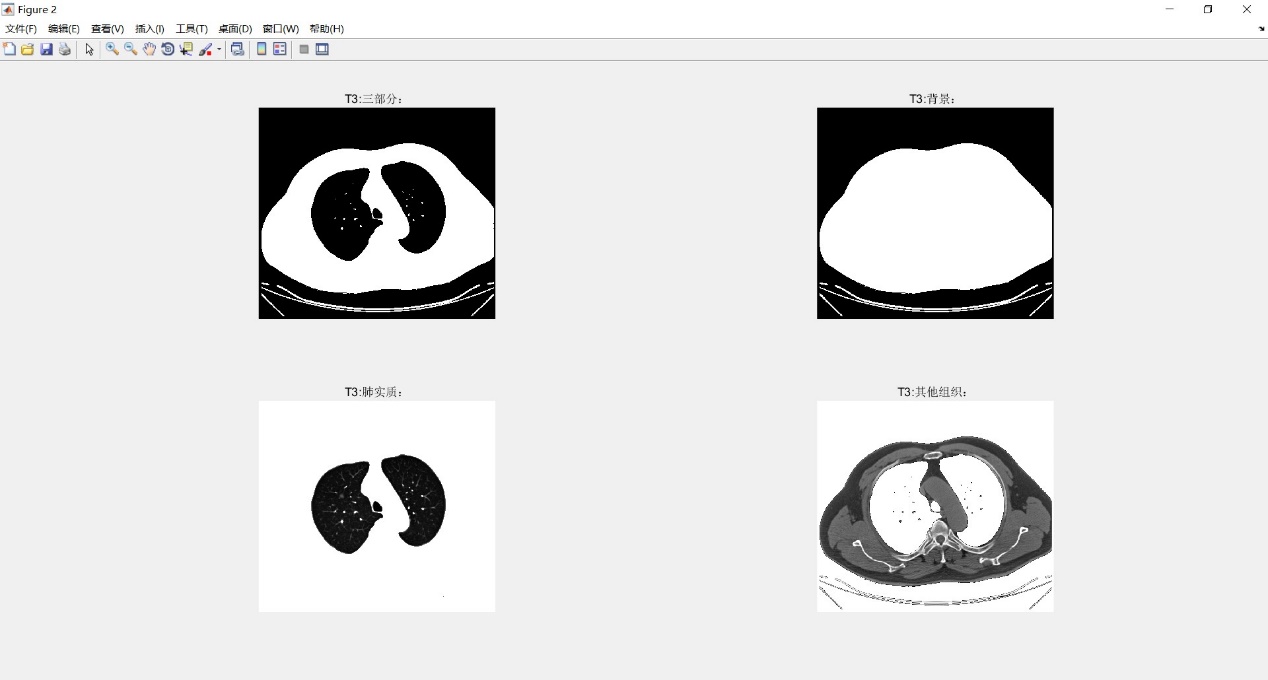
图片包含 屏幕截图

描述已自动生成

**2.3 题目3：**

总体思路是先将灰度图像二值化，然后对图像填充，身体内部为1，外部为0，之后对图像进行二次分割，以“身体-外界”和“肺部-身体”为两个界限。

matlab中DIP工具箱函数im2bw使用阈值变换法把灰度图像转换成二值图像。一般意义上是指只有纯黑（0）、纯白（255）两种颜色的图像。imfill是Matlab软件中自带的用于二值图像孔洞填充的函数，对于BW = imfill(BW,'holes')，填充二值图像中的空洞区域。如黑色的背景上有个白色的圆圈。则这个圆圈内区域将被填充。然后根据图示，从外向内对三大部分进行填充和显示，得出最终效果，如下图所示：



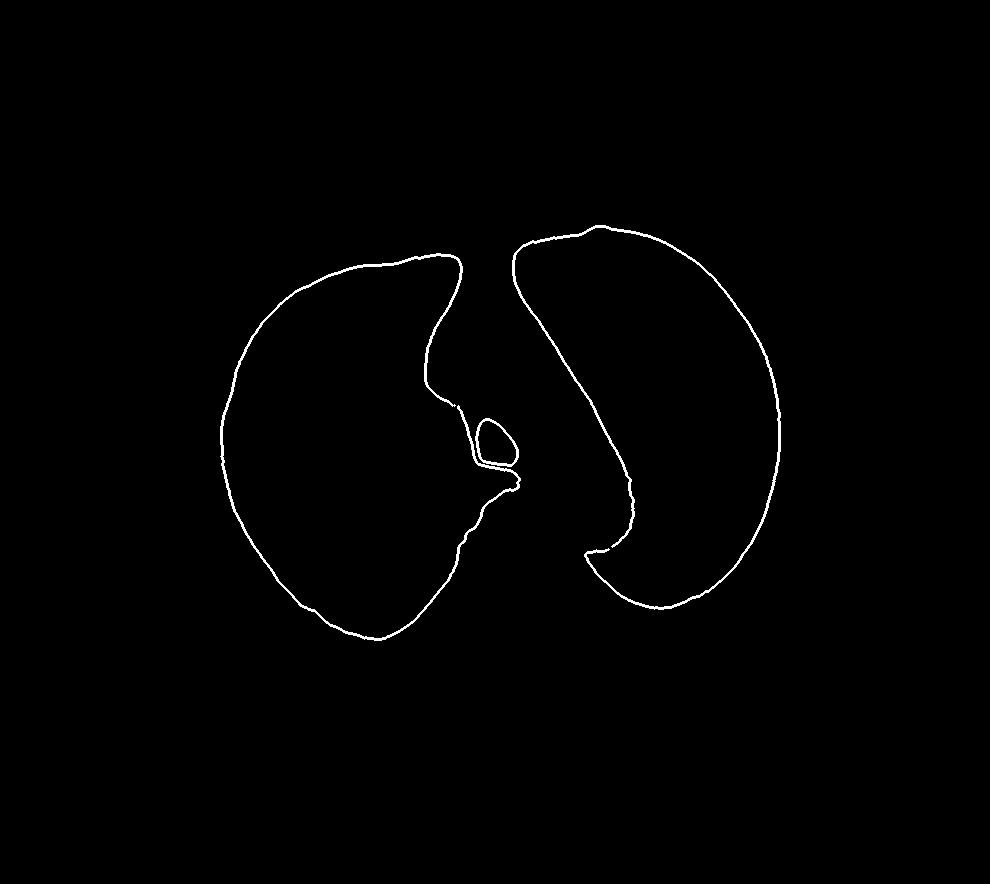
**2.4 题目4：**

在本题目中，由于肺部内部有许多白色的小点儿影响处理效果，所以要先去掉肺部中的白色小点儿，然后对图像进行膨胀、腐蚀和粘连，使图像尽可能得保留整体特征而去除局部的干扰因素。

膨胀和腐蚀操作的最基本组成部分是结构元素，其用于测试输出图像，通常要比待处理的图像小的多。二维平面结构元素由一个数值为0或1的矩阵组成。结构元素的原点指定了图像中需要处理的像素范围，结构元素中数值为1的点决定结构元素的邻域像素在进行膨胀或腐蚀操作时是否需要参与计算。腐蚀即删除对象边界某些像素，膨胀则是给图像中的对象边界添加像素。膨胀的算法：用3x3的结构元素，扫描图像的每一个像素 用结构元素与其覆盖的二值图像做“与”操作 如果都为0，结果图像的该像素为0，否则为1。腐蚀的算法：用3x3的结构元素，扫描图像的每一个像素 用结构元素与其覆盖的二值图像做“与”操作 如果都为1，结果图像的该像素为1。否则为0。

具体做法是，先创建一个指定半径为6的平面圆盘形的结构元素。对于SE = strel('disk', R, N)函数，当N大于0时，圆盘形结构元素由一组N(或N+2)个周期线型结构元素来近似。当N等于0时，不使用近似，即结构元素的所有像素是由到中心像素距离小于等于R的像素组成。N可以被忽略，此时缺省值是4。然后进行图像腐蚀，可以使用imerode函数进行图像腐蚀。imerode函数需要两个基本输入参数：待处理的输入图像以及结构元素对象。此外，imerode函数还可以接受3个可选参数：PADOPT(padopt) ——影响输出图片的大小、PACKOPT(packopt).——说明输入图像是否为打包的二值图像(二进制图像)。Imdilate函数用于对图像实现形态学中的膨胀操作，用法为BW=imdilate(I,se)。

图像处理后如下图所示：

****

**2.5 题目5：**

首先做图像压缩是在频率域处理的，通过DCT（离散余弦变换）将图像转到频率域。低频部分也存储了图像的大多信息。我们知道，低频部分集中较多能量，含有图像大多平滑信息，而高频部分主要是边缘或者噪声。人眼对低频的光波比较敏感，故我们将高频部分合理丢掉部分，然后将频率域的图像进行量化处理，量化后的频率图像再进行编码处理，比如用哈夫曼编码来构造最短编码。通过编码后的图像就占用很少的一部分空间了。

压缩过程：

1、将图像分块，分成8\*8 像素的小块来分别计算。

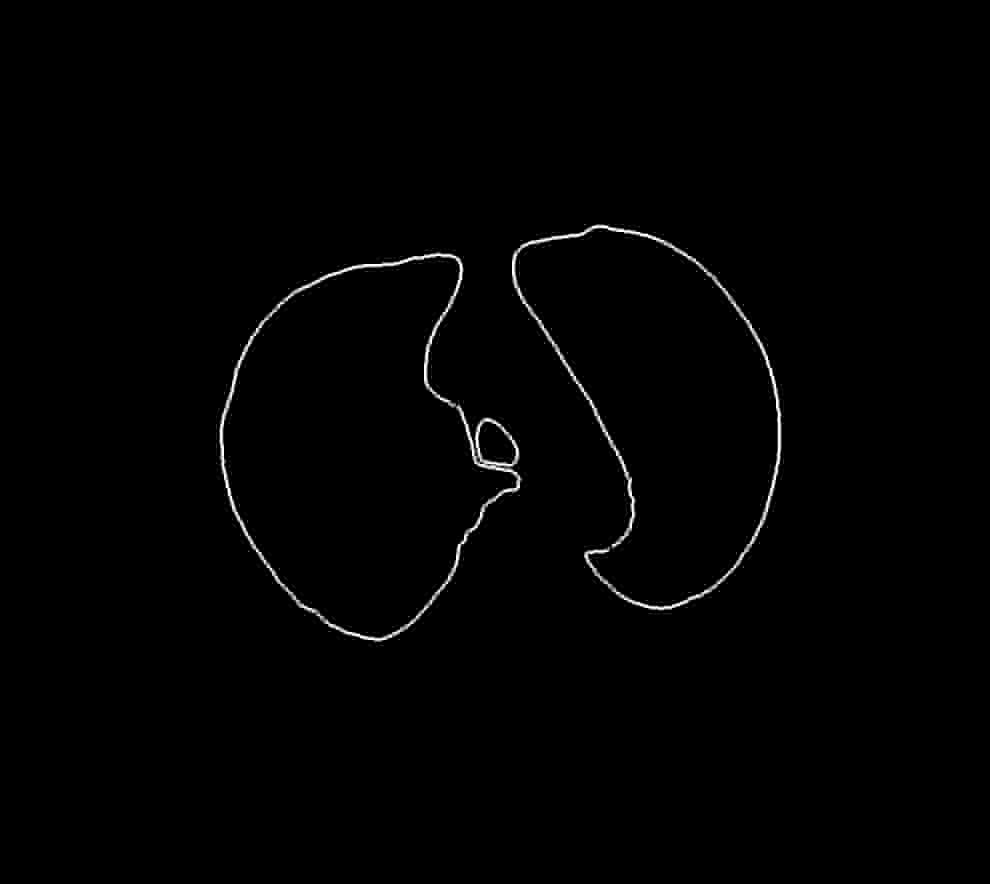
2、对每个小块进行理算余弦变换，将图像转换到频率域。

3、将图像量化，可通过量化矩阵（或者简单将每个像素除以N，取整，然后再乘以N，得到量化的目的，当然N越大，压缩的比率也就越大）

4、通过合适的编码规则来对量化后的图像编码

因为DCT可以说少计算了复数，更方便计算，所以这里用了DCT,而没有用FFT。压缩损失主要在量化的过程，图像做DCT变换只不过是一个从空间域到频率域的变换，并没有改变图像的属性。而量化的过程就是一个不断取整，保留大头，舍弃小的的过程。

压缩后效果如图：



压缩比为72.20%。

**附录代码：**

clear;

img = imread('大作业.png');

%题目1：转为灰度图，后续imhist输出直方图

img1 = rgb2gray(img);

%题目2：均衡化处理

img2 = histeq(img1);

%输出题目1和2的结果

figure;

subplot(2,2,1);imshow(img1,[]);title('T1:处理前：');

subplot(2,2,2);imhist(img1);title('T1:灰度直方图：');

subplot(2,2,3);imshow(img2,[]);title('T2:均衡化后图像：');

subplot(2,2,4);imhist(img2);title('T2:均衡化后直方图：');

%题目3：

%灰度图像二值化

img3 = im2bw(img1);

%图像填充，身体内部为1，外部为0

img4 = imfill(img3,'holes');

%图像二次分割，以“身体-外界”和“肺部-身体”为两个界限

[width, height] = size(img4);

for i=1:width

for j=1:height

if(img4(i,j)==1)

if(img3(i,j)==0)

img7(i,j)=img1(i,j);

img5(i,j)=255;

img6(i,j)=255;

else

img5(i,j)=img1(i,j);

img6(i,j)=255;

img7(i,j)=255;

end

else

img6(i,j)=img1(i,j);

img5(i,j)=255;

img7(i,j)=255;

end

end

end

figure;

subplot(221);imshow(img3,[]);title('T3:三部分：');

subplot(222);imshow(img4,[]);title('T3:背景：');

subplot(223);imshow(img7,[]);title('T3:肺实质：');

subplot(224);imshow(img5,[]);title('T3:其他组织：');

%题目4：

%去掉肺部中的白色小点儿

se = strel('disk',6);

img8 = imerode(img7,se);

%膨胀图像

img9 = imerode(img8,se);

se = strel('disk',7);

img10 = imdilate(img9,se);

%腐蚀图像

img11 = imdilate(img10,se);

%粘连图像

[height, width] = size(img7);

for i=1:height

for j=1:width

if((i>(height/2-40)&&i<(height/2+40))&&(j>(width/2-40)&&j<(width/2+40)))

m(i,j)=img7(i,j);

else

m(i,j)=img11(i,j);

end

end

end

image = edge(m,'log',4);

se = strel('disk',1);

img11 = imdilate(image,se);

figure;

imshow(img11,[]);title('T4:肺部：');

imwrite(img11,'resultT4.jpg');

%题目5：压缩图像

I = img11;

I = im2double(I); % 数据类型转换

T = dctmtx(8); % 计算二维离散DCT矩阵

dct = @(x)T \* x \* T'; % 设置函数句柄

B = blkproc(I,[8 8],dct); % 图像块处理

mask = [1 1 1 1 0 0 0 0 % 掩膜

1 1 1 0 0 0 0 0

1 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0];

B2 = blkproc(B,[8 8],@(x)mask.\* x); % 图像块处理

invdct = @(x)T' \* x \* T; % 设置函数句柄

I2 = blkproc(B2,[8 8],invdct); % 图像块处理

figure, imshow(I2),title('T5:压缩后（压缩比：72.20%）'); % 显示原始图像和压缩重构图像

imwrite(I2,'resultT5.jpg');