2019 年秋季《图像处理与分析》编程作业-05

【第九讲-图像形态学处理】

姓名：　　学号：

【问题描述】

编一个程序实现如下功能：

1、 读入一幅指纹图像（自己找）；

2、 对图像进行二值化（方法自定，可以是阈值法）；

2、采用形态学骨架提取和距离变换骨架提取两种算法分别提取图像骨架；

3、采用裁剪算法，并分析其效果。

问题分析：

本问题中涉及几个子问题，包括：图像的二值化、形态学骨架提取、距离变换骨架提取、裁剪算法的实现。下面对问题进行分析：

1、 图像的二值化

二值图像每个像素点位置的亮度不再由灰度表示，而是由布尔数值 True、False 来表示，当像素点值为 True 时显示白色、当像素点值为 False 时显示黑色。常用的简单二值化方式是阈值法。首先对定义灰度图像 I 的灰度阈值 th 为原始图像灰度的中间值[max(I)+min(I)]/2，当灰度大于 th 时二值化为 True；当灰度小于 th 时二值化为 False。

$$\mathrm{I}(x,y) = True, \quad when\ \mathrm{I}(x,y) >= th$$
$$\mathrm{I}(x,y) = False, \quad when\ \mathrm{I}(x,y) < th$$

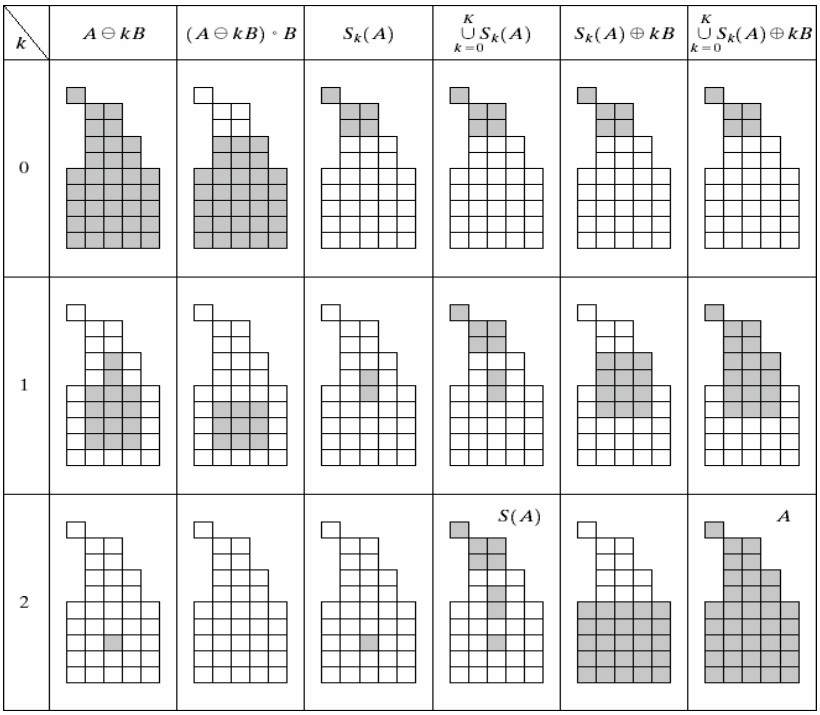
2、 形态学骨架提取

课堂上所提的形态学骨架提取法之一是通过腐蚀以及开运算来实现，其公式如下：

$$S(A) = \bigcup_{k=0}^{K} S_k(A)$$

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$$

公式表示，将经过 k 次基于 B 结构元的腐蚀变换减去该腐蚀变换结果再进行一次开运算的结果，直到腐蚀变换结果为空集时停止运算，将所有次的计算结果并在一起得到最终的骨架。一个运算的实例如下图：



根据上图算法，需要实现的是腐蚀还有膨胀操作，故设计如下的过程实现腐蚀和膨胀操作：

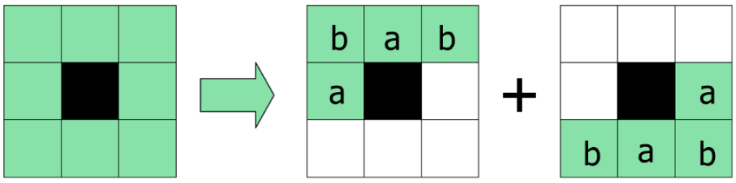
首先定义结构元矩阵 SE，其中包含元素值 1 和 0。将二值图像中的 True 和 False 转换为 1 和 0，再对结构元和图像进行卷积操作。

（1）**膨胀**：当卷积核卷积结果不等于零时，则说明二值图像中至少存在一个“True”元素处在结构元中“1”元素所在的位置，此时对卷积核所在的像素点赋值为 1；否则赋值为 0。最后将 0 转换为布尔值 False，1 转换为布尔值 True。

（2）**腐蚀**：当卷积核卷积结果不等于结构元中“1”的个数时，说明二值图像中“True”的像素点没有完全包含住结构元中“1”的位置，此时对卷积核所在的像素点位置赋值为 0；否则赋值为 1。最后将 0 转换为布尔值 False，1 转换为布尔值 True。

3、 距离变换骨架提取

距离变换图算法是一种针对栅格图像的特殊变换，是把二值图像变换为灰度图像，其中每个像素的灰度值等于它到栅格地图上相邻物体的最近距离。是把离散分布在空间中的目标根据一定的距离定义方式生成距离图，其中每一点的距离值是到所有空间目标距离中最小的一个。对于距离的量度是通过四方向距离（又称“城市块距离”或“出租车距离”）的运算来实现的，即只允许沿四个主方向而不允许沿对角方向进行跨栅格的最小路段的计数。因此，每个路段为一个像元边长。具体的变换如下图所示：



$$D(i,j)=Minimum(D(i-1,j-1)+b,D(i-1,j)+a,D(i-1,j+1)+b,D(i,j-1)+a,D(i,j)+0)$$
$$i=2,\ldots,m \qquad j=2,\ldots,n$$
$$D(i,j)=Minimum(D(i,j)+0,D(i,j+1)+a,D(i+1,j-1)+b,D(i+1,j)+a,D(i+1,j+1)+b)$$
$$i=m-1,\ldots,1 \qquad j=n-1,\ldots,1$$

根据上述算法，从距离变换图中提取出具有相对最大灰度值的那些像元所组成的图像就是图像的骨架。算法原理如下：

（1）求取二值图像的边界。边界提取可以通过形态学变化来实现，即用二值图像减去其腐蚀图像或者用二值图像的膨胀图像减去原图像来得到。

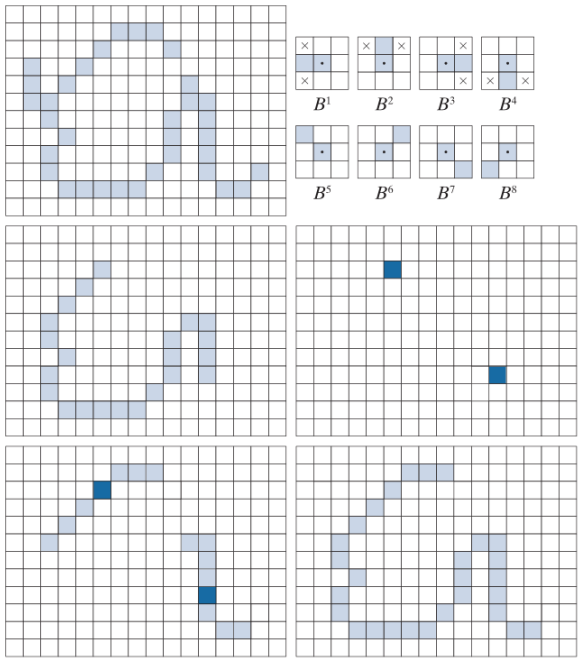
（2）对边界二值图像求取距离变换。经过两次循环，第一次从上至下、从左至右求解距离变换矩阵 D 的元素值；第二次循环从下至上、从右至左求解距离变换矩阵 D 的元素。

（3）求解距离变换图中的局部极大值。对距离变换图分别求水平和竖直方向的差分作为梯度，局部极大值位于梯度值符号变换的位置，根据这一原理将局部极大值选出并在对应像素点赋值为 True 得到一个新的二值图像。

（4）落入原二值图像中的局部极大值就是图像的骨架。将提取的局部最大值的二值图像与原二值图像求交集，得到的结果即为提取的骨架。

4、 剪裁算法

裁剪方法实际上是对细化和骨架绘制算法的补充，因为要清除这些算法产生的一些不必要的附加成分。基本的操作如下图所示：



(a) Set A of foreground pixels (shaded).
(b) SEs used for deleting end points.
(c) Result of three cycles of thinning.
(d) End points of (c).
(e) Dilation of end points conditioned on (a).
(f) Pruned image.

(1) 用一系列被设计用来检测终点的结构元素（B1~B8）对原图 A 进行细化得到 X1；

(2) 利用相同的结构元（B1~B8）对 X1 做击中击不中变换得到 X1 中所有端点集合 X2；

(3) 对端点集合 X2 在原图 A 的限定下进行三次膨胀处理得到 X3；

(4) 将 X3 与 X1 取并集得到 X4，即为裁剪后的图像。

实现细节及代码展示：

- **编程语言**：Python3.7+
- **所用模块**：Numpy1.17.2（用于矩阵处理），PIL5.3.0（用于图像读取等基本操作），matplotlib3.0.2（用于绘制图像等），Scipy1.1.0（调用其二维卷积函数）
- **自定义模块**：ErodeDilate.py（包含腐蚀函数与膨胀函数）、IM2BW.py（包含二值化函数）、CutImage.py（包含裁剪函数）、rgb1gray.py（包含自编写的 rgb 转灰度函数）

1、 二值化函数

```python
import numpy as np
def IM2BW(I):
    m,n=I.shape
    B=np.zeros((m,n))
    th=np.floor(np.max(I)+np.min(I))/2#定义阈值
    for i in range(m):
        for j in range(n):
            if I[i,j]>=th:
                B[i,j]=True         #灰度大于等于阈值，为True
            else:
                B[i,j]=False        #灰度小于阈值，为False
    return np.array(B,dtype='bool')
```

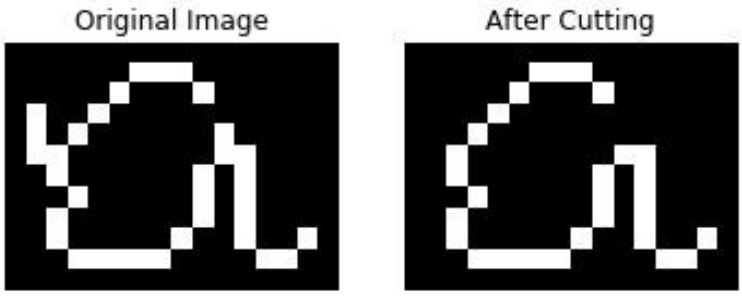
2、 腐蚀与膨胀模块

```python
import numpy as np
from scipy import signal#为了调用其二维卷积函数
def imdilate(I,SE):
    IM=np.array(I,dtype='uint8')        #转换输入图像数值类型
    IM_SE=signal.convolve(IM,SE,'same')#结构元SE与图像IM卷积
    m,n=IM.shape
    for i in range(m):
        for j in range(n):
            if IM_SE[i,j]!=0:#卷积结果不为0，则结果赋值为1（True），否则为0
                IM_SE[i,j]=1
            else:
                IM_SE[i,j]=0
    return np.array(IM_SE,dtype='bool')

def imerode(I,SE):
    IM=np.array(I,dtype='uint8')        #转换输入图像数值类型
    IM_SE=signal.convolve(IM,SE,'same')#结构元SE与图像IM卷积
    m,n=IM.shape
    s=np.sum(SE)    #统计结构元SE中“1”的个数
    for i in range(m):
        for j in range(n):
            if IM_SE[i,j]==s:
                #卷积结果=结构元“1”的个数时，说明结构元覆盖区域包含结构元“1”的范围
                IM_SE[i,j]=1
            else:
                IM_SE[i,j]=0
    return np.array(IM_SE,dtype='bool')
```

3、 剪裁模块

```python
import numpy as np
import ErodeDilate as ED
def Cut(I):
    I_copy=I
    #定义结构元B1~B8。
    #因为B1~B4中存在元素为"x"，因此定义B1_1~B4_1作为对应的翻转结构元
    B1=np.array([[0,0,0],[1,1,0],[0,0,0]])
    B1_1=np.array([[0,1,1],[0,0,1],[0,1,1]])
    B2=np.array([[0,1,0],[0,1,0],[0,0,0]])
    B2_1=np.array([[0,0,0],[1,0,1],[1,1,1]])
    B3=np.array([[0,0,0],[0,1,1],[0,0,0]])
    B3_1=np.array([[1,1,0],[1,0,0],[1,1,0]])
    B4=np.array([[0,0,0],[0,1,0],[0,1,0]])
    B4_1=np.array([[1,1,1],[1,0,1],[0,0,0]])
    B5=np.array([[1,0,0],[0,1,0],[0,0,0]])
    B6=np.array([[0,0,1],[0,1,0],[0,0,0]])
    B7=np.array([[0,0,0],[0,1,0],[0,0,1]])
    B8=np.array([[0,0,0],[0,1,0],[1,0,0]])
    SE=np.array([[1,1,1],[1,1,1],[1,1,1]])
    k=0#计数值
    N=3#进行N次操作
    while k<N:#N次结构元细化操作
        I1=~(ED.imerode(I,B1)&ED.imerode(~(I),B1_1))
        I2=~(ED.imerode(I,B2)&ED.imerode(~(I),B2_1))
        I3=~(ED.imerode(I,B3)&ED.imerode(~(I),B3_1))
        I4=~(ED.imerode(I,B4)&ED.imerode(~(I),B4_1))
        I5=~(ED.imerode(I,B5)&ED.imerode(~(I),np.ones((3,3))-B5))
        I6=~(ED.imerode(I,B6)&ED.imerode(~(I),np.ones((3,3))-B6))
        I7=~(ED.imerode(I,B7)&ED.imerode(~(I),np.ones((3,3))-B7))
        I8=~(ED.imerode(I,B8)&ED.imerode(~(I),np.ones((3,3))-B8))
        I=I&I1&I2&I3&I4&I5&I6&I7&I8
        k=k+1
    X1=I

    #每个结构元进行击中击不中变换提取端点集合
    I1=ED.imerode(I,B1)&ED.imerode(~(I),B1_1)
    I2=ED.imerode(I,B2)&ED.imerode(~(I),B2_1)
    I3=ED.imerode(I,B3)&ED.imerode(~(I),B3_1)
    I4=ED.imerode(I,B4)&ED.imerode(~(I),B4_1)
    I5=ED.imerode(I,B5)&ED.imerode(~(I),1-B5)
    I6=ED.imerode(I,B6)&ED.imerode(~(I),1-B6)
    I7=ED.imerode(I,B7)&ED.imerode(~(I),1-B7)
    I8=ED.imerode(I,B8)&ED.imerode(~(I),1-B8)
    X2=I1|I2|I3|I4|I5|I6|I7|I8
    X3=X2
    k=0
    while k<N:#进行N次在原图像I的限定下的膨胀
        X3=I_copy&ED.imdilate(X3,SE)
        k=k+1
    X4=X1|X3
    return np.array(X4,dtype='bool')
```

为了验证程序的正确性，对课本中的范例进行验证，程序文件为 cut_test.py。其验证结果如下：



4、 形态学骨架提取代码

```
'''形态学提取骨架并裁剪'''
import numpy as np
from PIL import Image
import rgb1gray    #自编写灰度化模块
import IM2BW       #自编写二值模块
import matplotlib.pyplot as plt
import ErodeDilate as ED # 自编写的腐蚀膨胀函数库
import CutImage as CI   #自编写裁剪模块

I=rgb1gray.rgb1gray(np.array(Image.open('fingerprint/smallfingerprint.jpg')))
I1=~(IM2BW.IM2BW(I))  #将原图转换为二值图像，并取反
SE=np.array([[0,1,0],[1,1,1],[0,1,0]])#结构元SE
m,n=I1.shape
sum_sk=I1&(~ED.imdilate(ED.imerode(I1,SE),SE))#二值图像I1减去自身的开操作
Ik=I1#Ik初始化为原二值图像I1（进行了0次腐蚀）
while np.any(Ik!=False):
    Ik1=ED.imdilate(ED.imerode(Ik,SE),SE)#对Ik进行开操作得到Ik1
    sk=Ik&(~Ik1)          #上一轮腐蚀操作的结果Ik减去其开操作Ik1
    sum_sk=sum_sk|sk     #将sk并集
    Ik=ED.imerode(Ik,SE) #再进行一次Ik的腐蚀操作
I_sk=sum_sk         #提取的骨架
I_cut=CI.Cut(I_sk)  #裁剪后的骨架
d=I_sk&(~I_cut)     #骨架裁剪掉的部分
plt.gray()
plt.figure(figsize=(16,8))
plt.subplot(141)
plt.imshow(I1)
plt.axis('off')
plt.title('(a)Binary Image(Inverted)')
plt.subplot(142)
plt.imshow(I_sk)
plt.axis('off')
plt.title('(b)Extracted skeleton')
plt.subplot(143)
plt.imshow(I_cut)
plt.axis('off')
plt.title('(c)After Prunning')
plt.subplot(144)
plt.imshow(d)
plt.axis('off')
plt.title('(d)Difference of (b) and (c)')
```

5、 基于距离变换的骨架提取

```
import numpy as np
from PIL import Image
import rgb1gray
import IM2BW  #二值化模块
import matplotlib.pyplot as plt
import ErodeDilate as ED # 自编写的腐蚀膨胀函数模块
import CutImage as CI  #裁剪模块

I=rgb1gray.rgb1gray(np.array(Image.open('fingerprint/smallfingerprint.jpg')))
II=~(IM2BW.IM2BW(I))      #将灰度图转换为二值图像并取反
SE=np.array([[0,1,0],[1,1,1],[0,1,0]])#结构元SE
I1=ED.imdilate(II,SE)&~II     #膨胀结果减去原二值图得到边界图像I1
A=np.array(I1,dtype='float64') #转换I1的数值类型便于计算距离
m,n=A.shape
for i in range(m):
    for j in range(n):
        if A[i,j]==0:
            A[i,j]=100#将图像背景值为100
```

```
#进行距离变换
#从上至下，从左至右
for i in range(1,m):
    for j in range(1,n-1):
        temp0=A[i,j]
        temp1=min([A[i,j-1]+3,temp0])
        temp2=min([A[i-1,j-1]+4,temp1])
        temp3=min([A[i-1,j]+3,temp2])
        temp4=min([A[i-1,j+1]+4,temp3])

        A[i,j]=temp4
#从下至上，从右至左
for i in range(m-2,-1,-1):
    for j in range(n-2,0,-1):
        temp0=A[i,j]
        temp1=min([A[i,j+1]+3,temp0])
        temp2=min([A[i+1,j-1]+4,temp1])
        temp3=min([A[i+1,j]+3,temp2])
        temp4=min([A[i+1,j+1]+4,temp3])
        A[i,j]=temp4
D=A
m,n=D.shape
```

```
D1=np.zeros((m,n))#存放距离变换的竖直方向差分梯度
D2=np.zeros((m,n))#存放距离变换的水平方向差分梯度
D3=np.zeros((m,n))#提取到的骨架（未转变为Bool型）
#计算水平与竖直方向的差分（梯度）
for i in range(1,m-1):
    for j in range(1,n-1):
        D1[i,j]=D[i+1,j]-D[i-1,j]#竖直方向
        D2[i,j]=D[i,j+1]-D[i,j-1]#水平方向
for i in range(1,m-1):
    for j in range(1,n-1):
        #判断局部极大值
        if ((D1[i,j]>0)&(D1[i+1,j]<=0))or((D2[i,j]>=0)&(D2[i,j+1]<0)):
            D3[i,j]=1
        else:
            D3[i,j]=0
for i in range(m):
    for j in range(n):
        #将局部极大值点与原图进行对比筛选最终的骨架
        if (D3[i,j]!=0)and(II[i,j]==True):
            D3[i,j]=1
        else:
            D3[i,j]=0
I_sk=np.array(D3,dtype='bool')#转换为Bool型，即提取的骨架
I_cut=CI.Cut(I_sk)      #裁剪后的骨架
d=I_sk&(~I_cut)         #裁剪掉的部分
```

效果展示：

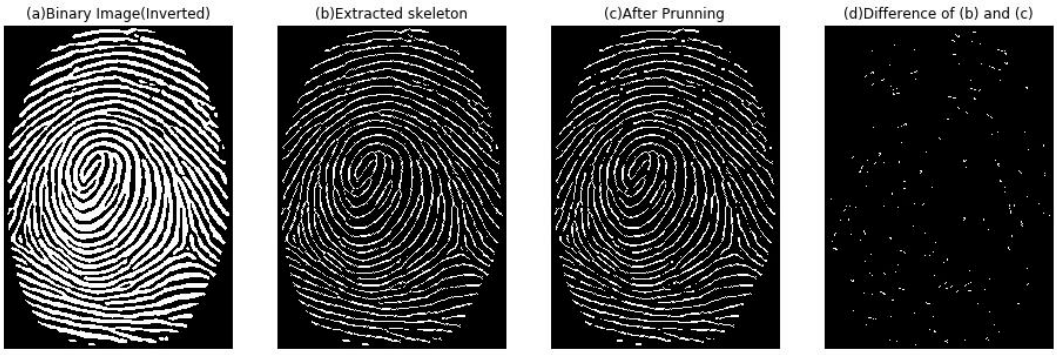
1、距离变换提取骨架

运行 dist1.py 得到实验图像指纹的骨架提取，如下图所示：

其中（a）为原始灰度图像，（b）为二值化并取反后的图像，（c）为边界图像，（d）为基于距离变换提取的骨架图像，（e）为裁剪后的图像，（f）为裁减掉的部分。根据实验结果，可以看到距离变换的提取结果较为良好，裁剪也起到了一定的作用，在裁剪后，一些骨架上的凸起或者毛刺被有效地去除了，结果比较符合要求。

2、形态学提取骨架

运行 xingtai_demo.py 得到实验图像，结果如下：



其中，（a）是取反后的二值图像，（b）是形态学算法提取得到的骨架图像，（c）是裁剪后的结果，（d）是裁减掉的部分。观察实验结果可以发现，骨架大致提取完整，但是该算法没有考虑骨架的连通性，因此在骨架中存在一些空洞。因此，这些应当从算法中进行改进。