Problem 8

Digital Image Processing

肖逸群 116033910112

Morpholigical Processing

问题描述

Implement the “Opening by reconstruction”, “Filling holes” and “Border clearing” operations on pages 681-683, and reproduce the results in Figure 9.29, Figures 9.31, and Figure 9.32. (The images fig9.29(a) and fig9.31(a) are provided in ftp address)

算法思想

本题涉及了形态学重建的两个核心，即测地膨胀和测地腐蚀。以此为基础，本题要求我们实现重建开操作、填充孔洞操作、边界清除操作。以下将介绍各种算法的基本思想：

1. **膨胀**
   令 F 表示标记图像，G 表示模板图像。则大小为 1 的标记图像关于模板的测地膨胀的定义如下：

$$D_G^{(1)}(F) = (F \oplus B) \cap G$$

   F 关于 G 的大小为 n 的测地膨胀定义为：

$$D_G^{(n)}(F) = D_G^{(1)}\left[D_G^{(n-1)}(F)\right]$$

2. **腐蚀**
   与膨胀的定义类似，大小为 1 的标记图像关于模板的测地腐蚀定义如下：

$$E_G^{(1)}(F) = (F \ominus B) \cup G$$

   F 关于 G 的大小为 n 的测地腐蚀定义为：

$$E_G^{(n)}(F) = E_G^{(1)}\left[E_G^{(n-1)}(F)\right]$$

由以上的定义可以看出，测地腐蚀和测地膨胀是关于集合的补集对偶的。有限数量图像的测地腐蚀和膨胀经过有限数量的迭代步骤后总会收敛，因为标记图像的扩散或收缩受模板约束。

1. **重建开操作**
   在形态学中，腐蚀会删除小的物体，而后续的膨胀试图恢复遗留物体的形状。然而，这种恢复的准确

性高度依赖于物体的形状和所用结构元的相似性。而本题用到的重建开操作可正确地恢复腐蚀后所保留物体的形状。
一幅图像 F 的大小为 n 的重建开操作定义为来自 F 的大小为 n 的腐蚀的 F 的重建，其公式如下：

$$O_R^{(n)}(F) = R_F^D[(F \ominus nB)]$$

其中，FΘnB 表示 B 对 F 的 n 次腐蚀。

2. **填充孔洞**
填充孔洞的操作需要我们先形成一幅标记图像，令 I(x, y) 代表一幅二值图像，并假定我们形成了一幅标记图像 F，除了在该图像的边界位置为 1 - I 之外，在其他位置均为 0，公式如下：

$$F(x, y) = \begin{cases} 1 - I(x, y), & (x, y) \text{在} I \text{的边界上} \\ 0, & \text{其他} \end{cases}$$

则在其基础上，填充孔洞的公式如下：

$$H = [R_{I^c}^D(F)]^c$$

3. **边界清除**
边界清除算法同样也需要一个标记图像，其标记图像的定义如下：

$$F(x, y) = \begin{cases} I(x, y), & (x, y) \text{位于} I \text{的边界上} \\ 0, & \text{其他} \end{cases}$$

此外，我们还需使用原始图像作为模板来进行边界清除操作。边界清除算法首先需要简单地提取接触到边界的物体，然后计算差值，公式如下：

$$X = I - R_I^D(F)$$

得到的 X 即是一幅没有边界接触的图像。

源码分析

- 实验过程如下：
  a. 读取原始图片，对其进行二值化操作，然后将其转为灰度图像。
  b. 对原始图像进行腐蚀操作，得到腐蚀图像。
  c. 对原始图像进行重建开操作，作为对比。
  d. 对腐蚀图像进行重建开操作。
  e. 显示图像。
  f. 求取原始图像的补集，得到补集图像。
  g. 获取原始图像的填充孔洞标记图像。
  h. 对原始图像进行填充孔洞操作，得到填充图像。
  i. 显示图像。
  j. 获取原始图像的边界清除标记图像。
  k. 获取边界清除后的图像。
  l. 显示图像。

```
1 def main():
```

```
    originalImage = Image.open('./resource/Fig0929(a)(text_image).tif').convert('1')

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    B = np.ones((51, 1))
    erosionImage = morphologyAlgorithm.erosionOperate(originalImage, B)
    plt.subplot(2, 2, 2)
    plt.imshow(erosionImage, cmap = plt.get_cmap('gray'))
    plt.title('Erosion')

    openingOriginalImage = morphologyAlgorithm.dilationOperate(originalImage, B)
    plt.subplot(2, 2, 3)
    plt.imshow(openingOriginalImage, cmap = plt.get_cmap('gray'))
    plt.title('Opening Original')

    B = np.ones((3, 3))
    reconstructionImage = morphologyAlgorithm.reconstructionOperate(originalImage, erosi
onImage, B)
    plt.subplot(2, 2, 4)
    plt.imshow(reconstructionImage, cmap = plt.get_cmap('gray'))
    plt.title('Reconstruction Opening')

    plt.show()

    plt.subplot(2, 2, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    plt.subplot(2, 2, 2)
    complementImage = morphologyAlgorithm.complementOperate(originalImage)
    plt.imshow(complementImage, cmap = plt.get_cmap('gray'))
    plt.title('Complement')

    B = np.ones((3, 3))
    plt.subplot(2, 2, 3)
    maskImage, fillingHolesImage = morphologyAlgorithm.holeFillingOperate(originalImage,
 B)
    plt.imshow(maskImage, cmap = plt.get_cmap('gray'))
    plt.title('Mark')

    plt.subplot(2, 2, 4)
    plt.imshow(fillingHolesImage, cmap = plt.get_cmap('gray'))
    plt.title('Filling Holes')

    plt.show()

    B = np.ones((3, 3))
    plt.subplot(1, 2, 1)
    maskImage2, borderClearingImage = morphologyAlgorithm.borderClearing(originalImage,
B)
    plt.imshow(maskImage2, cmap = plt.get_cmap('gray'))
    plt.title('Mark')

    plt.subplot(1, 2, 2)
    plt.imshow(borderClearingImage, cmap = plt.get_cmap('gray'))
    plt.title('Border Clearing')

    plt.show()
```

- 腐蚀操作：

```
def erosionOperate(originalImage, B):
    m, n = originalImage.size
    erosionImage = originalImage.copy()
    a, b = B.shape
    x2 = int(b / 2)
    y2 = int(a / 2)

    inputPixMatrix = originalImage.load()
    outputPixMatrix = erosionImage.load()

    for x in range(m):
        for y in range(n):
            if inputPixMatrix[x, y] != 0:
                flag = False
                for i in range(-x2, x2 + 1):
                    for j in range(-y2, y2 + 1):
                        if x + i < 0 or x + i >= m or y + j < 0 or y + j >= n:
                            flag = True
                            break
                        if B[j + y2][i + x2] == 1 and inputPixMatrix[x + i, y + j] == 0:
                                flag = True
                                break
                    if flag == True:
                        outputPixMatrix[x, y] = 0
                        break

    return erosionImage
```

- 膨胀操作：

```
def dilationOperate(originalImage, B):
    m, n = originalImage.size
    dilationImage = originalImage.copy()
    a, b = B.shape
    x2 = int(b / 2)
    y2 = int(a / 2)

    inputPixMatrix = originalImage.load()
    outputPixMatrix = dilationImage.load()

    for x in range(m):
        for y in range(n):
            if inputPixMatrix[x, y] == 0:
                flag = False
                for i in range(-x2, x2 + 1):
                    for j in range(-y2, y2 + 1):
                        if x + i < 0 or x + i >= m or y + j < 0 or y + j >= n:
                            continue
                        if B[j + y2][i + x2] == 1 and inputPixMatrix[x + i, y + j] != 0:
                            flag = True
                            break
                    if flag == True:
                        outputPixMatrix[x, y] = 255
                        break

    return dilationImage
```

- 重建开操作：

```
def reconstructionOperate(originalImage, erosionImage, B):
    m, n = erosionImage.size
    reconstructionImage = erosionImage.copy()

    erosionPixs = erosionImage.load()
    originalPixs = originalImage.load()
    outputPixMatrix = reconstructionImage.load()

    a, b = B.shape
    x2 = int(b / 2)
    y2 = int(a / 2)

    while 1:
        flag1 = True
        for x in range(m):
            for y in range(n):
                if originalPixs[x, y] == 0:
                    if erosionPixs[x, y] != 0:
                        flag1 = False
                    outputPixMatrix[x, y] = 0
                    continue
                if erosionPixs[x, y] == 0:
                    flag2 = False
                    for i in range(-x2, x2 + 1):
                        for j in range(-y2, y2 + 1):
                            if x + i < 0 or x + i >= m or y + j < 0 or y + j >= n:
                                continue
                            if B[j + y2]
[i + x2] == 1 and erosionPixs[x + i, y + j] != 0:
                                flag2 = True
                                break
                    if flag2 == True:
                        outputPixMatrix[x, y] = 255
                        flag1 = False
                        break
        if flag1 == True:
            break
        erosionImage = reconstructionImage.copy()
        erosionPixs = erosionImage.load()

    return reconstructionImage
```

- 边界提取操作：

```
def borderClearing(originalImage, B):
    m, n = originalImage.size
    markImage = originalImage.copy()
    borderClearingImage = originalImage.copy()

    originalPixs = originalImage.load()
    borderClearingPixs = borderClearingImage.load()
    markPixs = markImage.load()

    for i in range(m):
        for j in range(n):
            if i == 0 or j == 0 or i == m - 1 or j == n - 1:
                markPixs[i, j] = originalPixs[i, j]
            else:
                markPixs[i, j] = 0

    borderImage = reconstructionOperate(originalImage, markImage, B)
```

```
    borderPixs = borderImage.load()
    for i in range(m):
        for j in range(n):
            borderClearingPixs[i, j] = borderClearingPixs[i, j] - borderPixs[i, j]

    return markImage, borderClearingImage
```

- 求取图像补集操作：

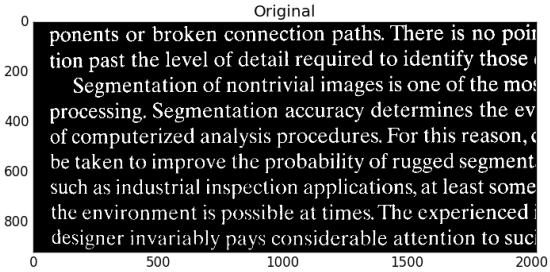
```
def complementOperate(originalImage):
    complementImage = originalImage.copy()
    complementPixs = complementImage.load()

    m, n = complementImage.size
    for i in range(m):
        for j in range(n):
            complementPixs[i, j] = 255 - complementPixs[i, j]

    return complementImage
```

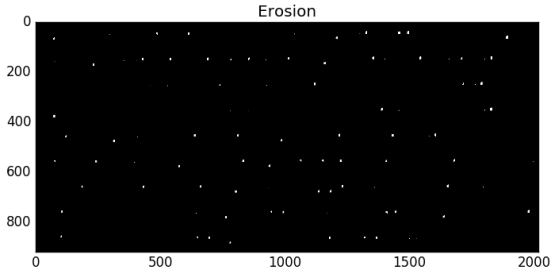
- 填充孔洞操作：

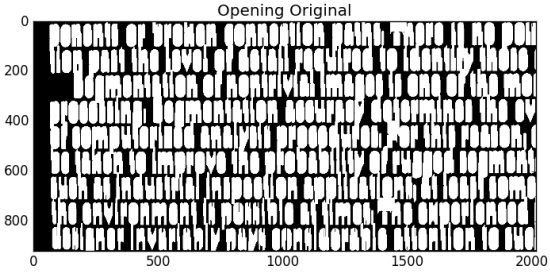
```
def holeFillingOperate(originalImage, B):
    m, n = originalImage.size
    markImage = originalImage.copy()
    maskImage = originalImage.copy()

    inputPixMatrix = originalImage.load()
    markPixs = markImage.load()
    maskPixs = maskImage.load()

    for x in range(m):
        for y in range(n):
            if x == 0 or y == 0 or x == m - 1 or y == n - 1:
                markPixs[x, y] = 255 - inputPixMatrix[x, y]
            else:
                markPixs[x, y] = 0
            maskPixs[x, y] = 255 - inputPixMatrix[x, y]

    fillingHolesImage = reconstructionOperate(maskImage, markImage, B)
    outputPixMatrix = fillingHolesImage.load()

    for x in range(m):
        for y in range(n):
            outputPixMatrix[x, y] = 255 - outputPixMatrix[x, y]

    return markImage, fillingHolesImage
```

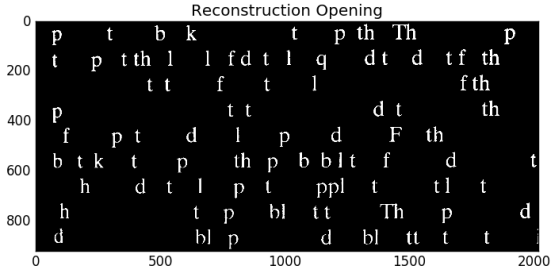
实验结果

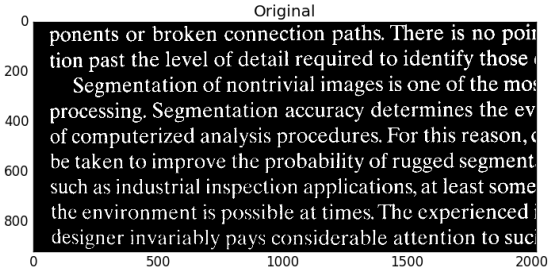
下列图片分别为课本里 9.29, 9.30, 9.31 的实验结果：

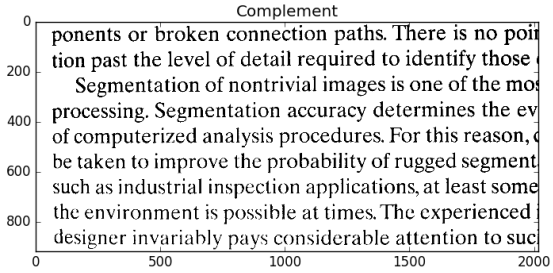

Original

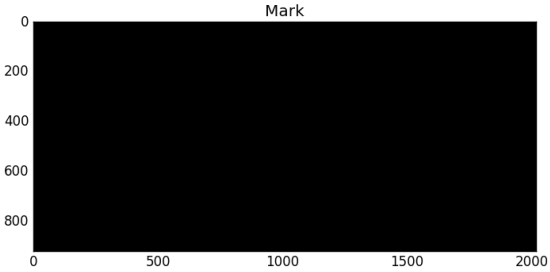

Erosion

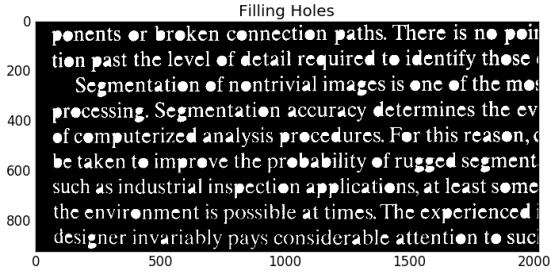

Opening Original

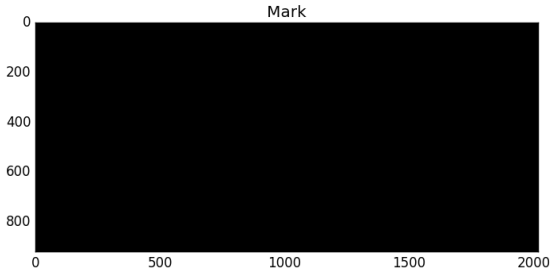

Reconstruction Opening

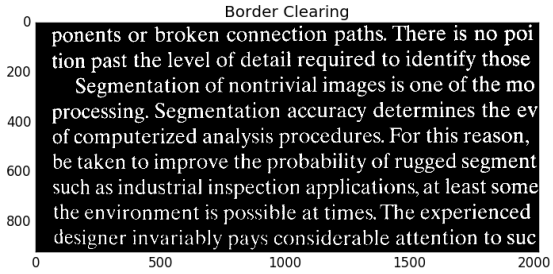

Original


Complement


Mark


Filling Holes


Mark
0
200
400
600
800
0
500
1000
1500
2000


Border Clearing
0
200
400
600
800
0
500
1000
1500
2000
ponents or broken connection paths. There is no poi
tion past the level of detail required to identify those
Segmentation of nontrivial images is one of the mo
processing. Segmentation accuracy determines the ev
of computerized analysis procedures. For this reason,
be taken to improve the probability of rugged segment
such as industrial inspection applications, at least some
the environment is possible at times. The experienced
designer invariably pays considerable attention to suc