

**“数字图像处理基础”实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 题目 | 实验九：形态学处理 | | |
| 姓名 | 22211374 | 学号 | 钟军凯 |
| 指导教师 | 彭亚辉 | | |
| 日期 | 2025年4月24日 | | |

**电子信息工程学院**

目 录

[1. 实验目的分析 3](#_Toc191473896)

[2. 实验方案设计 3](#_Toc191473897)

[3. 实验结果 4](#_Toc191473898)

[4. 分析与讨论 8](#_Toc191473899)

[5. 问题发现与探究 9](#_Toc191473900)

[6. 实验总结 1](#_Toc191473901)1

[指导老师评价 2](#_Toc191473902)1

1. 实验目的分析

本实验旨在通过Python编程实现对二值图像的形态学处理及分析，具体目标包括：掌握二值图像的膨胀、腐蚀、开运算、闭运算、梯度运算等基础形态学算法的原理与实现；理解并掌握二值图像中连通域分析的方法，能够通过标记连通域完成目标区域的分割；最终通过编程实践，提升对图像形态学处理技术的应用能力。

1. 实验方案设计
   1. **基础形态学处理**

对于腐蚀操作，定义一个n\*n的结构元，元素全是1，n为奇数。遍历图像中的所有n\*n区域，将结构元与n\*n区域进行元素级乘法，也就是对应位置元素相乘，然后取计算结果的最小值，如果最小值是0，则n\*n区域中心像素值置为0，如果最小值为255，则n\*n区域中心像素值置为255，也就是将元素级乘法的各个结果中的最小值赋予n\*n区域的中心。图像的边缘区域可以用边缘扩充的方式进行处理，扩充之后采用零填充。

对于膨胀操作，只需将腐蚀中的“将元素级乘法的各个结果中的最小值赋予n\*n区域的中心”中的“最小值”改成“最大值”即可。

对于开运算和闭运算，其实就是腐蚀和膨胀的组合，只是顺序不一样。开运算是先腐蚀后膨胀，闭运算是先膨胀后腐蚀，所以将腐蚀和膨胀两种操作实现了，开运算和闭运算就可以实现了。

对于梯度运算，这里我理解为边缘提取，可以先对图像进行腐蚀处理，然后原图像减去腐蚀后的图像，即可得到图像中前景的边缘。这里的腐蚀操作的结构元使用3\*3的，这样就可以让前景部分被腐蚀的部分最少，最终得到的边缘就不会很粗。在进行边缘提取之前，可先使用开运算，除去图像中的大部分噪声，为了能够去除更多的噪声，腐蚀结构元和膨胀结构元的大小均设置为5\*5。

* 1. **8-邻域连通域检测**

这里进行的是8-邻域连通域检测，此处给出我的想法：

设置一个标签变量，记录当前的标签到多少了。设置一个大小为（rows，columns，2）的数组，其中，通道一为图像像素值，通道二为该像素点的标签。

进行第一次遍历，遍历整幅图像，如果是前景点，搜索左、左上、上、右上四个方向，如果前景点周围没有前景点，则这一个前景点打上新标签；如果有前景点但是都没有标签，则遍历到的前景点和搜索到的前景点打上同样的标签；如果有前景点且有标签，则需要设置等价标签数组。在设置等价标签数组的时候，需要进行标签比较，参与比较的标签必须该像素点的标签的的最小等价标签，将比较得到的最小的标签赋予搜索到的没有标签的前景点，对于搜索到的有标签的前景点，则是将比较得到的最小标签赋予等价标签数组中，以这些前景点的标签的最小等价标签为索引的元素。

简要说明一下等价标签数组，数组的索引为当前标签，元素的值为对应标签的等价标签。这时会出现这样的情况，A的等价标签是B，然后发现B的等价标签是C，然后就会出现链式的等价关系，直到一个标签的等价标签是它本身，此时的标签就是最小等价标签（因为标签是从0开始的，一个新的标签只会与旧的标签等价，也就是大的标签只会与比它小的标签等价），对于这条链上的每一个标签都是。在初始化等价标签数组的时候，可以使用numpy.arange()函数，这样每一个标签的初始等价标签都是它本身。

进行第二次遍历，利用等价标签数组，找到每一个前景点当前标签的最小等价标签，然后改变前景点当前标签。

在进行连通域检测之前，需要先使用开运算去除噪声，为了能够去除更多的噪声，腐蚀结构元和膨胀结构元的大小均设置为5\*5。

最后将不同连通域设置为不同颜色进行显示。

* 1. **GUI设计**

腐蚀、膨胀、开运算、闭运算、梯度计算和连通域检测分别使用一个按键来控制，使用滑动条来调整腐蚀结构元和膨胀结构元的大小。

1. 实验结果

膨胀处理结果如图1所示。

|  |  |
| --- | --- |
|  |  |
|  |  |

**图 1**：膨胀处理结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）膨胀结构元大小为3\*3；（3）膨胀结构元大小为5\*5；（4）膨胀结构元大小为7\*7

腐蚀处理结果如图2所示。

|  |  |
| --- | --- |
|  |  |
|  |  |

**图 2**：腐蚀处理结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）腐蚀结构元大小为3\*3；（3）腐蚀结构元大小为5\*5；（4）腐蚀结构元大小为7\*7

开运算结果如图3所示。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**图 3**：开运算结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）腐蚀结构元大小3\*3，膨胀结构元大小3\*3；（3）腐蚀结构元大小5\*5，膨胀结构元大小5\*5；（4）腐蚀结构元大小7\*7，膨胀结构元大小7\*7；（5）腐蚀结构元大小7\*7，膨胀结构元大小3\*3；（6）腐蚀结构元大小3\*3，膨胀结构元大小7\*7

闭运算结果如图4所示。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**图 4**：闭运算结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）腐蚀结构元大小3\*3，膨胀结构元大小3\*3；（3）腐蚀结构元大小5\*5，膨胀结构元大小5\*5；（4）腐蚀结构元大小7\*7，膨胀结构元大小7\*7；（5）腐蚀结构元大小7\*7，膨胀结构元大小3\*3；（6）腐蚀结构元大小3\*3，膨胀结构元大小7\*7

梯度计算（边缘提取）结果如图5所示。

|  |  |
| --- | --- |
|  |  |

**图 5**：梯度计算结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）梯度计算结果

1. 邻域连通域检测结果如图6所示。

|  |  |
| --- | --- |
|  |  |

**图 6**：8-邻域连通域检测结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）8-邻域连通域

1. 分析与讨论

对于膨胀，由图1可知，随着膨胀结构元大小的增大，前景膨胀的程度越来越大，即前景所占的部分越来越多；

对于腐蚀，由图2可知，随着腐蚀结构元大小的增大，前景被腐蚀的程度越来越大，即前景所占的部分越来越少。

对于开运算，由图3可知，当腐蚀结构元和膨胀结构元相等的时候，结构元大小增大，图像中前景部分减少，这是因为开运算是先腐蚀后膨胀，被腐蚀的部分不会因为膨胀而重新出现，所以结构元大小增大之后，会先腐蚀掉更多的前景，然后膨胀剩下的前景。当腐蚀结构元和膨胀结构元大小不等的时候，当腐蚀结构元大小大于膨胀结构元的时候，图像整体呈现出被腐蚀的效果。当腐蚀结构元大小小于膨胀结构元的时候，图像整体呈现处膨胀的效果。

对于闭运算，由图4可知，当腐蚀结构元和膨胀结构元相等的时候，结构元大小增大，图像中前景部分增加，这是因为闭运算是先膨胀后腐蚀，先膨胀使得图像中前景部分增多，某些背景会被填充，此时在进行腐蚀，那些被填充的部分就不会恢复了，所以导致整体呈现出膨胀的效果。当腐蚀结构元和膨胀结构元大小不等的时候，当腐蚀结构元大小大于膨胀结构元的时候，图像整体呈现出被腐蚀的效果。当腐蚀结构元大小小于膨胀结构元的时候，图像整体呈现处膨胀的效果。

由图5和图6可知，图像的边缘提取和连通域检测都成功进行。

1. 问题发现与探究
   1. **问题一**
      1. **问题描述**

因为腐蚀操作和膨胀的操作都涉及到了类似于滑动窗口的操作，但是自己最初使用两层for循环实现的，但是我觉得这样会影响程序运行速度，而且也没有充分利用numpy的特性。

* + 1. **问题探究**

通过查阅资料得知，numpy中有滑动窗口的模块，需要使用from numpy.lib.stride\_tricks import sliding\_window\_view来包含进代码。使用windows = sliding\_window\_view(image\_array, (size, size))就可以创建一个关于滑动窗口的矩阵，也就是这个矩阵的元素是大小为size\*size的窗口，矩阵的大小更根据image\_array和size决定。这个时候，就可以用到numpy的特性进行高效的计算了。

* 1. **问题二**
     1. **问题描述**

如图7所示，这是连通域检测代码最初的结果，最初是考虑使用不同的灰度值来标记不同邻域的。由图7可知，本应是属于同一块连通域的两个区域，用了两个不同的灰度标记。

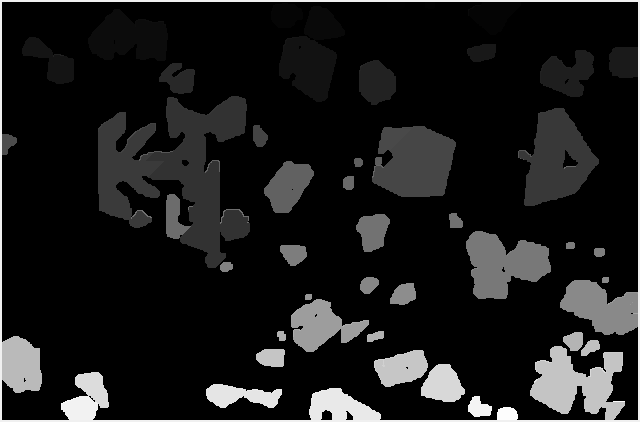


图 7

* + 1. **问题解决**

在最初的时候，在设置等价标签数组这里，通过比较得到最小标签值这一步，我没有使用当前标签的最小等价标签来参与比较，而是直接使用当前标签来进行比较，导致一条等价链中某个位置断开了，所以出现了如图7所示的情况，经过改进之后，得到了图8。

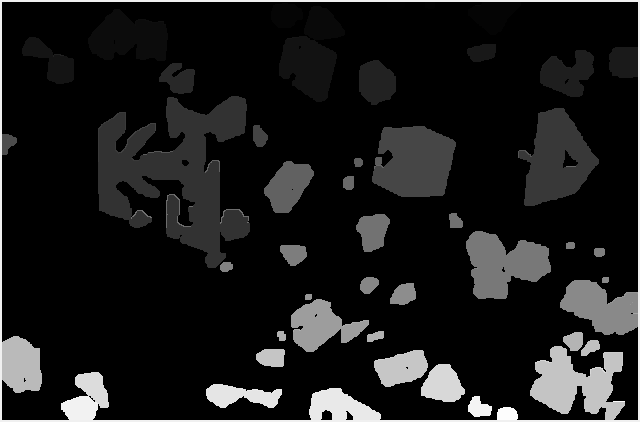


图 8

观察图8可知，在一些连通域的一些边缘处，出现了连通域判断错误的情况。通过代码调试，我发现有一些不是前景的像素点也被打上了标签，所以导致了如图8所示情况的发生。不是前景的像素点也被打上标签，说明是标签赋值的代码出了问题，经过自己的检查也发现了问题出在哪里，修改完之后，再把连通域的标记改为彩色，就得到了图6右侧图片的效果。问题解决。

1. 实验总结

本次实验通过Python编程实现了二值图像的形态学处理及连通域检测，不仅完成了膨胀、腐蚀、开闭运算等基础操作以优化图像结构，还独立实现了连通域分析算法，标记出了图像中的独立的连通域。整个实验使得我对形态学理论有了更加深入的理解，连通域检测算法的实现过程也强化了我的代码实现与问题分析能力，为后续复杂图像任务积累了关键技术经验。

附录：

【仿真程序】

|  |
| --- |
| import tkinter as tk  from tkinter import ttk  from tkinter import filedialog as fd  from PIL import Image  from PIL import ImageTk  import numpy as np  from matplotlib import pyplot as plt  from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk  from matplotlib.figure import Figure  import struct  import math  from numpy.lib.stride\_tricks import sliding\_window\_view  '''  @brief 获取图像数组  @param image: PIL的Image类对象  @return height: 图像的高（行数）  @return width: 图像的宽（列数）  @return image\_array: 图像对应的数组  '''  def get\_image\_data (image):  (width, height) = image.size # 获取图像的宽和高  image\_array = np.array(image) # 获取图像的像素数据并转化为数组  return height, width, image\_array  '''  @brief 读取raw文件  @param file\_name: raw文件路径  @return raw\_array: raw文件图像数组  '''  def read\_raw (file\_name):  # 获取raw文件表示的图片的数据  raw\_file = open(file\_name, "rb") # 打开文件，以只读、二进制的方式打开  raw\_width = struct.unpack("i", raw\_file.read(4))[0] # 获取raw文件表示的图片的宽度  raw\_height = struct.unpack("i", raw\_file.read(4))[0] # 获取raw文件表示的图片的高度  raw\_data = struct.unpack(f"{raw\_width \* raw\_height}B", raw\_file.read()) # 获取raw文件表示的图片的数组  raw\_file.close() # 关闭文件  # 将获取到的数组转换为二维数组  raw\_array = np.array(raw\_data).reshape((raw\_height, raw\_width))  return raw\_array  '''  @brief 显示变换后的图像  @param image\_array: 图像数组  '''  def show\_transform\_image (image\_array):  global transform\_image\_tk  # 转换  image = Image.fromarray(np.uint8(image\_array))  transform\_image\_tk = ImageTk.PhotoImage(image)  # 显示分割后的图片  transform\_image\_label.config(image = transform\_image\_tk)  def morphology\_process (image\_array, method, dilation\_se\_size = 3, erosion\_se\_size = 3):  # 获取图像行数和列数  rows = image\_array.shape[0]  columns = image\_array.shape[1]  # 膨胀或者腐蚀  if method == "dilation" or method == "erosion":  # 获取结构元大小  if method == "dilation":  se\_size = dilation\_se\_size  elif method == "erosion":  se\_size = erosion\_se\_size  # 边缘填充，采用零填充  new\_rows = rows + se\_size - 1  new\_columns = columns + se\_size - 1  expand\_array = np.zeros((new\_rows, new\_columns))  offset = int((se\_size - 1) / 2)  expand\_array[offset : rows + offset, offset : columns + offset] = np.copy(image\_array)  # 形态学处理  se = np.ones((se\_size, se\_size))  windows = sliding\_window\_view(expand\_array, (se\_size, se\_size))  if method == "dilation":  result = np.max(np.multiply(windows[:, :], se), axis = (-2, -1))  elif method == "erosion":  result = np.min(np.multiply(windows[:, :], se), axis = (-2, -1))  # 开运算或者闭运算  elif method == "opening" or method == "closing":  # 获取两个结构元大小  if method == "opening":  se\_size\_1 = erosion\_se\_size  se\_size\_2 = dilation\_se\_size  elif method == "closing":  se\_size\_1 = dilation\_se\_size  se\_size\_2 = erosion\_se\_size  # 第一次边缘填充，采用零填充  new\_rows = rows + se\_size\_1 - 1  new\_columns = columns + se\_size\_1 - 1  expand\_array\_1 = np.zeros((new\_rows, new\_columns))  offset = int((se\_size\_1 - 1) / 2)  expand\_array\_1[offset : rows + offset, offset : columns + offset] = np.copy(image\_array)  # 第一次形态学处理  se\_1 = np.ones((se\_size\_1, se\_size\_1))  windows\_1 = sliding\_window\_view(expand\_array\_1, (se\_size\_1, se\_size\_1))  if method == "opening": # 先进行腐蚀  result = np.min(np.multiply(windows\_1[:, :], se\_1), axis = (-2, -1))  elif method == "closing": # 先进行膨胀  result = np.max(np.multiply(windows\_1[:, :], se\_1), axis = (-2, -1))  # 第二次边缘填充，采用零填充  new\_rows = rows + se\_size\_2 - 1  new\_columns = columns + se\_size\_2 - 1  expand\_array\_2 = np.zeros((new\_rows, new\_columns))  offset = int((se\_size\_2 - 1) / 2)  expand\_array\_2[offset : rows + offset, offset : columns + offset] = np.copy(result)  # 第二次形态学处理  se\_2 = np.ones((se\_size\_2, se\_size\_2))  windows\_2 = sliding\_window\_view(expand\_array\_2, (se\_size\_2, se\_size\_2))  if method == "opening": # 后进行膨胀  result = np.max(np.multiply(windows\_2[:, :], se\_2), axis = (-2, -1))  elif method == "closing": # 后进行腐蚀  result = np.min(np.multiply(windows\_2[:, :], se\_2), axis = (-2, -1))  return result  def morphology\_edge (image\_array):  image\_array = morphology\_process(image\_array, "opening", dilation\_se\_size = 5, erosion\_se\_size = 5)  transform\_image\_array = morphology\_process(image\_array, "erosion")  result = image\_array - transform\_image\_array  return result  def connected\_analysis (image\_array):  image\_array = morphology\_process(image\_array, "opening", dilation\_se\_size = 5, erosion\_se\_size = 5)  label\_value = 0  values = np.zeros(4, dtype = np.int32)  labels = np.zeros(4, dtype = np.int32)  equ\_label = np.arange(image\_array.shape[0] \* image\_array.shape[1])  expand\_array = np.pad(image\_array, pad\_width = 1, mode = 'constant', constant\_values = 0)  rows = expand\_array.shape[0]  columns = expand\_array.shape[1]  array\_with\_label = np.zeros((rows, columns, 2), dtype = np.int32)  array\_with\_label[:, :, 0] = expand\_array  array\_with\_label[:, :, 1] = -1 # 表示未打标签  for i in range(1, rows - 1):  for j in range(1, columns - 1):  if array\_with\_label[i, j, 0] == 255:  values[0] = array\_with\_label[i, j - 1, 0] # 左  values[1] = array\_with\_label[i - 1, j - 1, 0] # 左上  values[2] = array\_with\_label[i - 1, j, 0] # 上  values[3] = array\_with\_label[i - 1, j + 1, 0] # 右上  labels[0] = array\_with\_label[i, j - 1, 1]  labels[1] = array\_with\_label[i - 1, j - 1, 1]  labels[2] = array\_with\_label[i - 1, j, 1]  labels[3] = array\_with\_label[i - 1, j + 1, 1]  if np.count\_nonzero(values == 255) == 0: # 周围没有前景点  array\_with\_label[i, j, 1] = label\_value  label\_value += 1  else: # 周围有前景点  if np.count\_nonzero(labels != -1) == 0: # 前景点都没有标签  array\_with\_label[i, j, 1] = label\_value  array\_with\_label[i, j - 1, 1] = label\_value if values[0] == 255 else (-1)  array\_with\_label[i - 1, j - 1, 1] = label\_value if values[1] == 255 else (-1)  array\_with\_label[i - 1, j, 1] = label\_value if values[2] == 255 else (-1)  array\_with\_label[i - 1, j + 1, 1] = label\_value if values[3] == 255 else (-1)  label\_value += 1  else: # 有的前景点有标签  labels\_1 = labels[labels != -1]  for k in range(labels\_1.shape[0]):  # 寻找最小等价标签  while equ\_label[labels\_1[k]] != labels\_1[k]:  labels\_1[k] = equ\_label[labels\_1[k]]  min\_label = np.min(labels\_1)    # 打标签  array\_with\_label[i, j, 1] = min\_label  array\_with\_label[i, j - 1, 1] = min\_label if (labels[0] == -1 and values[0] == 255) else labels[0]  array\_with\_label[i - 1, j - 1, 1] = min\_label if (labels[1] == -1 and values[1] == 255) else labels[1]  array\_with\_label[i - 1, j, 1] = min\_label if (labels[2] == -1 and values[2] == 255) else labels[2]  array\_with\_label[i - 1, j + 1, 1] = min\_label if (labels[3] == -1 and values[3] == 255) else labels[3]  # 修改等价标签  equ\_label[labels\_1] = min\_label    # 第二次遍历  for i in range(1, rows - 1):  for j in range(1, columns - 1):  if array\_with\_label[i, j, 0] == 255:  # 寻找最小等价标签  while equ\_label[array\_with\_label[i, j, 1]] != array\_with\_label[i, j, 1]:  array\_with\_label[i, j, 1] = equ\_label[array\_with\_label[i, j, 1]]  # 彩色标注  temp\_array = np.zeros((rows, columns, 3), dtype = np.int32)  rgb\_array = np.random.randint(0, 256, (np.max(array\_with\_label[:, :, 1]) + 1, 3))  for i in range(np.max(array\_with\_label[:, :, 1]) + 1):  temp\_array[array\_with\_label[:, :, 1] == i] = rgb\_array[i]  result = temp\_array[1 : rows - 1, 1 : columns - 1]  show\_transform\_image(result)  transform\_image\_tip.config(text = f"8-领域连通域检测，有{label\_value}个")  def image\_morphology (method):  global morphology\_method, dilation\_se\_size, erosion\_se\_size, image\_array  morphology\_method = method  transform\_image\_array = morphology\_process(image\_array, method = morphology\_method, dilation\_se\_size = dilation\_se\_size, erosion\_se\_size = erosion\_se\_size)    show\_transform\_image(transform\_image\_array)  scale\_frame.grid(row = 3, column = 1)  if method == "dilation":  dilation\_se\_size\_tip.grid(row = 0, column = 0)  dilation\_se\_size\_scale.grid(row = 0, column = 1)  erosion\_se\_size\_tip.grid\_forget()  erosion\_se\_size\_scale.grid\_forget()  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 结构元大小为{dilation\_se\_size}")  elif method == "erosion":  dilation\_se\_size\_tip.grid\_forget()  dilation\_se\_size\_scale.grid\_forget()  erosion\_se\_size\_tip.grid(row = 1, column = 0)  erosion\_se\_size\_scale.grid(row = 1, column = 1)  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 结构元大小为{erosion\_se\_size}")  elif method == "opening" or method == "closing":  dilation\_se\_size\_tip.grid(row = 0, column = 0)  dilation\_se\_size\_scale.grid(row = 0, column = 1)  erosion\_se\_size\_tip.grid(row = 1, column = 0)  erosion\_se\_size\_scale.grid(row = 1, column = 1)  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 腐蚀结构元大小为{erosion\_se\_size}, 膨胀结构元大小为{dilation\_se\_size}")  def get\_image\_morphology\_edge ():  global image\_array  edge = morphology\_edge(image\_array)  show\_transform\_image(edge)  scale\_frame.grid\_forget()  transform\_image\_tip.config(text = "边界")  def update\_dilation\_se\_size (se\_size):  global morphology\_method, dilation\_se\_size, erosion\_se\_size, image\_array  dilation\_se\_size = int(se\_size)  transform\_image\_array = morphology\_process(image\_array, method = morphology\_method, dilation\_se\_size = dilation\_se\_size, erosion\_se\_size = erosion\_se\_size)  show\_transform\_image(transform\_image\_array)  if morphology\_method == "dilation":  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 结构元大小为{dilation\_se\_size}")  elif morphology\_method == "opening" or morphology\_method == "closing":  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 腐蚀结构元大小为{erosion\_se\_size}, 膨胀结构元大小为{dilation\_se\_size}")  def update\_erosion\_se\_size (se\_size):  global morphology\_method, dilation\_se\_size, erosion\_se\_size, image\_array  erosion\_se\_size = int(se\_size)  transform\_image\_array = morphology\_process(image\_array, method = morphology\_method, dilation\_se\_size = dilation\_se\_size, erosion\_se\_size = erosion\_se\_size)  show\_transform\_image(transform\_image\_array)  if morphology\_method == "erosion":  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 结构元大小为{erosion\_se\_size}")  elif morphology\_method == "opening" or morphology\_method == "closing":  transform\_image\_tip.config(text = f"{morphology\_method}处理后的图像, 腐蚀结构元大小为{erosion\_se\_size}, 膨胀结构元大小为{dilation\_se\_size}")  def file\_operation ():  global origin\_image\_tk, image\_array  # 获取文件路径  file\_path = fd.askopenfilename()  # 获取文件格式  file\_format = []  for i in reversed(range(len(file\_path))):  if file\_path[i] == '.': # 由于只需要文件格式，所以遇到'.'就退出  break  file\_format.append(file\_path[i])  file\_format.reverse() # 列表反转  file\_format\_str = "".join(file\_format) # 转换为字符串  # 判断文件格式并得到Image对象和图像数组  if file\_format\_str == "raw":  image\_array = read\_raw(file\_path)  image = Image.fromarray(image\_array)  else:  image = Image.open(file\_path).convert('L')  \_, \_, image\_array = get\_image\_data(image)  # 转换为tkinter能解析的PhotoImage对象  origin\_image\_tk = ImageTk.PhotoImage(image)  # 显示图像  origin\_image\_label.config(image = origin\_image\_tk)  origin\_image\_tip.config(text = "原图像")  transform\_image\_label.config(image = "")  transform\_image\_tip.config(text = "")  scale\_frame.grid\_forget()  if \_\_name\_\_ == '\_\_main\_\_':  # 初始化参数  dilation\_se\_size = 3  erosion\_se\_size = 3  morphology\_method = None  # 创建基本界面  root = tk.Tk()  root.title("数字图像处理实验八：基于边缘的图像分割") # 设置界面标题  root.grid()  root.grid\_columnconfigure(0, weight = 1) # root的第一列会适应界面大小的改变  root.grid\_rowconfigure(0, weight = 1) # root的第一行会适应界面大小的改变  # 创建一个画布，画布可以被滚动条控制  canvas = tk.Canvas(root)  canvas.grid(row = 0, column = 0, sticky = "nsew") # canvas会填充整个界面  # 创建滚动条  scrollbar\_1 = ttk.Scrollbar(root, orient = "vertical", command = canvas.yview)  scrollbar\_1.grid(row = 0, column = 1, sticky = "ns") # 纵轴方向填充  scrollbar\_2 = ttk.Scrollbar(root, orient = "horizontal", command = canvas.xview)  scrollbar\_2.grid(row = 1, column = 0, sticky = "ew") # 横轴方向填充  # canvas与滚动条关联  canvas.config(yscrollcommand = scrollbar\_1.set)  canvas.config(xscrollcommand = scrollbar\_2.set)  # 创建Frame容器，用于存放各种子容器  frame = ttk.Frame(canvas, padding = 10)  frame.grid(row = 0, column = 0)  canvas.create\_window((0, 0), window = frame, anchor = "nw") # 将frame嵌入canvas  frame.bind("<Configure>", lambda event: canvas.configure(scrollregion = canvas.bbox("all"))) # 使滚动条适应frame  # 创建存放按键的Frame容器  button\_frame = ttk.Frame(frame, padding = 10)  button\_frame.grid(row = 0, column = 0)  # 创建“打开文件”按键  open\_file\_button = ttk.Button(button\_frame, text = "打开文件", command = file\_operation)  open\_file\_button.grid(row = 0, column = 0, columnspan = 4)  # 创建功能按键  dilation\_button = ttk.Button(button\_frame, text = "膨胀", command = lambda: image\_morphology("dilation"))  dilation\_button.grid(row = 1, column = 0)  erosion\_button = ttk.Button(button\_frame, text = "腐蚀", command = lambda: image\_morphology("erosion"))  erosion\_button.grid(row = 1, column = 1)  opening\_button = ttk.Button(button\_frame, text = "开运算", command = lambda: image\_morphology("opening"))  opening\_button.grid(row = 1, column = 2)  closing\_button = ttk.Button(button\_frame, text = "闭运算", command = lambda: image\_morphology("closing"))  closing\_button.grid(row = 1, column = 3)  edge\_button = ttk.Button(button\_frame, text = "边界获取", command = get\_image\_morphology\_edge)  edge\_button.grid(row = 2, column = 0, columnspan = 4)  connected\_analysis\_button = ttk.Button(button\_frame, text = "8-邻域连通域标记", command = lambda: connected\_analysis(image\_array))  connected\_analysis\_button.grid(row = 3, column = 0, columnspan = 4)  # 创建“退出”按键  quit\_button = ttk.Button(button\_frame, text = "退出", command = root.destroy)  quit\_button.grid(row = 4, column = 0, columnspan = 4)  # 创建图像标签  origin\_image\_label = ttk.Label(frame)  origin\_image\_label.grid(row = 1, column = 0)  transform\_image\_label = ttk.Label(frame)  transform\_image\_label.grid(row = 1, column = 1)  # 创建图像标注  origin\_image\_tip = ttk.Label(frame)  origin\_image\_tip.grid(row = 2, column = 0)  transform\_image\_tip = ttk.Label(frame)  transform\_image\_tip.grid(row = 2, column = 1)  # 创建存放滑动条的Frame容器  scale\_frame = ttk.Frame(frame, padding = 10)  # scale\_frame.grid(row = 3, column = 1)  # 创建滑动条  dilation\_se\_size\_tip = ttk.Label(scale\_frame, text = "膨胀结构元大小: ")  dilation\_se\_size\_tip.grid(row = 0, column = 0)  dilation\_se\_size\_scale = tk.Scale(scale\_frame, from\_ = 3, to = 45, orient = "horizontal", length = 150, resolution = 2, command = update\_dilation\_se\_size)  dilation\_se\_size\_scale.grid(row = 0, column = 1)  dilation\_se\_size\_scale.set(3)  erosion\_se\_size\_tip = ttk.Label(scale\_frame, text = "腐蚀结构元大小: ")  erosion\_se\_size\_tip.grid(row = 1, column = 0)  erosion\_se\_size\_scale = tk.Scale(scale\_frame, from\_ = 3, to = 45, orient = "horizontal", length = 150, resolution = 2, command = update\_erosion\_se\_size)  erosion\_se\_size\_scale.grid(row = 1, column = 1)  erosion\_se\_size\_scale.set(3)  root.mainloop() |

指导老师评价

|  |  |  |
| --- | --- | --- |
|  | | |
| 仿真实验报告成绩 | 交流展示成绩 | 总评成绩 |
|  |  |  |

指导教师：

年 月 日

**实验报告提交说明:**

文件名 **姓名\_学号\_实验x.docx**

文件格式 Microsoft Word