图像处理与可视化 Homework-6 报告

林子开 21307110161

2023 年 11 月 19 日

目录

1 图像分割：基于 Kmeans 算法

1.1 Python 代码

Listing 1: Python codes

```python
import numpy as np
import random
from PIL import Image
import copy
from numba import njit


# 计算全局的灰度直方图
@njit
def globalHistogram(pixels:np.array) -> np.array:
    """
    para:pixels图片的灰度值矩阵
    return:全局灰度直方图
    """
    hist = np.zeros(256)
    for i in  range(pixels.shape[0]):
        for j in  range(pixels.shape[1]):
            p = pixels[i,j]
            hist[p] += 1
    return hist

```

```python

# Kmeans分割
def Kmeans(pixels:np.uint8, k: int) -> np.array:
    """
    para: pixels 是原图的像素灰度矩阵
    para: k 表示要分成几类
    return: 一个与原图相同尺寸的矩阵，矩阵的每个元素记录了该像素属于哪一个类别
    """
    hist = globalHistogram(pixels) # 原图的全局灰度直方图，kmeans本质上是对全局灰度直方图进行聚类
    classification = np.uint8(np.zeros(256)) # 用于存储每个灰度值的类别
    epsilon = 0.1 # 迭代停止阈值
    cla = np.empty(pixels.shape) # 用于记录最优的分割结果
    W =  float('inf') # 损失函数，即类内离差平方和

    for _ in range(5):   # 由于kmeans算法受到初始值的影响，可能会陷入局部最优，因此需要多次尝试
        # 随机初始化灰度值的类中心
        centers = np.float64( sorted(random.sample( range(256),k)))
        while True:
            # 第一步，更新灰度值的类别
            for p in  range(256):
                classification[p] = np.argmin([(p-center)**2 for center in centers]) # 分到最近的center

            # 第二步，更新聚类中心
            flg = 0 # 用于标识是否出现了某一类没有像素的情况
            new_centers = np.float64(np.zeros(k))
            for index in  range( len(new_centers)): # index表示是哪一个中心
                subHist = np.where(classification==index,1,0)*hist # 将所有属于该类的灰度值的直方图选取出来
                # 如果有一个类没有属于它的点了，则重新选点，重新开始
                if np. sum(subHist) < 1:
                    flg = 1
                    continue # 跳出内部的for循环
                else: # 聚类中心更新
                    new_centers[index] = np.inner(subHist,np.array( range(256))) / np.sum(subHist)

            if flg == 1: # 如果某个类不含任何像素，则重新随机选中心，重新开始迭代
                print('某一个类不包含任何像素，重新随机选取聚类中心，重新开始迭代')
                centers =  sorted(random.sample( range(256),k))
                continue

            # 根据新、旧聚类中心（可以各自表示为k维向量）之间的距离变化，判断是否满足迭代终止条件
            if np.linalg.norm(centers-new_centers) < epsilon:
                centers = new_centers
                break
            else:
                centers = new_centers

        # 计算损失函数
```

```
        W_new = 0
        for p in  range(256):
            classification[p] = np.argmin([(p-center)**2 for center in centers]) # 分到最近的
                center
            # 更新损失函数W
            W_new += hist[p]*(p-centers[classification[p]])**2
        if W_new < W:
            cla = copy.deepcopy(classification)
            W = W_new

    for i in  range(pixels.shape[0]):
        for j in  range(pixels.shape[1]):
            pixels[i,j] = cla[pixels[i,j]]

    return pixels


# 一些好看的颜色
color_dict = {0:(255,215,0), 1:(30,144,255), 2:(192,255,62),
               3:(132,112,255),4:(250,220,220),5:(255,127,80)}

def main():
    # 先用OSTU算法对图像进行二值化
    from util import OTSU
    OTSU(myPicture = '受到椒盐噪声污染的肝脏图像.png')

    # 以下是kmeans
    for k in (2,3,4,5,6):
        print(k)

        #导入图片
        myPicture = '受到椒盐噪声污染的肝脏图像.png'
        im_raw = Image. open(myPicture)
        im = im_raw.convert('L')
        pixels = np.array(im)
        pixels = np.uint8(pixels)

        # 基于Kmeans进行分割
        pixels_class = Kmeans(pixels,k)
        im_output = Image.fromarray(pixels_class)
        im_output = im_output.convert('RGB')
        pixels_output = im_output.load()
        pixels_class = np.transpose(pixels_class)
        for i in  range(im_output.size[0]):  # for every pixel:
            for j in  range(im_output.size[1]):
                pixels_output[i,j] = color_dict[pixels_class[i,j]]
        im_output.save('exercise1_result/'+'k='+ str(k)+'.png')


if __name__ == '__main__':
    main()
```

1.2 二分类，并与 OTSU 算法比较

用于测试的图片（带 1% 的椒盐噪声）如下：

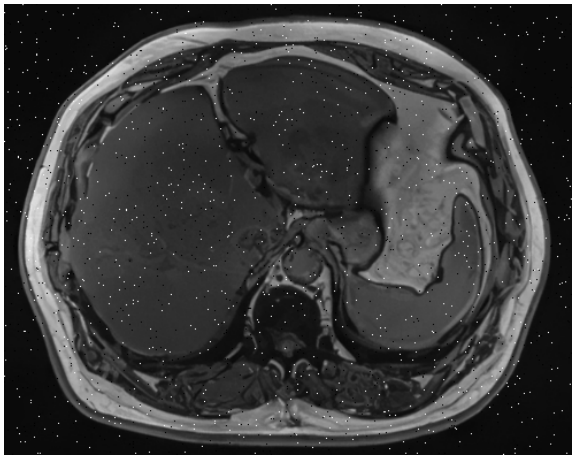
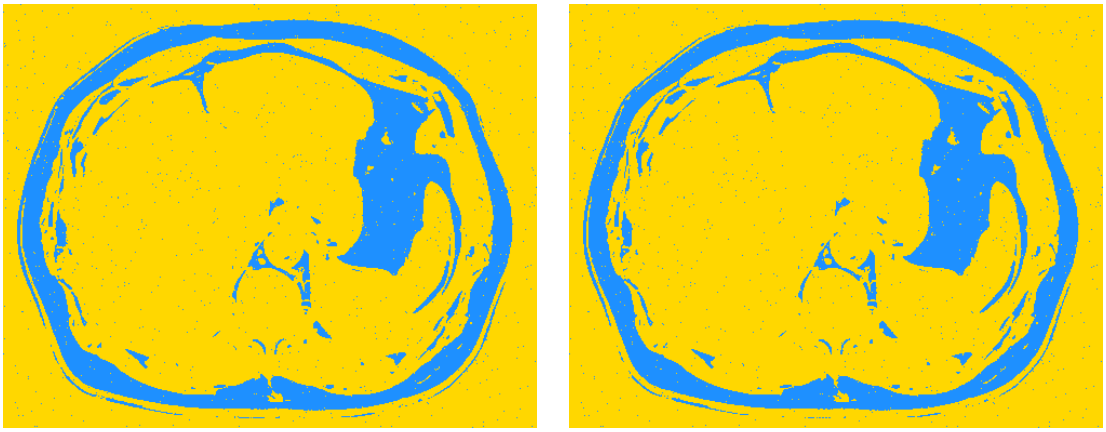


图 1: 测试图像（带 1% 的椒盐噪声）

用 Kmeans 做二类分割的效果如图2(a)所示，用 OSTU 进行二值化的效果如图2(b)所示。（注意，虽然用 OSTU 进行二值化的最终效果应该是黑白图片，但是为了与 Kmeans 统一，我将二值化后的结果为 0 的像素点设置为橙色，将二值化后结果为 255 的像素点设置为蓝色，以便读者与 kmeans 进行比较。）



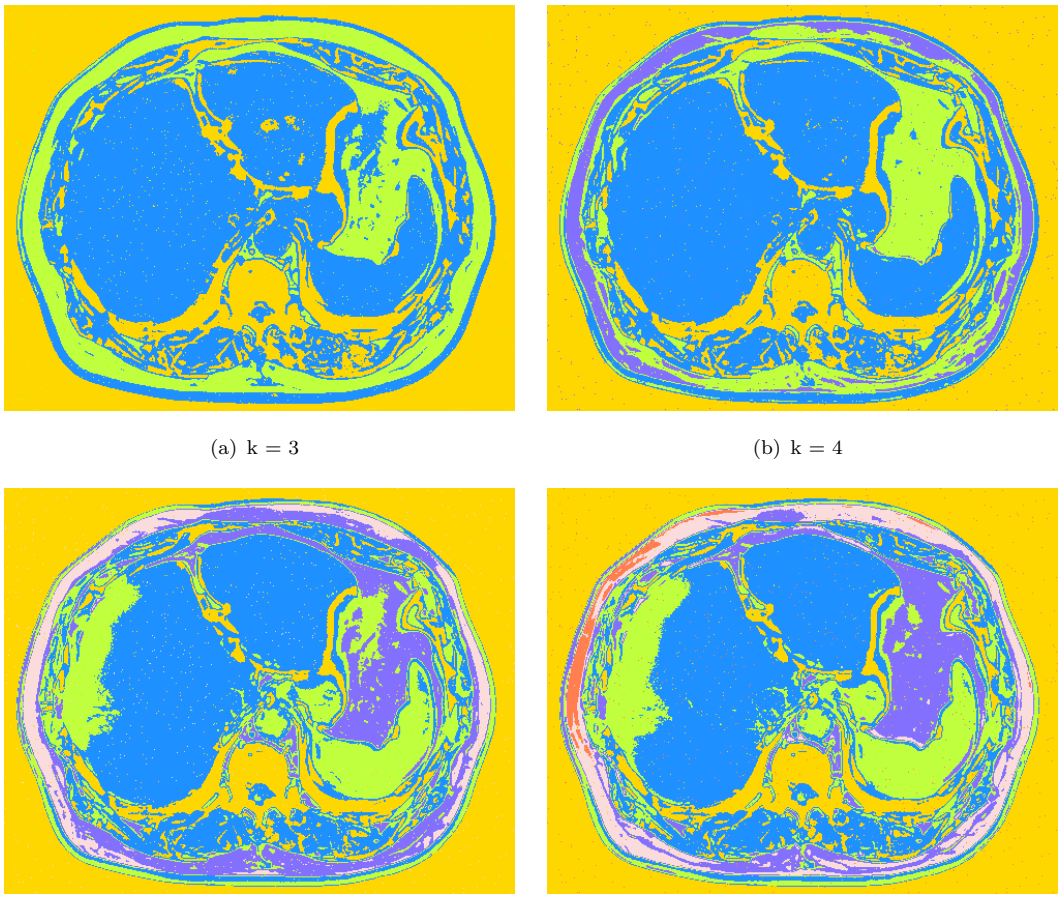
(a) Kmeans 二类分割　　(b) OSTU 二值化

图 2: Kmeans 和 OSTU 二值化效果对比

可以看出，两个算法进行二类分割的**效果基本一致**。

1.3 多分类

多分类分割的效果如下所示



(a) k = 3　(b) k = 4

(c) k = 5　(d) k = 6

图 3: 多类分割效果

可以发现，当 $k = 3$ 时，分类效果比较好。当 $k > 3$ 时，会分割出一些无意义的颜色类别。

1.4 分析带噪声图像分类不准确的原因并提出改进措施

椒盐噪声只分布在 0 和 255 两个灰度值上，因此在 kmeans 聚类时，带有椒盐噪声的像素点一定会被分到均值最低的类或均值最高的类，因此带椒盐噪声像素点的分类是确定的。另一方面，kmeans 算法**并不考虑图像的空间信息**，只是根据灰度值的大小对所有像素的灰度值进行聚类。当某个像素点出现了椒盐噪声，算法并没有考虑到其周围像素点的信息，不管噪声所在像素点邻域内的像素被分到了哪一类，都不会改变该噪声像素点的分类结果。所以，在分割的图像中，我们会明显地看到噪声所在的位置被分配到了和其邻域像素点完全不同的类别中，kmeans 对带噪声的图像进行分割的能力并不好。

如果想要得到更好的分割结果，可以**先用中值滤波器对椒盐噪声先进行去噪操作，再进行分割**。而对于其他类型的噪声，则要选取其他适合的滤波器，如高斯滤波器等，同样先去噪、后分割。

2 基于形态学操作进行补洞与噪声去除

2.1 Python 代码

Listing 2: Python codes

```python
import numpy as np
from PIL import Image
from numba import njit


# 腐蚀函数
@njit
def erode(pixels:np.array,b):
    """
    para: pixels是图像的灰度值矩阵
    b: b是结构元的大小，按照chessboard距离进行衡量；结构元为正方形
    return: 腐蚀后的图像的灰度值矩阵
    """
    new_pixels = np.zeros(pixels.shape)
    height, width = pixels.shape
    k = b//2
    for i in  range(height):
        for j in  range(width):
            up =  max(0,i-k)
            down =  min(height,i+k)
            left =  max(0,j-k)
            right =  min(width,j+k)
            new_pixels[i,j] = np. max(pixels[up:down,left:right])
    return new_pixels


# 膨胀函数
@njit
def dialte(pixels:np.array,b):
    """
    para: pixels是图像的灰度值矩阵
    b: b是结构元的大小，按照chessboard距离进行衡量；结构元为正方形
    return: 膨胀后的图像的灰度值矩阵
    """
    new_pixels = np.zeros(pixels.shape)
    height, width = pixels.shape
    k = b//2
    for i in  range(height):
        for j in  range(width):
            up =  max(0,i-k)
            down =  min(height,i+k)
            left =  max(0,j-k)
            right =  min(width,j+k)
            new_pixels[i,j] = np. min(pixels[up:down,left:right])
    return new_pixels


```

```python
# 闭操作，用于填补孔洞
@njit
def closing(pixels:np.array,b):
    """
    para: pixels是图像的灰度值矩阵
    b: b是结构元的大小，按照chessboard距离进行衡量；结构元为正方形
    return: 进行“闭”操作（先膨胀，后腐蚀）的图像的灰度值矩阵
    """
    pixels = dialte(pixels,b) # 先膨胀
    pixels = erode(pixels,b) # 后腐蚀
    return pixels


# 开操作，用于消除噪声
@njit
def opening(pixels:np.array,b):
    """
    para: pixels是图像的灰度值矩阵
    b: b是结构元的大小，按照chessboard距离进行衡量；结构元为正方形
    return: 进行“开”操作（先腐蚀，后膨胀）的图像的灰度值矩阵
    """
    pixels = erode(pixels,b) #先腐蚀
    pixels = dialte(pixels,b) # 后膨胀
    return pixels


def main():
    # 首先导入图像，并且转换为numpy的array类型
    myPicture = 'zmic_fdu_noise.bmp'
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)

    k1 = 7 # 闭操作的结构元的大小
    k2 = 5 # 开操作的结构元的大小

    # 先对图像进行“闭操作”，填补孔洞
    pixels = closing(pixels,k1)
    # 再对图像进行“开操作”，消除噪声
    pixels = opening(pixels,k2)


    pixels = np.uint8(pixels)  # 转成uint8格式，否则无法正常保存为图像
    im = Image.fromarray(pixels)
    im.save('exercise2_result/'+'result.png') # 存储图像


if __name__ == '__main__':
    main()
```

2.2 操作结果

先对图像用大小为 7×7 的正方形结构元进行闭操作，对孔洞进行填充；再对图像用大小为 5×5 的正方形结构元进行开操作，去除椒盐噪声。效果如下：

图 4: 进行形态学处理后的图像