HW5

赵心怡 19307110452

[HW5-1] 实现K类均值分类的分割算法或基于高斯混合模型的分割算法，并使用噪声污染过的图像（如P=0.1%的椒盐噪声）测试：

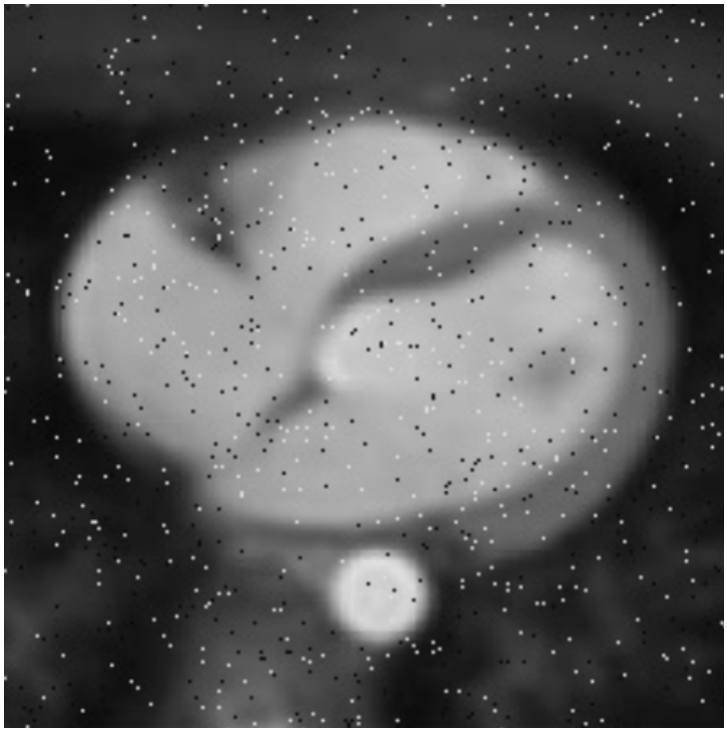
k means的算法是先随机k个初始中心，然后根据最小距离原则给每个点分类，然后更新类中心，直到两次聚类中心欧式距离和小于阈值则不再迭代

```
def k_means(img, k, threshold):
    minvalue = img.min()
    maxvalue = img.max() # 图片灰度值的最大最小值
    init_center = random.sample(range(minvalue,maxvalue + 1), k) # 随机生成k个初始中心
    current_center = init_center
    classmatrix = center_classify(img, init_center) # 初始各个像素的分类
    while True:
        last_center = current_center.copy() # 上一步聚类中心
        for each_class in range(k):
            this_class_pixel = img[np.where(classmatrix == each_class)] # 对每一类找到对应的像素点
            current_center[each_class] = sum(this_class_pixel) / len(this_class_pixel) # 算平均值更新新的聚类中心
        new_class = center_classify(img, current_center) # 根据新的聚类中心得到每个像素点新的类别
        if np.sum((np.array(current_center) - np.array(last_center))**2) <= threshold**2:
            break                # 如果两次聚类中心欧式距离和小于阈值则不再迭代
        classmatrix = new_class # 否则继续更新，把当前得到的新的聚类给下一步
    new_img = (new_class * 255/ (k-1)).astype(np.uint8)
    return new_img


def center_classify(img, center_list):
    height, width = img.shape
    classmatrix = np.zeros((height, width)) # 类别矩阵
    distance = []
    for (i,j) in itertools.product(range(height), range(width)):
        for k in range(len(center_list)):
            each_dist = np.sum((img[i][j] - center_list[k]) ** 2)
            distance.append(each_dist) # 计算当前像素点到每个聚类中心的距离
        classmatrix[i][j] = np.argmin(distance) # 算出距离最短的类别
        distance = [] # 置空方便下一个像素点使用
    return classmatrix
```

（1）测试二类分割，并对比自己实现的算法的分割结果与阈值算法（如OSTU或基于最大熵）二值化的结果；

首先我们用上次作业的代码生成了有椒盐噪声（p=1%）的图像，因为p=0.1%点太少了看不清因此选择p=1%。



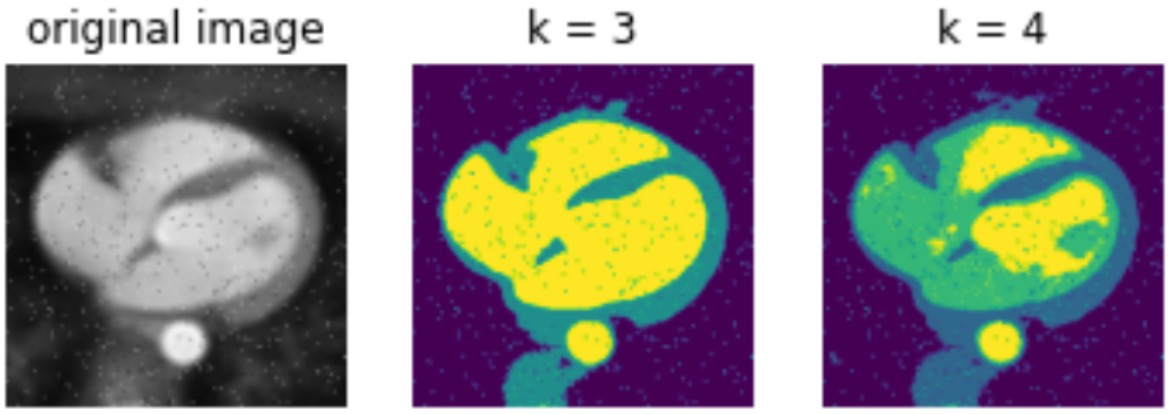
OSTU分类方法基于最大化组间方差。我们遍历每个灰度值，将图像分为前景和后景，再算出背景以及前景分别的概率以及均值，计算组间方差，再找到最大值所对应的灰度值即可。

```
def OSTU(img):
    minvalue = img.min()
    maxvalue = img.max() # 图片灰度值的最大最小值
    criterion_value = np.zeros(256)
    for threshold in range(minvalue,maxvalue+1):    # 移动阈值
        background = img[img < threshold]  # 背景
        weight_bg = background.size / img.size # 背景的概率
        mean_bg = background.mean() if background.size else 0  # 背景的均值
        foreground = img[img >= threshold] # 前景
        weight_fg = foreground.size / img.size #前景的概率
        mean_fg = foreground.mean() if foreground.size else 0 # 前景的均值
        criterion_value[threshold] = weight_bg * weight_fg * (mean_bg - mean_fg) ** 2   # 组间方差
    best_threshold = np.argmax(criterion_value) # 选择最大的值对应的灰度值作为最终阈值
    newImg = img > best_threshold
    return newImg * 255 # 得到最终图片
```

可以看到两种方法下的分割阈值的效果类似。但是时间上k_means方法花的时间长一些，而且时间受到初始类中心的位置影响较大。如图所示，前景和后景的分割不够准确。

（2）测试多类（大于等于三类）分割（请自己设定分割标签类别的个数）；

设定标签类别个数为3和4，分割结果如下



结果可以发现噪声图像分割效果并不好，噪声仍然保留着。k_means方法速度更慢，而且仍然受初始聚类中心影响。

（3）针对噪声图像，讨论为什么分割的结果不准确，有什么方法可以取得更好的分割结果（备注：不要求实现该方法，只是讨论）。

多类分割的结果都无法将噪声从图像中删除，同时分割的效果也并不理想，从图上的结果可以看出k_means算法的分割只能根据图像的已有信息来分割，没有自动判断噪声点的能力。

所谓的噪声对kmeans分割而言只是正常的图像而已，认为噪声和背景不是一类。

如果要得到较好的分割结果，应该还是先去除噪声，如通过中值滤波器等方法，然后再进行分类。

[HW5-2] 请使用课程学习的形态学操作实现二值图像的补洞和离散点去除。

我们尝试选择5*5的卷积核进行卷积操作。开闭操作函数以及对应的膨胀核腐蚀函数代码如下：

```
# Problem 2

def dilate(img,time):#膨胀操作
    height,width = img.shape
    kernel = np.array(((1,1,1,1,1),(1,1,1,1,1),(1,1,0,1,1),(1,1,1,1,1),
(1,1,1,1,1)))#选择核
    # 对卷积次数
```

```
    img_new = img.copy()
    for i in range(time):#膨胀次数
        tmp = np.pad(img_new, (2, 2), 'edge') #扩张图片
        for y in range(2, height+2):
            for x in range(2, width+2):
                if np.sum(kernel * tmp[y-2:y+3, x-2:x+3]) >= 255: # 原图片5*5范围内有无白色块，该像素点就是白的
                    img_new[y-2, x-2] = 255 #修改点
    return img_new



def erode(img, time):#腐蚀操作
    height, width = img.shape
    kernel = np.array(((1,1,1,1,1),(1,1,1,1,1),(1,1,0,1,1),(1,1,1,1,1),(1,1,1,1,1)))#卷积核
    # 对卷积次数
    img_new = img.copy()
    for i in range(time):#腐蚀次数
        tmp = np.pad(img_new, (2, 2), 'edge') #扩展图片
        for y in range(2, height+2):
            for x in range(2, width+2):
                if np.sum(kernel * tmp[y-2:y+3, x-2:x+3]) < 255*23: #原图5*5范围有无黑色块，该像素点就是黑的
                    img_new[y-2, x-2] = 0#修改点
    return img_new


def open(img, time):#先腐蚀再膨胀
    newImg = erode(img, time)
    cv2.imwrite('open_1.jpg', newImg)
    newImg = dilate(newImg, time)
    cv2.imwrite('open_2.jpg', newImg)
    return newImg

def close(img,time):#先膨胀再腐蚀
    newImg = dilate(img, time=time)
    cv2.imwrite('close_1.jpg', newImg)
    newImg = erode(newImg, time=time)
    cv2.imwrite('close_2.jpg', newImg)
    return newImg

```

预先处理：

因为开运算闭运算的前提是区分开前景核后景，而我们的图片前景是深色，因此预先作了翻转处理，这样就默认文字部分是前景。处理完图片后再翻转回来。

```
img = cv2.imread('fdu.jpg', cv2.IMREAD_GRAYSCALE) # 读入图片
newImg1 = 255-open(255-img,1)
newImg2 = 255-close(255-img,1)
```

观察原图可以发现除了椒盐噪声外，I的内部有白线，上面也有白线。

开闭操作结果分析：

开操作的结果：

从输出图像可以发现开操作能够除去孤立的小点，突出物，断开狭窄的颈而总的位置和形状不变。

图中黑色的噪声点被抹去了但字内内部的白色噪声还在，I上面的一条黑线被抹掉了，但是D的右边却出现了缺口。

不同的核以及结构元素大小的不同将导致不同的分割

闭操作的结果：

从输出图像可以看出闭运算能够填平小孔，填补小裂缝，而总的位置和形状不变。
图中I内部白色的裂缝被填平，白色的噪声也消除了，但是黑色噪声还在，而且黑线也还在。

补洞和离散点去除：

为了尝试能否同时消除两种噪声同时不损失图像中的文字信息，我先进行一次开操作再进行一次闭操作。

开操作+闭操作结果：

从输出图像可以发现，开闭操作结合可以消除两种噪点，但是也可能会造成信息损失，如第一步的开操作使D缺了的一角依然没有填补上。