数字图像处理作业

程铭 517021910750

1 区域生长算法

1.1 简单图像及复杂图像结果展示

（1）简单图像区域生长算法结果

图 1：简单图像原图像

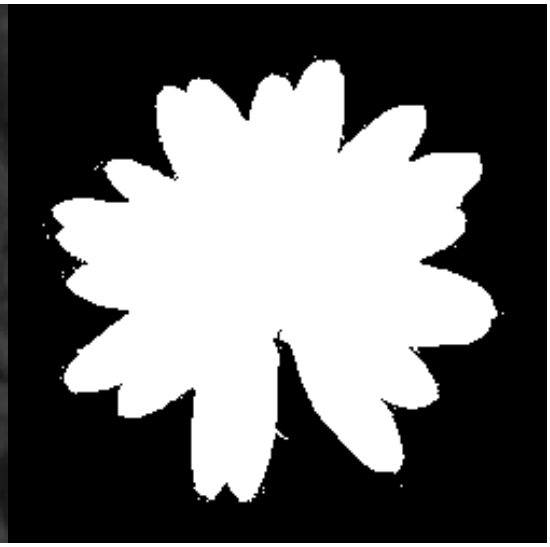

图 2：简单图像区域生长算法结果

如图 1，2 所示，在选取合适的种子点以及阈值后，对于简单图像，可以得到较好的结果。原因：对于简单图像，物体和背景的像素值相差较大，不会发生误判的情况（生长出边界）。

（2）复杂图像区域生长算法结果

图 3：复杂图像原图像

图 4：复杂图像区域生长算法结果

如图 3，4 所示，对于复杂图像，由于其物体和背景的像素值相差不大，只有部分轮廓可以分割出来，因此分割效果不是很好。

因此，从上述结果的对比分析可以知道，区域生长算法更适用于简单图像，即物体和背景的像素值差别较大，否则算法会误判物体和背景。另外，若物体内部像素分布较为平均（较为平滑），区域生长算法表现也较好。

1.2 不同种子点的选取的影响

（1）简单图像结果

图 5：简单图像原图像

图 6：种子点选取位置为花朵的中心

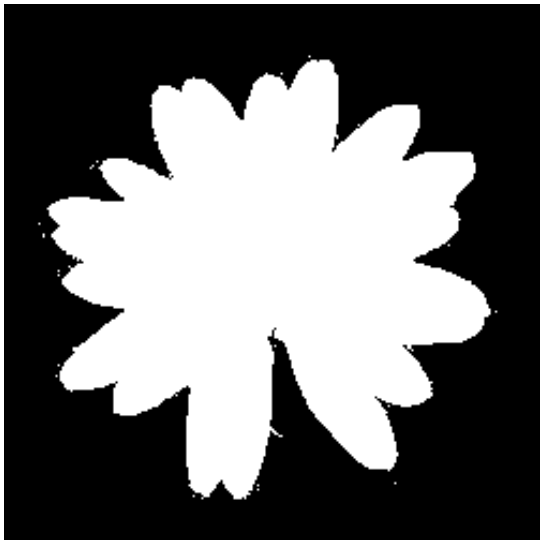

图 7：种子点选取位置为背景

图 8：种子点选取位置为花瓣

结果如图 5,6,7,8 所示，种子点位置选取不同，结果会有较大的不同。当种子点选取在花朵中间（坐标为(122, 124)）或者花瓣（坐标为(166, 80)）时，部分花瓣没有被分离出来。当种子点选取位置为背景时（坐标为(30, 26)），分割的结果较好，此时整个花朵都可以被分割出来。

（2）复杂图像结果

图 9：复杂图像原图像

图 10：种子点选取在犀牛身上

图 11：种子点选取为天空

图 12：种子点选取为草地

如图 9，10，11，12 所示，对于复杂图像，区域生长算法表现不是很好，且不同种子点的选取对结果的影响较大，甚至出现全白色的情况（图 11）。由于天空处的灰度值与犀牛、草地等处的灰度值差异较大，因此当种子点选取为天空时（坐标为(357,59)），其遇到犀牛、草地等就不再生长了，因此表现为全白色的情况；当种子点选取在犀牛身上（坐标为(312,255)）、或者是草地处（坐标为(107,422)）时，犀牛的部分轮廓可以被分割出来，这是因为犀牛的灰度值和草地的灰度值差异不大。

总体而言，对于复杂图像，区域生长算法表现不是很好，且不同位置的种子点选取影响很大。

1.3 不同阈值大小的结果对比

（1）简单图像

对于简单图像，固定种子点的位置（坐标(11, 12)），设定不同的阈值，比较结果。

图 13：阈值为 3

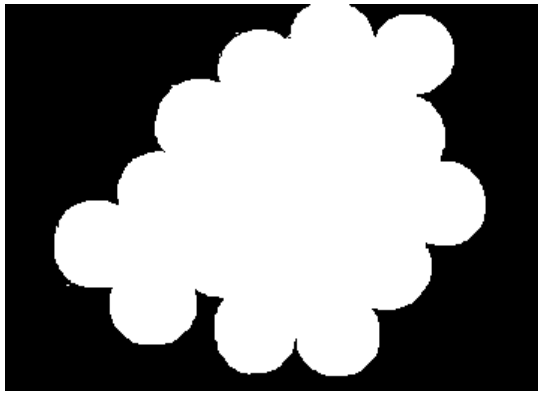

图 14：阈值为 5

图 15：阈值为 7

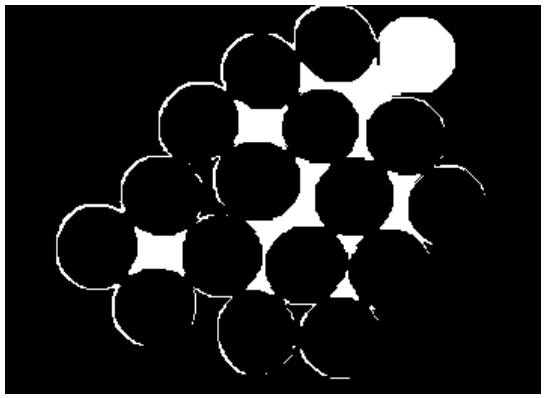

图 16：阈值为 15

如图 13，14，15，16 所示，对于不同的阈值，得到的结果差异较大。对比图 13、15 可知，当阈值选取较小时，得到的结果细节粗糙，有毛刺、锯齿存在；由图 16 可知，当阈值选取较大时，无法将物体和背景分开，即：生长越过了物体的边界。而对于合适的阈值（图 15），得到的结果较好，边界处也没有锯齿等存在。

因此，可以得出结论：对于简单图像，不同的阈值选取会影响结果的好坏：阈值过小，边界处会有锯齿、毛刺，阈值过大，会出现过度生长现象，无法将物体与背景相分离。

（2）复杂图像

固定种子点位置为大象身上，坐标为：（235，181）

图 17：阈值为 3

图 18：阈值为 5

图 19：阈值为 7

图 20：阈值为 15

如图 17，18，19，20 所示，对于复杂图像，不同阈值的选取会得到不同的结果。当阈值较小时，生长不足，图像几乎呈全白；当阈值过大时，生长越界，几乎无法分辨出大象的轮廓；而当阈值选取较合适（如图 18）时，得到的结果相对较好，可以看到大象的轮廓。

因此，可以总结：对于较小的阈值，生长不足，无法全部获得目标，甚至会出现全白的现象；对于较大的阈值，生长过度，各个区域均被生长到，依然无法分割出背景和物体。

总结严重影响区域生长算法结果的因素

（1）种子点的选取不同。对于不同的种子点，得到的结果不同，因此对于每个图像，应该选取与其对应的种子点，一般选取区域中心的位置较好。然而，由于种子点的选取需要人工设定，因此，选取恰当的种子点位置可能需要较多次的尝试。

（2）阈值的选取不同。对于不同的阈值大小（即不同的相似性准则），会得到不同的结果。当阈值选取过大时，生长过度，图像各个区域都被生长到，导致无法区分物体和背景；对于过小的阈值，生长不充分，仅有部分物体被生长到。因此，需要选择合适的阈值，才能得到较好的结果。

2 区域合并及分裂算法

2.1 区域合并及分割算法结果

（1）简单图像

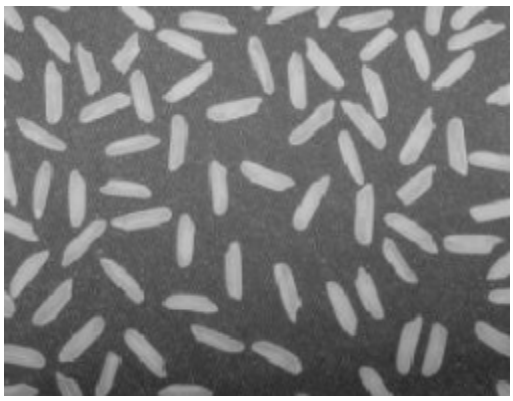

图 21：简单图像原图像

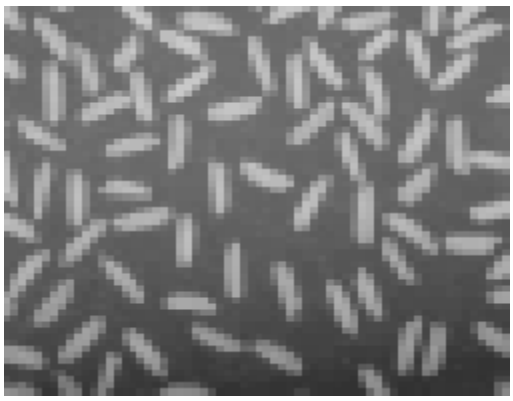

图 22：阈值为 10

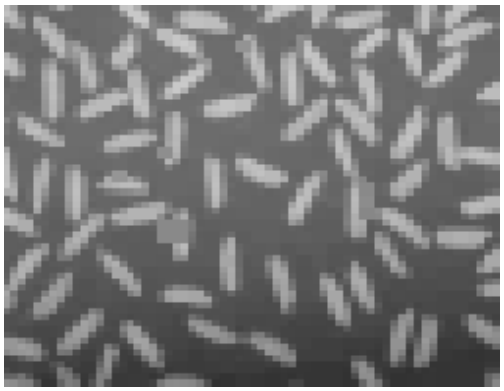

图 23：阈值为 20

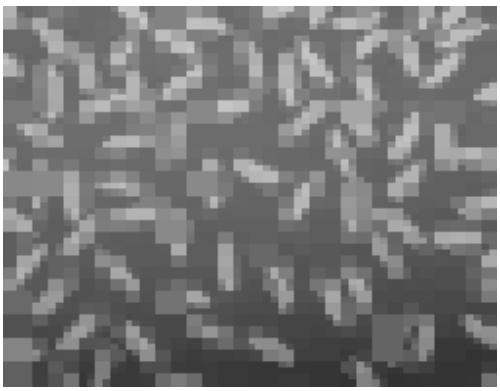

图 24：阈值为 50

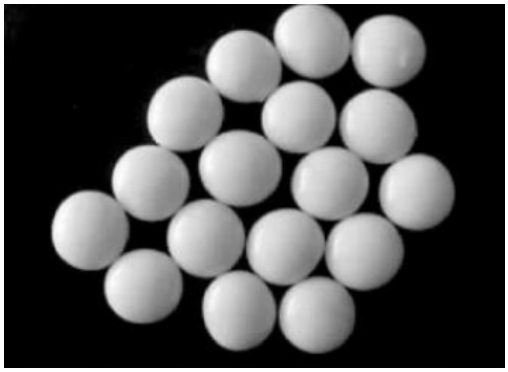

图 25：简单图像原图像

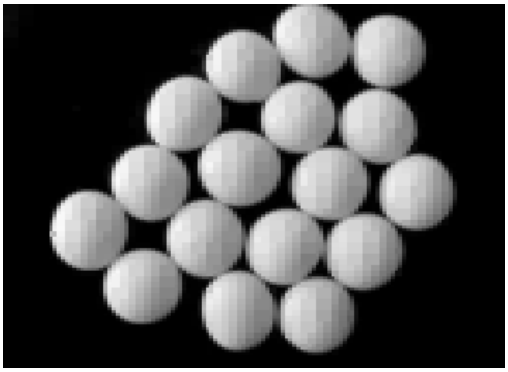

图 26：阈值为 10

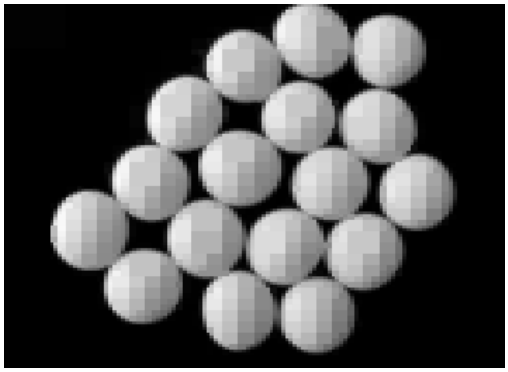

图 27：阈值为 20

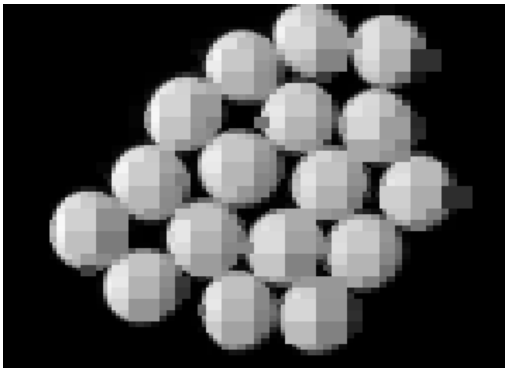

图 28：阈值为 50

由上述两组图可知，不同的阈值选取会得到不同的结果。当阈值选取较小时，相似性度量较

为严格，因此物体和背景可以分开，但选取过小会导致分裂过多，从而导致同一区域的像素没有被划分到同一区域；对于较大的阈值，相似性度量较为宽松，导致部分分裂操作没有进行，进而导致不同区域的像素被合并至同一区域，导致物体和背景无法分开。

（2）复杂图像

图 29：复杂图像原图像

图 30：阈值为 10

图 31：阈值为 20

图 32：阈值为 50

图 33：复杂图像原图像

图 34：阈值为 10

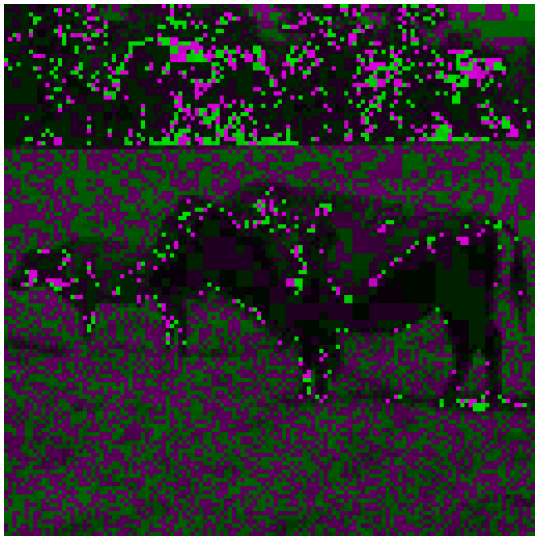

图 35：阈值为 20

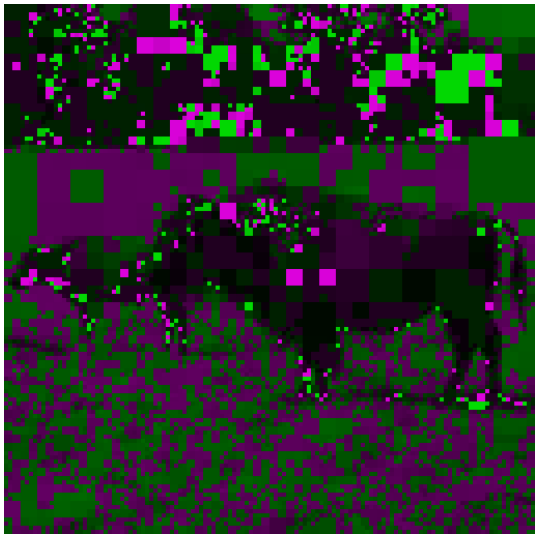

图 36：阈值为 50

与简单图像中的分析同理，对于复杂图像，不同的阈值也会导致不同的结果：当阈值较小时，相似性度量较为严格，物体和背景可以得到较好的分离，但选取过小会导致分裂过多，从而导致同一区域的像素没有被划分到同一区域；当阈值较大时，相似性度量较为宽松，导致一些分裂操作没有进行，从而导致将不同区域的像素合并到一起，进而导致物体和背景混在一起。

影响区域合并及分裂算法结果的因素

（1）分裂过程中的最小子区域的选取。分裂过程中的最小子区域的选取不同，会影响算法的结果：当选取较大时，会使得区域间的分界线较为粗糙（因为达到最小子区域的大小后，不会再分裂）；当选取较小时，算法运行时间较长。

（2）阈值选取的不同。当阈值选取较小时，相似性度量较为严格，物体和背景可以得到较好的分离，但选取过小会导致分裂过多，从而导致同一区域的像素没有被划分到同一区域；当阈值较大时，相似性度量较为宽松，导致一些分裂操作没有进行，从而导致将不同区域的像素合并到一起，进而导致物体和背景混在一起。

```
import cv2
import numpy as np
import os

def region_grow(img, seed_list, thresh):

    seed_np = np.full(img.shape,255)
    while(len(seed_list)>0):
        temp = seed_list.pop(0)
        x = temp[0]
        y = temp[1]
        seed_np[x,y] = 0
        for x_ in [x-1, x, x+1]:
            for y_ in [y-1, y, y+1]:
                if x_ >= 0 and y_ >= 0 and x_ < img.shape[0] and y_ <
img.shape[1]:
                    dis = abs(int(img[x,y]) - int(img[x_, y_]))
                    if dis < thresh:
                        flag = True
                    else:
                        flag = False
                    if flag and seed_np[x_,y_] != 0:
                        seed_np[x_,y_] = 0
                        seed_list.append([x_,y_])

    return seed_np

def interface():
    thresh_list = [3,5,7,15]
    for _, _, fname_list in os.walk('./regionGrow/'):
        for fname in fname_list:
            img = cv2.imread('./regionGrow/' + fname, 0)
            # combo_x_1, combo_y_1 = int(img.shape[0] / 2), int(img.shape[1] /
2)
            # combo_x_2, combo_y_2 = int(img.shape[0] * 3 / 4),
int(img.shape[1] / 4)
            # combo_x_3, combo_y_3 = int(img.shape[0] * 3 / 4),
int(img.shape[1] * 3 / 4)
            # # center_x, center_y = int(img.shape[0] / 2), int(img.shape[1] /
2)
            # # top_x, top_y = int(img.shape[0] / 4), int(img.shape[1] / 4)
            # # bottom_x, bottom_y = int(img.shape[0] * 3 / 4),
int(img.shape[1] * 3 / 4)
            # # seed_x_list = [center_x, top_x, bottom_x]
            # # seed_y_list = [center_y, top_y, bottom_y]
            # seed_x_list = [combo_x_1, combo_x_2, combo_x_3]
```

```
            # seed_y_list = [combo_y_1, combo_y_2, combo_y_3]
            seed = [11, 12]
            for thresh in thresh_list:
                img_after = region_grow(img, seed, thresh)
                cv2.imwrite('./after_region_grow/' + fname.split('.')[0] + '_'
+ \
                        'thresh_' + str(thresh) + '.png', img_after)
                cv2.waitKey(0)
                cv2.destroyAllWindows()

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

```
import cv2
import numpy as np
import os

# def visual(img, region):
#     res_img = np.zeros_like(img)

#     arr_np = np.unique(region)
#     for id in arr_np:
#         region_ = np.zeros_like(region)
#         region_[region==id] = 1
#         c = 1.0 * np.sum(img*region_) / np.sum(region_)
#         res_img[region==id] = c
#     return res_img

def judge(array, threshold):
    h, w = array.shape
    pixel_list = []
    threshold = threshold
    for i in range(h):
        for j in range(w):
            pixel_list.append(array[i,j])
    if len(pixel_list) <= 4:
        return True
    else:
        flag = True
        mean = np.mean(pixel_list)
        for pixel in pixel_list:
            if (pixel - mean) > threshold:
                flag = False
        return flag


def region_split_merge(img, region, threshold):
    h,w = img.shape
```

```
    temp_1 = img[:h/2), :w/2)]
    temp_2 = img[:h/2), w/2):]
    temp_3 = img[h/2):, :w/2)]
    temp_4 = img[h/2):, w/2):]

    judge_flag = (judge(temp_1, threshold) and judge(temp_2, threshold) and
judge(temp_3, threshold) and judge(temp_4, threshold))
    if judge_flag:
        mean = np.mean(img)
        for i in range(h):
            for j in range(w):
                region[i][j] = mean


    else:
        #         if (x2 - x1)/2)<minimum:
        #             temp_1 = region(img, x1, y1, x2, y1+(y2 - y1)/2))
        #             temp_2 = region(img, x1, y1+(y2 - y1)/2), x2, y2)
        #             return temp_1, temp_2
        #         elif (y2 - y1)/2)<minimum:
        #             temp_1 = region(img, x1, y1, x1+(x2 - x1)/2), y2)
        #             temp_2 = region(img, x1+(x2 - x1)/2), y1, x2, y2)
        #             return temp_1, temp_2
        #         else:
        #             temp_1 = region(img, x1, y1, x1+(x2 - x1)/2), y1+(y2 -
y1)/2))
        #             temp_2 = region(img, x1+(x2 - x1)/2), y1, x2, y1+(y2 -
y1)/2))
        #             temp_3 = region(img, x1, y1+(y2 - y1)/2), x1+(x2 -
x1)/2), y2)
        #             temp_4 = region(img, x1+(x2 - x1)/2), y1+(y2 - y1)/2),
x2, y2)
        #             return temp_1, temp_2, temp_3, temp_4
        if judge(np.concatenate([temp_1, temp_2], axis=1), threshold):
            if judge(np.concatenate([temp_1, temp_3], axis=0), threshold):
                region[:int(region.shape[0]/2), :int(region.shape[1]/2)] =
np.full(temp_1.shape, np.mean(temp_1))
                region[:int(region.shape[0]/2), int(region.shape[1]/2):] =
np.full(temp_2.shape, np.mean(temp_1))
                region[int(region.shape[0]/2):, :int(region.shape[1]/2)] =
np.full(temp_3.shape, np.mean(temp_1))

            else:
                region[:int(region.shape[0]/2), :int(region.shape[1]/2)] =
np.full(temp_1.shape, np.mean(temp_1))
                region[:int(region.shape[0]/2), int(region.shape[1]/2):] =
np.full(temp_2.shape, np.mean(temp_1))

        elif judge(np.concatenate([temp_1, temp_3], axis=0), threshold):
```

```
            region[:int(region.shape[0]/2), :int(region.shape[1]/2)] =
np.full(temp_1.shape, np.mean(temp_1))
            region[int(region.shape[0]/2):, :int(region.shape[1]/2)] =
np.full(temp_3.shape, np.mean(temp_1))

        if judge(np.concatenate([temp_3, temp_4], axis=1), threshold):
            #         if (x2 - x1)/2)<minimum:
            #             temp_1 = region(img, x1, y1, x2, y1+(y2 - y1)/2))
            #             temp_2 = region(img, x1, y1+(y2 - y1)/2), x2, y2)
            #             return temp_1, temp_2
            #         elif (y2 - y1)/2)<minimum:
            #             temp_1 = region(img, x1, y1, x1+(x2 - x1)/2), y2)
            #             temp_2 = region(img, x1+(x2 - x1)/2), y1, x2, y2)
            #             return temp_1, temp_2
            #         else:
            #             temp_1 = region(img, x1, y1, x1+(x2 - x1)/2), y1+(y2
- y1)/2))
            #             temp_2 = region(img, x1+(x2 - x1)/2), y1, x2, y1+(y2
- y1)/2))
            #             temp_3 = region(img, x1, y1+(y2 - y1)/2), x1+(x2 -
x1)/2), y2)
            #             temp_4 = region(img, x1+(x2 - x1)/2), y1+(y2 -
y1)/2), x2, y2)
            #             return temp_1, temp_2, temp_3, temp_4
            if judge(np.concatenate([temp_2, temp_4], axis=0), threshold):
                region[int(region.shape[0]/2):, :int(region.shape[1]/2)] =
np.full(temp_3.shape, np.mean(temp_4))
                region[int(region.shape[0]/2):, int(region.shape[1]/2):] =
np.full(temp_4.shape, np.mean(temp_4))
                region[:int(region.shape[0]/2), int(region.shape[1]/2):] =
np.full(temp_2.shape, np.mean(temp_4))

            else:
                region[int(region.shape[0]/2):, :int(region.shape[1]/2)] =
np.full(temp_3.shape, np.mean(temp_4))
                region[int(region.shape[0]/2):, int(region.shape[1]/2):] =
np.full(temp_4.shape, np.mean(temp_4))

        elif judge(np.concatenate([temp_2, temp_4], axis=0), threshold):
            region[:int(region.shape[0]/2), int(region.shape[1]/2):] =
np.full(temp_2.shape, np.mean(temp_2))
            region[int(region.shape[0]/2):, int(region.shape[1]/2):] =
np.full(temp_4.shape, np.mean(temp_2))

        region_split_merge(temp_1, region[:int(region.shape[0]/2),
:int(region.shape[1]/2)], threshold)
        region_split_merge(temp_2, region[:int(region.shape[0]/2),
int(region.shape[1]/2):], threshold)
```

```
        region_split_merge(temp_3, region[int(region.shape[0]/2):,
:int(region.shape[1]/2)], threshold)
        region_split_merge(temp_4, region[int(region.shape[0]/2):,
int(region.shape[1]/2):], threshold)

    return region

# def region(img, x1, y1, x2, y2):


def RegionMerge(image, threshold):
    # while not q.empty():
    #     if c % 100 == 0:
    #         print(c)
    #     c += 1
    #     reg = q.get()
    #     if split(thre, min_size):
    #         regs = reg.split(min_size)
    #         for r in regs:
    #             q.put(r)
    #     else:
    #         temp_lis.append(reg)
    img = image.convert(mode='L')
    img = np.array(img).astype("int")
    region = np.empty(img.shape)
    img_merge = region_split_merge(img, region, threshold)

    return img_merge

def interface():
    from PIL import Image
    thresh_list = [10, 20, 50]
    for _, _, fname_list in os.walk('./region_merge/'):
        for fname in fname_list:
            img = Image.open('./region_merge/' + fname)
            for thre_ in thresh_list:
                img_merge = RegionMerge(img, thre_)
                temp = Image.fromarray(img_merge).convert('P')
                temp.save('./after_region_merge/' + fname.split('.')[0] + '_'
+ str(thre_) + '.png')


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