**北京邮电大学软件学院**

**2022-2023学年第二学期实验报告**

**课程名称：** 多媒体技术与应用

**项目名称： 实验五 图像锐化及分割功能**

**项目完成人：**

**姓名： 王宇涵 学号： 2020211730**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_李朝晖\_\_\_\_\_\_\_\_\_\_\_

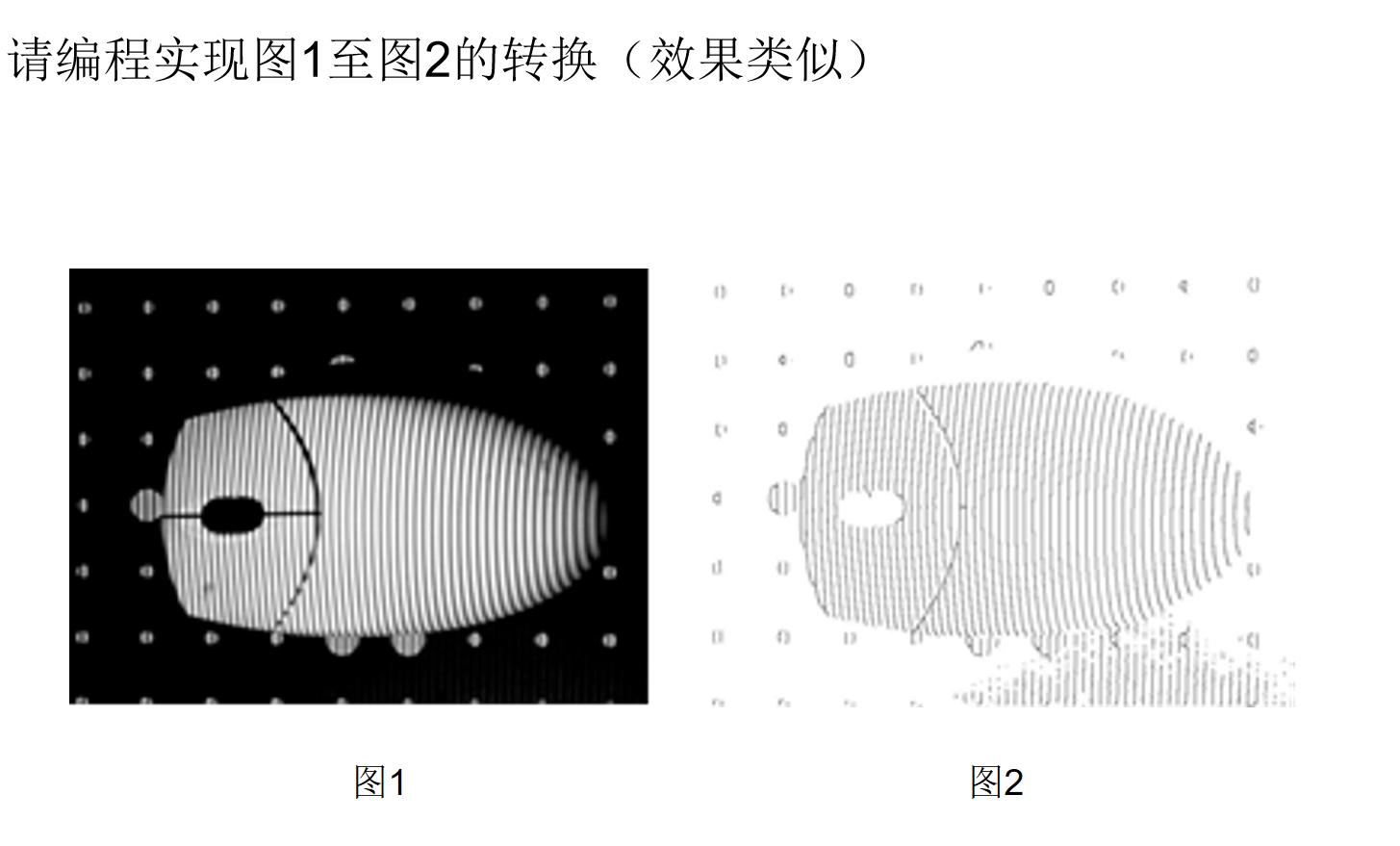
**日 期： 2023年 5 月 18 日**

1. **实验目的**

掌握图像锐化及分割相关原理及方法

1. **实验内容**

**图像锐化**



**图像分割**





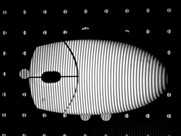
1. **实验环境**

Windows10操作系统

Pycharm(编译器)

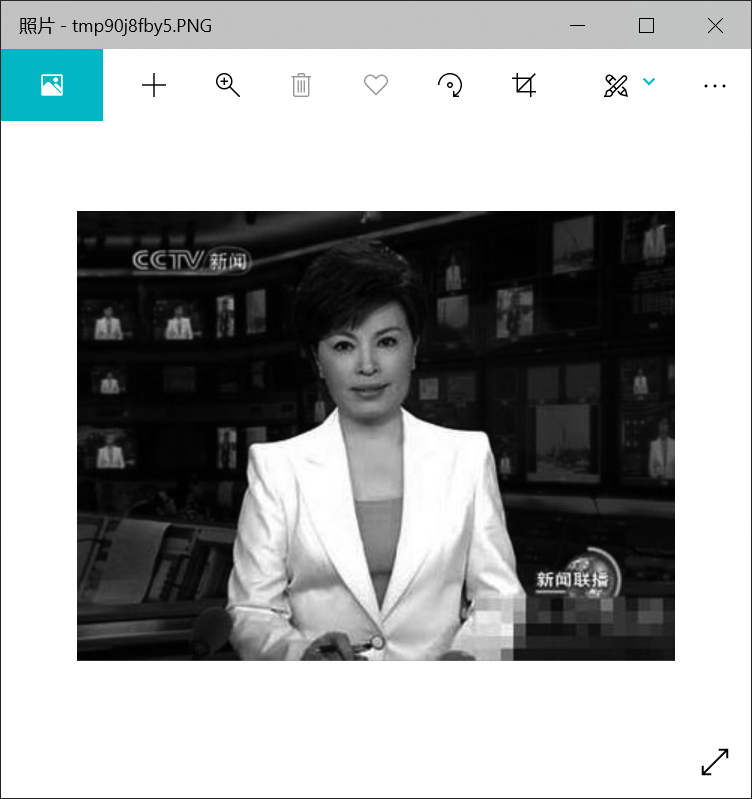
Python3.8

1. **实验结果**
2. 边缘提取



|  |  |  |
| --- | --- | --- |
| 所用掩模 | 提取结果（反转颜色后） | 灰度图 |
| Prewitt(X) | tmpablda586 | tmp_5cmgc22 |
| Prewitt(Y) | tmphdna8q0f | tmp_5cmgc22 |
| SobelX | tmp1j4484ul | tmp_5cmgc22 |
| SobelY | tmpp621nf5m | tmp_5cmgc22 |
| Laplacain1 | tmpasvhckuo | tmp_5cmgc22 |
| Laplacain2 | tmpax_iiniz | tmp_5cmgc22 |
| Laplacain3 | tmpt08d1ybv | tmp_5cmgc22 |
| Laplacain4 | tmp9bgxkrmx | tmp_5cmgc22 |

2.



|  |  |  |
| --- | --- | --- |
| 所用掩模 | 提取结果 | 原图 |
| Prewitt(X) |  |  |
| Prewitt(Y) |  |  |
| SobelX |  |  |
| SobelY |  |  |
| Laplacain1 |  |  |
| Laplacain2 |  |  |
| Laplacain3 |  |  |
| Laplacain4 |  |  |

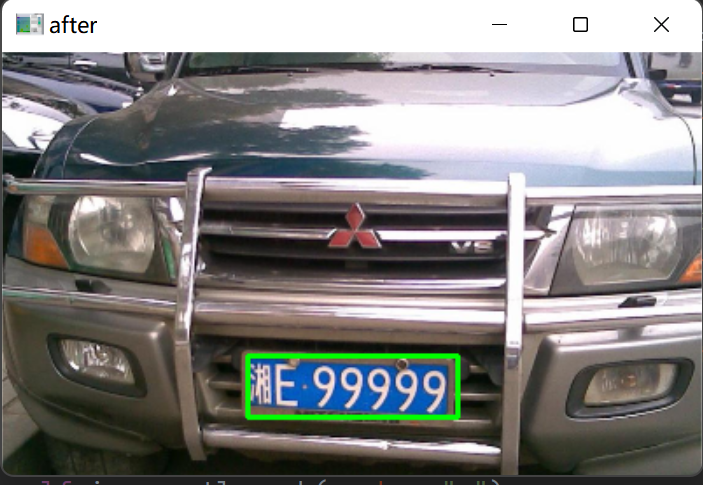
1. 图像提取

|  |  |  |
| --- | --- | --- |
| 原图 | 预处理 | 提取结果 |
|  |  |  |

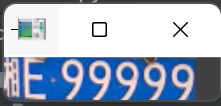
最终结果:



3.切割车牌







1. **附录**

**边缘提取**

Prewitt和Sobel算子是计算数字梯度时最常用的算子

Prewitt模板比Sobel模板简单,但是Sobel模板能够有效抑制噪声.

拉普拉斯算子

缺点:

对噪声具有敏感性

幅值产生双边缘

不能检测边缘的方向

优点:

可以利用零交叉的性质进行边缘定位

可以确定一个像素是在边缘暗的一边还是亮的一边

1. 导入所需库：导入了PIL（Python Imaging Library）、numpy、matplotlib.pyplot和matplotlib.image库，用于图像处理和显示。
2. 定义函数rgbTogray(rgb)：将真彩图转换为灰度图的函数。使用加权平均法将RGB图像的每个像素转换为灰度值。
3. 定义函数drawHistogram(image)：绘制图像的直方图。将图像转换为numpy数组，然后使用matplotlib.pyplot.hist函数绘制直方图。
4. 定义函数convolution(image, ck)：实现卷积操作的函数。对输入的图像和给定的卷积核进行卷积计算，返回卷积后的新图像。
5. 定义了多个卷积核：包括Prewitt、Sobel和Laplacian等不同类型的卷积核。
6. 打开图像文件：使用Image.open函数打开指定路径的图像文件。
7. 转换为灰度图：使用convert方法将图像转换为灰度图。
8. 进行卷积计算：使用convolution函数将灰度图像与指定的卷积核进行卷积计算。
9. 反色处理：使用ImageOps.invert函数对卷积后的图像进行反色处理，得到反色图像。
10. 显示图像：使用show方法显示反色图像。

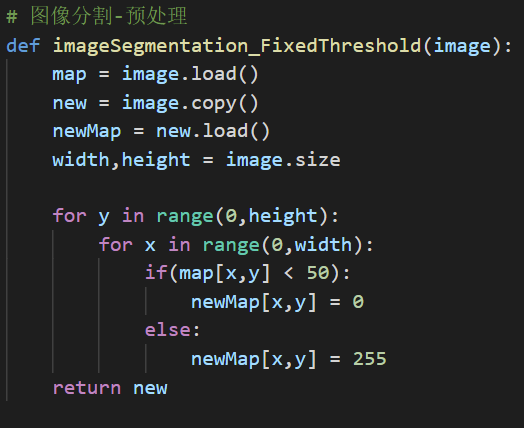
* # 真彩图转换为灰度图
    
  def rgbTogray(rgb):
    
   return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
    
    
  def drawHistogram(image):
    
   pic = np.array(image)
    
   plt.hist(pic[:, :].ravel(), bins=256,facecolor='#black', edgecolor='#898312')
    
   plt.show()
    
    
  # 卷积
    
  def convolution(image,ck):
    
   map = image.load()
    
   new = image.copy()
    
   newMap = new.load()
    
   width,height = image.size
    
    
   for y in range(0,height-2):
    
   for x in range(0,width-2):
    
   pixel = 0
    
   for i in range(0,3):
    
   for j in range(0,3):
    
   cur = map[x+i,y+j]\*ck[i][j]
    
   pixel = pixel + (int)(cur)
    
   newMap[x+1,y+1] = pixel
    
   return new
    
    
  # 卷积核
    
  PrewittX = [[-1,-1,-1],[0,0,0],[1,1,1]]
    
  PrewittY = [[-1,0,1],[-1,0,1],[-1,0,1]]
    
  SobelX = [[-1,-2,-1],[0,0,0],[1,2,1]]
    
  SobelY = [[-1,0,1],[-2,0,2],[-1,0,1]]
    
  Laplacain1 = [[0,1,0],[1,-4,1],[0,1,0]]
    
  Laplacain2 = [[0,-1,0],[-1,4,-1],[0,-1,0]]
    
  Laplacain3 = [[1,1,1],[1,-8,1],[1,1,1]]
    
  Laplacain4 = [[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]

**图像分割**

思路:



1. 预处理获得黑白图



设置阈值为50, 灰度值低于50的设置成黑色, 大于等于50的设置成白色.

效果:



* 1. 对图像进行处理,把这张图片的灰度数组变成”0-1”数组

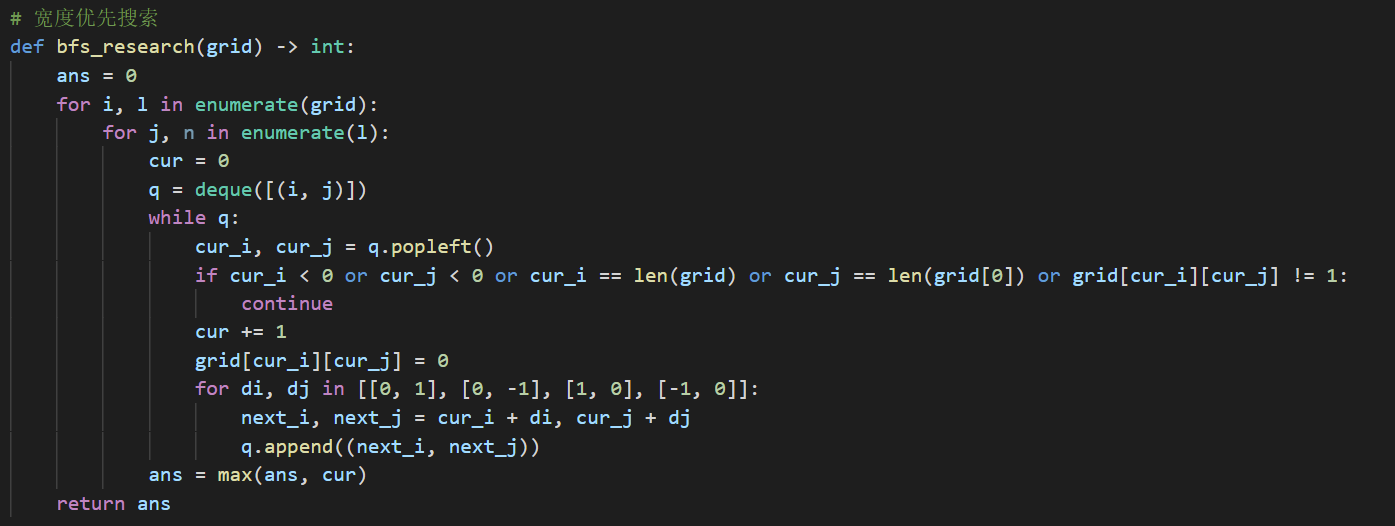


arr: 最后用来比对的数组

arr\_temp1: 用来搜索最大白色面积的数组

arr\_temp2: 用来生成arr过程中使用的消耗数组

1. 宽度优先搜索获取最大的白块面积

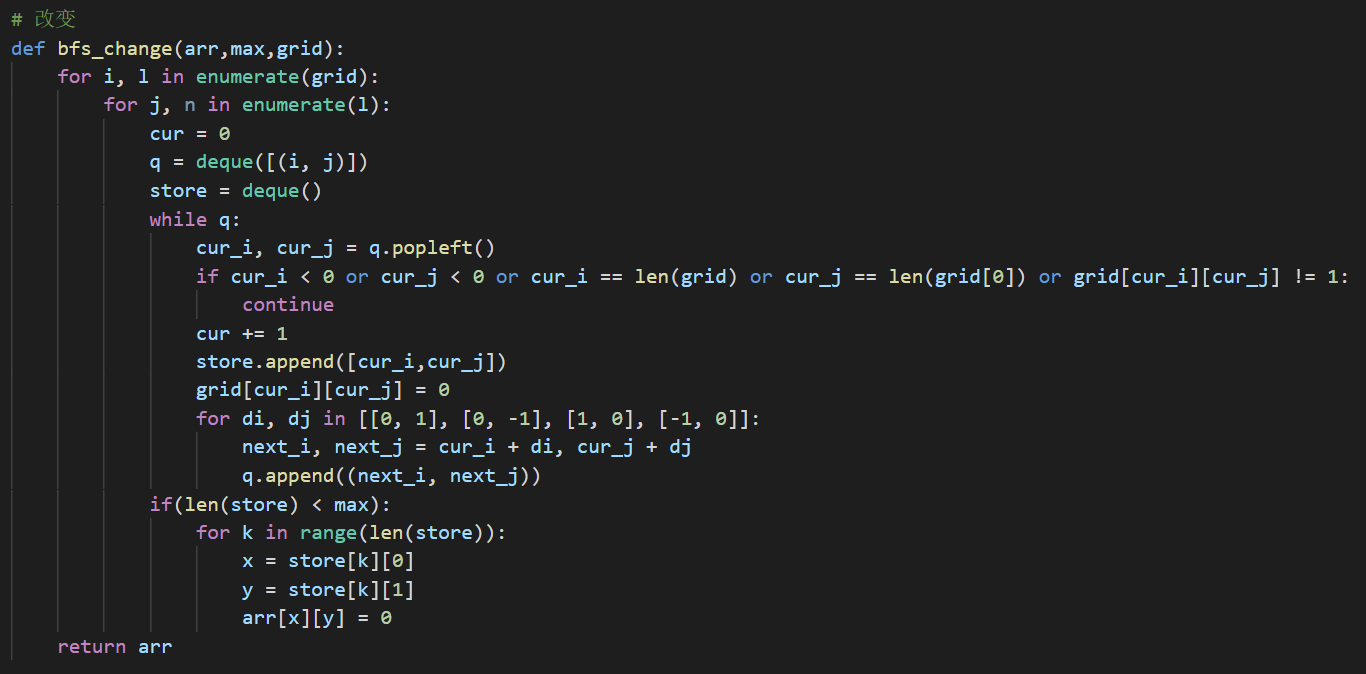


搜索出的结果为: 31833

1. 去除噪音

思路: 再次利用宽度优先搜索, 对于每一个搜索出的白色区域, 进行判断, 如果小于最大的白块区域的面积,则把这部分的”1”设置为0. 最后返回得到的用来最后比对的数组arr.

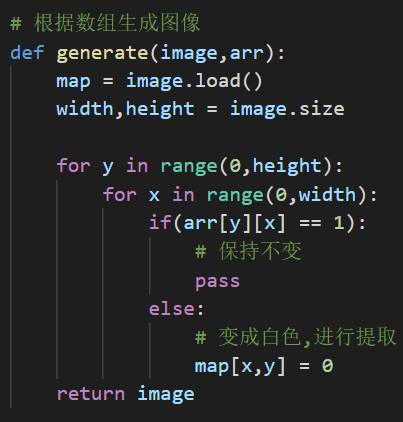
因为宽度优先搜索在搜索过程中会把”1”置”0”,所以就需要一个辅助数组”arr\_temp”



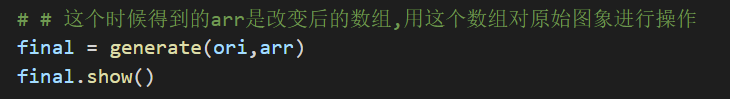
调用部分:



1. 最后用arr与原图做对比,提取人像



调用部分:



生成的结果:



**分割车牌：**

主要思路如下：

1. 导入所需库：通过导入cv2、numpy和os库，准备所需的图像处理和文件操作功能。
2. 加载图像：使用load\_image函数，根据指定的路径加载图像文件。
3. 灰度拉伸：对加载的图像进行灰度拉伸操作，通过拉伸灰度值的范围来改善图像质量，使用gray\_stretch函数实现。  
   在较暗的区域而导致图像偏暗，可以用灰度拉伸功能来拉伸(斜率>1)物体灰度区间以改善图像；同样如果图像灰度集中在较亮的区域而导致图像偏亮，也可以用灰  
   度拉伸功能来压缩(斜率<1)物体灰度区间以改善图像质量。  
   灰度拉伸  
   定义：灰度拉伸，也称对比度拉伸，是一种简单的线性点运算。作用：扩展图像的  
    直方图，使其充满整个灰度等级范围内  
   公式：  
   g(x,y) = 255 / (B - A) \* [f(x,y) - A],  
   其中，A = min[f(x,y)],最小灰度级；B = max[f(x,y)],最大灰度级；  
    f(x,y)为输入图像,g(x,y)为输出图像  
   缺点：如果灰度图像中最小值A=0，最大值B=255，则图像没有什么改变
4. 图像整体二值化：将灰度拉伸后的图像进行整体二值化操作，将图像像素分为两类（黑色和白色），使用image\_binary函数实现。
5. 车牌定位：利用形态学操作对二值化后的图像进行处理，寻找到可能的车牌位置，使用loacte\_plate函数实现。该函数首先找到图像中的轮廓，然后计算每个轮廓区域的面积和长宽比，并进行颜色识别，选取权值最大的区域作为车牌位置。
6. 车牌分割：根据定位到的车牌位置坐标，在原图上绘制矩形框出车牌位置，并使用cut\_license函数进行图像分割，将车牌从原图中裁剪出来。
7. 遍历图像文件夹：通过runing函数，遍历指定文件夹中的图像文件。对于每个图像文件，依次调用上述的图像处理和识别函数，实现车牌识别。

#方法
  
#导入图片资源 path为路径
  
def load\_image(path):
  
 src=cv.imread(path)
  
 return src
  
#灰度拉伸方法
  
  
def gray\_stretch(image):
  
 max\_value=float(image.max())
  
 min\_value=float(image.min())
  
 for i in range(image.shape[0]):
  
 for j in range(image.shape[1]):
  
 image[i,j]=(255/(max\_value-min\_value)\*image[i,j]-(255\*min\_value)/(max\_value-min\_value))
  
 return image
  
'''
  
图像整体二值化
  
'''
  
def image\_binary(image):
  
 max\_value=float(image.max())
  
 min\_value=float(image.min())
  
 '''
  
 这里利用自适应二值化以及自动求阈值的方法明显效果不好。因此设置阈值这样设置的效果很容易想到，当图片为一张纯色图时阈值为对应像素值，当图包含
  
 255与0时，阈值为122，总体的适应的效果会比较好。方法返回二值图
  
 '''
  
 ret=max\_value-(max\_value-min\_value)/2
  
 ret,thresh=cv.threshold(image,ret,255,cv.THRESH\_BINARY)
  
 return thresh
  
  
'''
  
矩形轮廓角点，寻找到矩形之后记录角点，用来做参考以及画图。
  
'''
  
def find\_rectangle(contour):
  
 y,x=[],[]
  
 for value in contour:
  
 y.append(value[0][0])
  
 x.append(value[0][1])
  
 return [min(y),min(x),max(y),max(x)]
  
  
'''
  
车牌定位方法，需要两个参数，第一个是用来寻找位置，第二个为原图，用来绘制矩形。寻找位置的图片为经过几次形态学操作的图片。这里利用权值的操作，实
  
现了定位的最高概率。
  
'''
  
def loacte\_plate(image,after):
  
 '''
  
 定位车牌号
  
 '''
  
 #寻找轮廓
  
 contours,hierarchy=cv.findContours(image,cv.RETR\_EXTERNAL,cv.CHAIN\_APPROX\_SIMPLE)
  
 img\_copy = after.copy()
  
 #找出最大的三个区域
  
 solving=[]
  
 for c in contours:
  
 r=find\_rectangle(c)
  
 '''
  
 这里就算出面积和长宽比
  
 '''
  
 a=(r[2]-r[0])\*(r[3]-r[1])
  
 s=(r[2]-r[0])/(r[3]-r[1])
  
  
 solving.append([r,a,s])
  
 #通过参考选出面积最大的区域
  
 solving=sorted(solving,key=lambda b: b[1])[-3:]
  
 #颜色识别
  
 maxweight,maxindex=0,-1
  
 for i in range(len(solving)):#
  
 wait\_solve=after[solving[i][0][1]:solving[i][0][3],solving[i][0][0]:solving[i][0][2]]
  
 #BGR转HSV
  
 hsv=cv.cvtColor(wait\_solve,cv.COLOR\_BGR2HSV)
  
 #蓝色车牌的范围 Hsv色彩空间的设置。
  
 lower=np.array([100,50,50])
  
 upper=np.array([140,255,255])
  
 #利用inrange找出掩膜
  
 mask=cv.inRange(hsv,lower,upper)
  
 #计算权值用来判断。
  
 w1=0
  
 for m in mask:
  
 w1+=m/255
  
 w2=0
  
 for n in w1:
  
 w2+=n
  
 #选出最大权值的区域
  
 if w2>maxweight:
  
 maxindex=i
  
 maxweight=w2
  
 return solving[maxindex][0]
  
'''
  
框出车牌 获取位置坐标，并返回图像
  
'''
  
#对图像的预处理
  
def find\_plates(image):
  
 image=cv.resize(image,(400,int(400 \* image.shape[0] / image.shape[1])))
  
 #转换为灰度图像
  
 gray\_image=cv.cvtColor(image,cv.COLOR\_BGR2GRAY)
  
 #灰度拉伸
  
 #如果一幅图像的灰度集中在较暗的区域而导致图像偏暗，可以用灰度拉伸功能来拉伸(斜率>1)物体灰度区间以改善图像；
  
 # 同样如果图像灰度集中在较亮的区域而导致图像偏亮，也可以用灰度拉伸功能来压缩(斜率<1)物体灰度区间以改善图像质量
  
 stretchedimage=gray\_stretch(gray\_image)#进行灰度拉伸，是因为可以改善图像的质量
  
  
 '''进行开运算，用来去除噪声'''
  
 #构造卷积核
  
  
 kernel=cv.getStructuringElement(cv.MORPH\_ELLIPSE,(30,30))
  
 #开运算
  
 openingimage=cv.morphologyEx(stretchedimage,cv.MORPH\_OPEN,kernel)
  
 #获取差分图，两幅图像做差 cv2.absdiff('图像1','图像2')
  
 strtimage=cv.absdiff(stretchedimage,openingimage)
  
  
 #图像二值化
  
 binaryimage=image\_binary(strtimage)
  
 #canny边缘检测
  
 canny=cv.Canny(binaryimage,binaryimage.shape[0],binaryimage.shape[1])
  
 #5 24效果最好
  
 kernel=np.ones((5,24),np.uint8)
  
 closingimage=cv.morphologyEx(canny,cv.MORPH\_CLOSE,kernel)
  
 openingimage=cv.morphologyEx(closingimage,cv.MORPH\_OPEN,kernel)
  
 #11 6的效果最好
  
 kernel=np.ones((11,6),np.uint8)
  
 openingimage=cv.morphologyEx(openingimage,cv.MORPH\_OPEN,kernel)
  
 #消除小区域，定位车牌位置
  
 rect=loacte\_plate(openingimage,image)#rect包括轮廓的左上点和右下点，长宽比以及面积
  
 #展示图像
  
 cv.imshow('image',image)
  
 cv.rectangle(image, (rect[0]-5, rect[1]-5), (rect[2]+5,rect[3]+5), (0, 255, 0), 2)
  
 cv.imshow('after', image)
  
 cutimg = cut\_license(image, rect)
  
 cv.imshow('cutimg', cutimg)
  
 cv.waitKey(0)
  
 cv.destroyAllWindows()
  
def cut\_license(afterimg,rect):
  
 '''
  
 图像分割函数
  
 '''
  
 #转换为宽度和高度
  
 rect[2] = rect[2] - rect[0]
  
 rect[3] = rect[3] - rect[1]
  
 rect\_copy = tuple(rect.copy())
  
 rect = [0, 0, 0, 0]
  
 #创建掩膜
  
 mask = np.zeros(afterimg.shape[:2], np.uint8)
  
 # mask = np.astype('uint8')
  
 #创建背景模型 大小只能为13\*5，行数只能为1，单通道浮点型
  
 bgdModel = np.zeros((1, 65), np.float64)
  
 #创建前景模型
  
 fgdModel = np.zeros((1, 65), np.float64)
  
 #分割图像
  
 cv.grabCut(afterimg, mask, rect\_copy, bgdModel, fgdModel, 10, cv.GC\_INIT\_WITH\_RECT)
  
 mask2 = np.where((mask == 1) | (mask == 0), 0, 1).astype('uint8')
  
 img\_show = afterimg \* mask2[:, :, np.newaxis]
  
 # 分割车牌与背景
  
 return img\_show[rect\_copy[1]:rect\_copy[1] + rect\_copy[3], rect\_copy[0]:rect\_copy[0] + rect\_copy[2]]
  
  
def runing():
  
 file\_path='.\\plates'
  
 for filewalks in os.walk(file\_path):
  
 for files in filewalks[2]:
  
 print('正在处理',os.path.join(filewalks[0],files))
  
 find\_plates(load\_image(os.path.join(filewalks[0],files)))
  
  
  
runing()

代码主要包括以下功能：

1. 导入所需库：导入了cv2、numpy和os库。
2. load\_image函数：根据给定路径加载图像并返回。
3. gray\_stretch函数：对灰度图像进行灰度拉伸操作，通过拉伸灰度值的范围来改善图像质量。
4. image\_binary函数：对图像进行整体二值化操作，根据阈值将图像像素分为两类（黑色和白色）。
5. find\_rectangle函数：根据轮廓点集，找到包围轮廓的矩形的角点坐标。
6. loacte\_plate函数：定位车牌的方法，根据经过形态学操作后的图像，在图像中寻找车牌位置。通过计算区域的面积和长宽比，并进行颜色识别，选取最大权值的区域作为车牌位置。
7. find\_plates函数：对图像进行预处理和定位车牌操作。首先将图像进行灰度拉伸、开运算和闭运算等形态学操作，然后调用loacte\_plate函数定位车牌位置，最后在原图上绘制矩形框出车牌位置。
8. cut\_license函数：图像分割函数，根据定位到的车牌位置坐标，在原图中裁剪出车牌图像。
9. runing函数：遍历指定文件夹中的图像文件，调用find\_plates函数进行车牌识别。

**源代码：**

边缘提取源代码:

from PIL import Image, ImageOps  
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
from numpy.lib.function\_base import diff  
from numpy.lib.type\_check import imag  
# 真彩图转换为灰度图  
def rgbTogray(rgb):  
 return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])  
  
def drawHistogram(image):  
 pic = np.array(image)  
 plt.hist(pic[:, :].ravel(), bins=256,facecolor='#black', edgecolor='#898312')  
 plt.show()  
  
# 卷积  
def convolution(image,ck):  
 map = image.load()  
 new = image.copy()  
 newMap = new.load()  
 width,height = image.size  
  
 for y in range(0,height-2):  
 for x in range(0,width-2):  
 pixel = 0  
 for i in range(0,3):  
 for j in range(0,3):  
 cur = map[x+i,y+j]\*ck[i][j]  
 pixel = pixel + (int)(cur)  
 newMap[x+1,y+1] = pixel  
 return new  
  
# 卷积核  
PrewittX = [[-1,-1,-1],[0,0,0],[1,1,1]]  
PrewittY = [[-1,0,1],[-1,0,1],[-1,0,1]]  
SobelX = [[-1,-2,-1],[0,0,0],[1,2,1]]  
SobelY = [[-1,0,1],[-2,0,2],[-1,0,1]]  
Laplacain1 = [[0,1,0],[1,-4,1],[0,1,0]]  
Laplacain2 = [[0,-1,0],[-1,4,-1],[0,-1,0]]  
Laplacain3 = [[1,1,1],[1,-8,1],[1,1,1]]  
Laplacain4 = [[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]  
  
# image = Image.open(r'播音员（gray）.jpg')  
image = Image.open(r'img1.jpg')  
# image.show()  
# Convert the image to grayscale  
gray\_image = image.convert('L')  
  
# Display the grayscale image  
# gray\_image.show()  
  
image = convolution(gray\_image,Laplacain4)  
# image.show()  
# Assuming `image` is the grayscale image obtained after convolution  
# inverted\_image = Image.eval(image, lambda x: 255 - x)  
inverted\_image = ImageOps.invert(image)  
# Display the inverted image  
inverted\_image.show()

人脸提取源代码:

from PIL import Image  
import numpy as np  
from collections import deque   
from numpy.lib.function\_base import diff  
from numpy.lib.type\_check import imag  
  
# 图像分割-预处理  
def imageSegmentation\_FixedThreshold(image):  
 map = image.load()  
 new = image.copy()  
 newMap = new.load()  
 width,height = image.size  
   
 for y in range(0,height):  
 for x in range(0,width):  
 if(map[x,y] < 50):  
 newMap[x,y] = 0  
 else:  
 newMap[x,y] = 255  
 return new  
  
   
# 宽度优先搜索  
def bfs\_research(grid) -> int:  
 ans = 0  
 for i, l in enumerate(grid):  
 for j, n in enumerate(l):  
 cur = 0  
 q = deque([(i, j)])  
 while q:  
 cur\_i, cur\_j = q.popleft()  
 if cur\_i < 0 or cur\_j < 0 or cur\_i == len(grid) or cur\_j == len(grid[0]) or grid[cur\_i][cur\_j] != 1:  
 continue  
 cur += 1  
 grid[cur\_i][cur\_j] = 0  
 for di, dj in [[0, 1], [0, -1], [1, 0], [-1, 0]]:  
 next\_i, next\_j = cur\_i + di, cur\_j + dj  
 q.append((next\_i, next\_j))  
 ans = max(ans, cur)  
 return ans  
  
# 改变  
def bfs\_change(arr,max,grid):  
 for i, l in enumerate(grid):  
 for j, n in enumerate(l):  
 cur = 0  
 q = deque([(i, j)])  
 store = deque()  
 while q:  
 cur\_i, cur\_j = q.popleft()  
 if cur\_i < 0 or cur\_j < 0 or cur\_i == len(grid) or cur\_j == len(grid[0]) or grid[cur\_i][cur\_j] != 1:  
 continue  
 cur += 1  
 store.append([cur\_i,cur\_j])  
 grid[cur\_i][cur\_j] = 0  
 for di, dj in [[0, 1], [0, -1], [1, 0], [-1, 0]]:  
 next\_i, next\_j = cur\_i + di, cur\_j + dj  
 q.append((next\_i, next\_j))  
 if(len(store) < max):  
 for k in range(len(store)):  
 x = store[k][0]  
 y = store[k][1]  
 arr[x][y] = 0  
 return arr  
  
# 根据数组生成图像  
def generate(image,arr):  
 map = image.load()  
 width,height = image.size  
   
 for y in range(0,height):  
 for x in range(0,width):  
 if(arr[y][x] == 1):  
 # 保持不变  
 pass  
 else:  
 # 变成黑色,进行提取  
 map[x,y] = 0  
 return image  
  
ori = Image.open(r'播音员（gray）.jpg')  
ori.show()  
new = imageSegmentation\_FixedThreshold(ori)  
new.show()  
  
width,height = new.size  
arr = np.array(new)  
temp = new.copy()  
arr\_temp1 = np.array(temp)  
arr\_temp2 = np.array(temp)  
  
for i in range(len(arr)):  
 for j in range(len(arr[i])):  
 if(arr[i][j] == 255):  
 arr[i][j] = 1  
  
for i in range(len(arr\_temp1)):  
 for j in range(len(arr\_temp1[i])):  
 if(arr\_temp1[i][j] == 255):  
 arr\_temp1[i][j] = 1  
  
for i in range(len(arr\_temp2)):  
 for j in range(len(arr\_temp2[i])):  
 if(arr\_temp2[i][j] == 255):  
 arr\_temp2[i][j] = 1  
  
# 得到最大的白色区域的面积  
maxArea = bfs\_research(arr\_temp1)  
  
# 开始改变  
arr = bfs\_change(arr,maxArea,arr\_temp2)  
  
# # 这个时候得到的arr是改变后的数组,用这个数组对原始图象进行操作  
final = generate(ori,arr)  
final.show()

切割车牌

#导库  
import cv2 as cv  
import numpy as np  
import os  
#方法  
#导入图片资源 path为路径  
def load\_image(path):  
 src=cv.imread(path)  
 return src  
#灰度拉伸方法   
'''  
它可以有选择的拉伸某段灰度区间以改善输出图像,如果一幅图像的灰度集中  
在较暗的区域而导致图像偏暗，可以用灰度拉伸功能来拉伸(斜率>1)物体灰度区间以改善图像；同样如果图像灰度集中在较亮的区域而导致图像偏亮，也可以用灰  
度拉伸功能来压缩(斜率<1)物体灰度区间以改善图像质量。  
灰度拉伸  
定义：灰度拉伸，也称对比度拉伸，是一种简单的线性点运算。作用：扩展图像的  
 直方图，使其充满整个灰度等级范围内  
公式：  
g(x,y) = 255 / (B - A) \* [f(x,y) - A],  
其中，A = min[f(x,y)],最小灰度级；B = max[f(x,y)],最大灰度级；  
 f(x,y)为输入图像,g(x,y)为输出图像  
缺点：如果灰度图像中最小值A=0，最大值B=255，则图像没有什么改变  
'''  
def gray\_stretch(image):  
 max\_value=float(image.max())  
 min\_value=float(image.min())  
 for i in range(image.shape[0]):  
 for j in range(image.shape[1]):  
 image[i,j]=(255/(max\_value-min\_value)\*image[i,j]-(255\*min\_value)/(max\_value-min\_value))  
 return image  
'''  
图像整体二值化  
'''  
def image\_binary(image):  
 max\_value=float(image.max())  
 min\_value=float(image.min())   
 '''  
 这里利用自适应二值化以及自动求阈值的方法明显效果不好。因此设置阈值这样设置的效果很容易想到，当图片为一张纯色图时阈值为对应像素值，当图包含  
 255与0时，阈值为122，总体的适应的效果会比较好。方法返回二值图  
 '''  
 ret=max\_value-(max\_value-min\_value)/2  
 ret,thresh=cv.threshold(image,ret,255,cv.THRESH\_BINARY)  
 return thresh  
  
'''  
矩形轮廓角点，寻找到矩形之后记录角点，用来做参考以及画图。  
'''  
def find\_rectangle(contour):  
 y,x=[],[]  
 for value in contour:  
 y.append(value[0][0])  
 x.append(value[0][1])  
 return [min(y),min(x),max(y),max(x)]  
  
'''  
车牌定位方法，需要两个参数，第一个是用来寻找位置，第二个为原图，用来绘制矩形。寻找位置的图片为经过几次形态学操作的图片。这里利用权值的操作，实  
现了定位的最高概率。  
'''  
def loacte\_plate(image,after):  
 *'''  
 定位车牌号  
 '''* #寻找轮廓  
 contours,hierarchy=cv.findContours(image,cv.RETR\_EXTERNAL,cv.CHAIN\_APPROX\_SIMPLE)  
 img\_copy = after.copy()  
 #找出最大的三个区域  
 solving=[]  
 for c in contours:  
 r=find\_rectangle(c)  
 '''  
 这里就算出面积和长宽比  
 '''  
 a=(r[2]-r[0])\*(r[3]-r[1])   
 s=(r[2]-r[0])/(r[3]-r[1])  
  
 solving.append([r,a,s])  
 #通过参考选出面积最大的区域  
 solving=sorted(solving,key=lambda b: b[1])[-3:]  
 #颜色识别  
 maxweight,maxindex=0,-1  
 for i in range(len(solving)):#  
 wait\_solve=after[solving[i][0][1]:solving[i][0][3],solving[i][0][0]:solving[i][0][2]]  
 #BGR转HSV  
 hsv=cv.cvtColor(wait\_solve,cv.COLOR\_BGR2HSV)  
 #蓝色车牌的范围 Hsv色彩空间的设置。  
 lower=np.array([100,50,50])  
 upper=np.array([140,255,255])  
 #利用inrange找出掩膜  
 mask=cv.inRange(hsv,lower,upper)  
 #计算权值用来判断。  
 w1=0  
 for m in mask:  
 w1+=m/255  
 w2=0  
 for n in w1:  
 w2+=n  
 #选出最大权值的区域  
 if w2>maxweight:  
 maxindex=i  
 maxweight=w2  
 return solving[maxindex][0]   
'''  
框出车牌 获取位置坐标，并返回图像  
'''  
#对图像的预处理  
def find\_plates(image):  
 image=cv.resize(image,(400,int(400 \* image.shape[0] / image.shape[1])))  
 #转换为灰度图像  
 gray\_image=cv.cvtColor(image,cv.COLOR\_BGR2GRAY)  
 #灰度拉伸  
 #如果一幅图像的灰度集中在较暗的区域而导致图像偏暗，可以用灰度拉伸功能来拉伸(斜率>1)物体灰度区间以改善图像；  
 # 同样如果图像灰度集中在较亮的区域而导致图像偏亮，也可以用灰度拉伸功能来压缩(斜率<1)物体灰度区间以改善图像质量  
 stretchedimage=gray\_stretch(gray\_image)#进行灰度拉伸，是因为可以改善图像的质量  
  
 '''进行开运算，用来去除噪声'''  
 #构造卷积核  
  
 kernel=cv.getStructuringElement(cv.MORPH\_ELLIPSE,(30,30))  
 #开运算  
 openingimage=cv.morphologyEx(stretchedimage,cv.MORPH\_OPEN,kernel)  
 #获取差分图，两幅图像做差 cv2.absdiff('图像1','图像2')  
 strtimage=cv.absdiff(stretchedimage,openingimage)  
  
 #图像二值化  
 binaryimage=image\_binary(strtimage)  
 #canny边缘检测  
 canny=cv.Canny(binaryimage,binaryimage.shape[0],binaryimage.shape[1])  
 #5 24效果最好  
 kernel=np.ones((5,24),np.uint8)  
 closingimage=cv.morphologyEx(canny,cv.MORPH\_CLOSE,kernel)  
 openingimage=cv.morphologyEx(closingimage,cv.MORPH\_OPEN,kernel)  
 #11 6的效果最好  
 kernel=np.ones((11,6),np.uint8)  
 openingimage=cv.morphologyEx(openingimage,cv.MORPH\_OPEN,kernel)  
 #消除小区域，定位车牌位置  
 rect=loacte\_plate(openingimage,image)#rect包括轮廓的左上点和右下点，长宽比以及面积  
 #展示图像  
 cv.imshow('image',image)  
 cv.rectangle(image, (rect[0]-5, rect[1]-5), (rect[2]+5,rect[3]+5), (0, 255, 0), 2)  
 cv.imshow('after', image)  
 cutimg = cut\_license(image, rect)  
 cv.imshow('cutimg', cutimg)  
 cv.waitKey(0)  
 cv.destroyAllWindows()  
def cut\_license(afterimg,rect):  
 *'''  
 图像分割函数  
 '''* #转换为宽度和高度  
 rect[2] = rect[2] - rect[0]  
 rect[3] = rect[3] - rect[1]  
 rect\_copy = tuple(rect.copy())  
 rect = [0, 0, 0, 0]  
 #创建掩膜  
 mask = np.zeros(afterimg.shape[:2], np.uint8)  
 # mask = np.astype('uint8')  
 #创建背景模型 大小只能为13\*5，行数只能为1，单通道浮点型  
 bgdModel = np.zeros((1, 65), np.float64)  
 #创建前景模型  
 fgdModel = np.zeros((1, 65), np.float64)  
 #分割图像  
 cv.grabCut(afterimg, mask, rect\_copy, bgdModel, fgdModel, 10, cv.GC\_INIT\_WITH\_RECT)  
 mask2 = np.where((mask == 1) | (mask == 0), 0, 1).astype('uint8')  
 img\_show = afterimg \* mask2[:, :, np.newaxis]  
 # 分割车牌与背景  
 return img\_show[rect\_copy[1]:rect\_copy[1] + rect\_copy[3], rect\_copy[0]:rect\_copy[0] + rect\_copy[2]]  
  
def runing():   
 file\_path='.\\plates'  
 for filewalks in os.walk(file\_path):  
 for files in filewalks[2]:  
 print('正在处理',os.path.join(filewalks[0],files))  
 find\_plates(load\_image(os.path.join(filewalks[0],files)))  
  
  
runing()