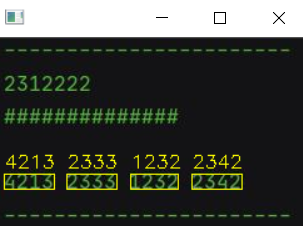
*'''  
2021.2.1FromIvicxDS:openCV;E7：数字识别  
'''***'''  
素材列表:  
0 1 2 3 4 5 6 7 8 9  
  
-----------------------  
this is a type of word  
\*\*\*\*\*  
4000 1234 5678 4020  
-----------------------  
  
-----------------------  
2312222  
##############  
  
4213 2333 1232 2342  
-----------------------  
  
-----------------------  
this is a type of word  
\*\*\*\*\*  
2301 2310 0000 3332  
-----------------------  
  
-----------------------  
2391 0012 2222 3321  
-----------------------  
  
-----------------------  
5514 3333 2222 1111  
-----------------------  
  
-----------------------  
1234 5678 9012 3456  
-----------------------  
'''**import cv2  
import numpy  
import E\_myutils  
def showPicture(name=**""**,picture=None):  
 *#图像的显示,也可以显示多窗口* cv2.imshow(name,picture)  
 *#在键盘中按任意键退出显示并向后执行语句  
 #cv2.waitKey(1000)表示只显示1秒* cv2.waitKey(0)  
 cv2.destroyAllWindows()  
  
accuracyX=0  
accuracyY=0  
resizedSize=(57,88)  
upFloatValue=5  
  
img=cv2.imread(**"image/E7/material.JPG"**)  
*#showPicture("material",img)#step1读取模板*ref=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  
*#showPicture("gray",ref)#step2转灰度图*ref=cv2.threshold(ref,100,255,cv2.THRESH\_BINARY)[1]*#返回值有两个  
#showPicture("tz",ref)#step3转为二值图*refCnts,hierachy=cv2.findContours(ref.copy(),cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_SIMPLE)  
*#RETR\_EXTERNAL表示只检测外轮廓  
#step4轮廓检测*cv2.drawContours(img,refCnts,-1,(0,255,255),1)  
*#showPicture("4",img)*refCnts=E\_myutils.sort\_contours(refCnts,method=**"left\_to\_right"**)[0]  
*#step5对轮廓按照从左到右排序*digits={}  
for (i,c) in enumerate(refCnts):  
 (x,y,w,h)=cv2.boundingRect(c)  
 roi=ref[y:y+h,x:x+w]  
 roi=cv2.resize(roi,resizedSize)  
 digits[i]=roi  
 *#showPicture(picture=roi)  
#step6将数字的方形区域与数字对应*rectKernel=cv2.getStructuringElement(cv2.MORPH\_RECT,(9,3))  
sqKernel=cv2.getStructuringElement(cv2.MORPH\_RECT,(5,5))  
*#指定卷积核大小*img=cv2.imread(**"image/E7/2.JPG"**)  
*#showPicture(picture=img)*gray\_img=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  
*#showPicture(picture=gray\_img)  
#step1读取要识别的图，进行灰度处理*tophat\_img=cv2.morphologyEx(gray\_img,cv2.MORPH\_TOPHAT,rectKernel)  
*#showPicture(picture=tophat\_img)#step2滤除一些杂点，突出重点*gradX=cv2.Sobel(tophat\_img,ddepth=cv2.CV\_32F,dx=1,dy=0,ksize=-1)  
gradX=numpy.absolute(gradX)  
(minVal,maxVal)=(numpy.min(gradX),numpy.max(gradX))  
gradX=(255\*((gradX-maxVal)/(maxVal-minVal)))  
gradX=gradX.astype(**"uint8"**)  
*#showPicture(picture=gradX)*gradX=cv2.morphologyEx(gradX,cv2.MORPH\_CLOSE,rectKernel)  
*#step3执行闭操作：先膨胀再腐蚀【有些细节膨胀后腐蚀不掉】，是图像更像一块一块的  
#showPicture(picture=gradX)*thresh=cv2.threshold(gradX,0,255,  
 cv2.THRESH\_BINARY|cv2.THRESH\_OTSU)[1]  
*#step4自动按照图像特征二值化图像【0,255】的原因cv2.THRESH\_OTSU  
#showPicture(picture=thresh)*thresh=cv2.morphologyEx(thresh,cv2.MORPH\_CLOSE,sqKernel)  
*#step5再进行一次闭操作，进一步去除图像中的空点  
#showPicture(picture=thresh)*threshCnts,hierarchy=cv2.findContours(thresh.copy(),cv2.RETR\_EXTERNAL,  
 cv2.CHAIN\_APPROX\_SIMPLE)  
*#step6寻找轮廓*cnts=threshCnts  
cur\_img=img.copy()  
cv2.drawContours(cur\_img,cnts,-1,(0,255,255),1)  
*#showPicture(picture=cur\_img)*locations=[]  
for (i,c) in enumerate(cnts):  
 (x,y,w,h)=cv2.boundingRect(c)  
 ar=w/float(h)*#计算长宽比* if ar>2 and ar<4:  
 locations.append((x,y,w,h))  
*#step7根据长宽比判断哪些轮廓要保留*locations=sorted(locations,key=lambda x:x[0])  
*#step8将这几个大轮廓从左到右排序*output=[]  
for(i,(gX,gY,gW,gH)) in enumerate(locations):  
 groupOutput=[]  
 group=gray\_img[gY-accuracyY:gY+gH+accuracyY,gX-accuracyX:gX+gW+accuracyX]  
 *#showPicture(picture=group)  
 #提取出每个组* group=cv2.threshold(group,0,255,cv2.THRESH\_BINARY|cv2.THRESH\_OTSU)[1]  
 *#showPicture(picture=group)  
 #二值化* digitCnts,hierarchy=cv2.findContours(group.copy(),cv2.RETR\_EXTERNAL  
 ,cv2.CHAIN\_APPROX\_SIMPLE)  
 *#计算轮廓* digitCnts=E\_myutils.sort\_contours(digitCnts,  
 method=**"left\_to\_right"**)[0]  
 *#轮廓排序* for i in digitCnts:  
 (x,y,w,h)=cv2.boundingRect(i)  
 roi=group[y:y+h,x:x+w]  
 roi=cv2.resize(roi,resizedSize)  
 *#showPicture(picture=roi)* scores=[]  
 for(digit,digitROI) in digits.items():*#匹配* result=cv2.matchTemplate(roi,digitROI,  
 cv2.TM\_CCOEFF\_NORMED)  
 score=cv2.minMaxLoc(result)[1]  
 scores.append(score)  
 groupOutput.append(str(numpy.argmax(scores)))  
 print(scores)  
 cv2.rectangle(img,(gX-accuracyX,gY-accuracyY),  
 (gX+gW+accuracyX,gY+gH+accuracyY),(0,255,255),1)  
 cv2.putText(img,**""**.join(groupOutput),  
 (gX,gY-upFloatValue),  
 cv2.FONT\_HERSHEY\_SIMPLEX,0.5,(0,255,255),1)  
 output.extend(groupOutput)*#一组一组算*print(output)  
showPicture(picture=img)  
>

*'''  
2021.2.1:轮廓排序  
'''*import cv2  
  
def sort\_contours(cnts,method=**"left\_to\_right"**):  
 boundingBoxes=[cv2.boundingRect(i) for i in cnts]  
 *#做轮廓的外接矩形，求左上角坐标排序  
 #返回一个元组(x,y,h,w)* **'''  
 print(str(boundingBoxes))  
  
 x\_first=[boundingBoxes[i][0] for i in range(len(cnts))]  
 y\_first=[boundingBoxes[i][1] for i in range(len(cnts))]  
 result=[]  
  
 if method=="left\_to\_right":  
 result=x\_first.sort()  
 elif method=="right\_to\_left":  
 result=x\_first.sort(reverse=True)  
 elif method=="top\_to\_bottom":  
 result=y\_first.sort()  
 elif method=="bottom\_to\_top":  
 result=y\_first.sort(reverse=True)** **else:  
 print("ERROR>>")  
 return  
 return cnts,result  
 '''** reverse=False  
 i=0  
 if method==**"right\_to\_left"** or method==**"bottom\_to\_top"**:  
 reverse=True  
 if method==**"top\_to\_bottom"** or method==**"bottom\_to\_top"**:  
 i=1  
 (cnts,boundingBoxes)=zip(\*sorted(zip(cnts,boundingBoxes),  
 key=lambda b:b[1][i],reverse=reverse))  
 return cnts,boundingBoxes

<>

[-0.05364607274532318, -0.2645203471183777, -0.39649903774261475, 0.0027115060947835445, 0.8096084594726562, -0.136179119348526, 0.4108525514602661, -0.3552934229373932, -0.021433299407362938, -0.1468234360218048]

[0.042707428336143494, 0.4031142294406891, 0.6846742630004883, 0.12916265428066254, -0.3299455940723419, 0.063449926674366, -0.1249714121222496, 0.24160173535346985, 0.1440039873123169, 0.019077714532613754]

[-0.02424243837594986, 0.9033583402633667, 0.3067706525325775, 0.03328927978873253, -0.24325157701969147, -0.03409712389111519, -0.0410596989095211, 0.16012582182884216, 0.024601053446531296, 0.045254696160554886]

[0.012971470132470131, 0.19137240946292877, 0.020804407075047493, 0.6814792156219482, 0.027544010430574417, 0.33811306953430176, 0.43262606859207153, 0.01843164674937725, 0.2028648555278778, -0.08825971186161041]

[0.042707428336143494, 0.4031142294406891, 0.6846742630004883, 0.12916265428066254, -0.3299455940723419, 0.063449926674366, -0.1249714121222496, 0.24160173535346985, 0.1440039873123169, 0.019077714532613754]

[0.04994450509548187, 0.13789275288581848, 0.09712174534797668, 0.7244396209716797, -0.017605412751436234, 0.37449759244918823, 0.3846539556980133, 0.0005261909682303667, 0.19761903584003448, -0.09086531400680542]

[0.07214286923408508, 0.23703832924365997, 0.08651958405971527, 0.6869606971740723, -0.032295022159814835, 0.3683793246746063, 0.3875252902507782, 0.06705203652381897, 0.1893271654844284, -0.041374292224645615]

[0.06150553748011589, 0.2037203162908554, 0.07827658951282501, 0.7041563987731934, -0.056790970265865326, 0.3673911690711975, 0.37082821130752563, 0.054970934987068176, 0.17775605618953705, -0.03813581168651581]

[-0.02424243837594986, 0.9033583402633667, 0.3067706525325775, 0.03328927978873253, -0.24325157701969147, -0.03409712389111519, -0.0410596989095211, 0.16012582182884216, 0.024601053446531296, 0.045254696160554886]

[0.030348438769578934, 0.38178205490112305, 0.688876748085022, 0.12940345704555511, -0.3824603855609894, 0.06558416783809662, -0.17840169370174408, 0.2609422504901886, 0.12933595478534698, 0.03174784407019615]

[0.06150553748011589, 0.2037203162908554, 0.07827658951282501, 0.7041563987731934, -0.056790970265865326, 0.3673911690711975, 0.37082821130752563, 0.054970934987068176, 0.17775605618953705, -0.03813581168651581]

[0.042707428336143494, 0.4031142294406891, 0.6846742630004883, 0.12916265428066254, -0.3299455940723419, 0.063449926674366, -0.1249714121222496, 0.24160173535346985, 0.1440039873123169, 0.019077714532613754]

[-0.06717415153980255, 0.4737454056739807, 0.6686770915985107, 0.09631752967834473, -0.33880823850631714, 0.03456108644604683, -0.20212726294994354, 0.29647690057754517, 0.03173379600048065, -0.011959332972764969]

[0.06150553748011589, 0.2037203162908554, 0.07827658951282501, 0.7041563987731934, -0.056790970265865326, 0.3673911690711975, 0.37082821130752563, 0.054970934987068176, 0.17775605618953705, -0.03813581168651581]

[-0.04755401983857155, -0.20117081701755524, -0.34426000714302063, -0.01553888339549303, 0.8045121431350708, -0.15104594826698303, 0.4133634567260742, -0.38091713190078735, 0.03294460102915764, -0.2099490910768509]

[-0.08177696168422699, 0.48243004083633423, 0.6445682644844055, 0.05212554708123207, -0.33045247197151184, -0.009188362397253513, -0.2080652117729187, 0.3229677975177765, 0.012266471982002258, -0.004921946208924055]

['4', '2', '1', '3', '2', '3', '3', '3', '1', '2', '3', '2', '2', '3', '4', '2']

Process finished with exit code 0