# 基于opencv的图像数字识别

2023E8021682136李睿涵

## 选题意义及应用

现今社会，各式各样图片中的数字识别已成为一个基本的需求。数字识别在很多场景中都有应用，比如车牌识别、银行支票处理、邮政编码识别、文档数字化以及各种计数系统等。虽然深度学习的出现极大地提高了识别的准确率（相对于传统的边缘检测，模板匹配），但是一些特殊的低功耗场合可能仍然需要非深度学习的数字识别。本人的项目便基于这个想法以实现基于传统的边缘检测和模板匹配的图像数字识别项目。

## 算法与程序设计思路

给定一张图片，通过传统方法识别上面的数字需要以下几个步骤。

1:为每个数字创建模板，然后在图像中寻找与模板最匹配的部分。我选择类似下图这样的比较标准的模板。



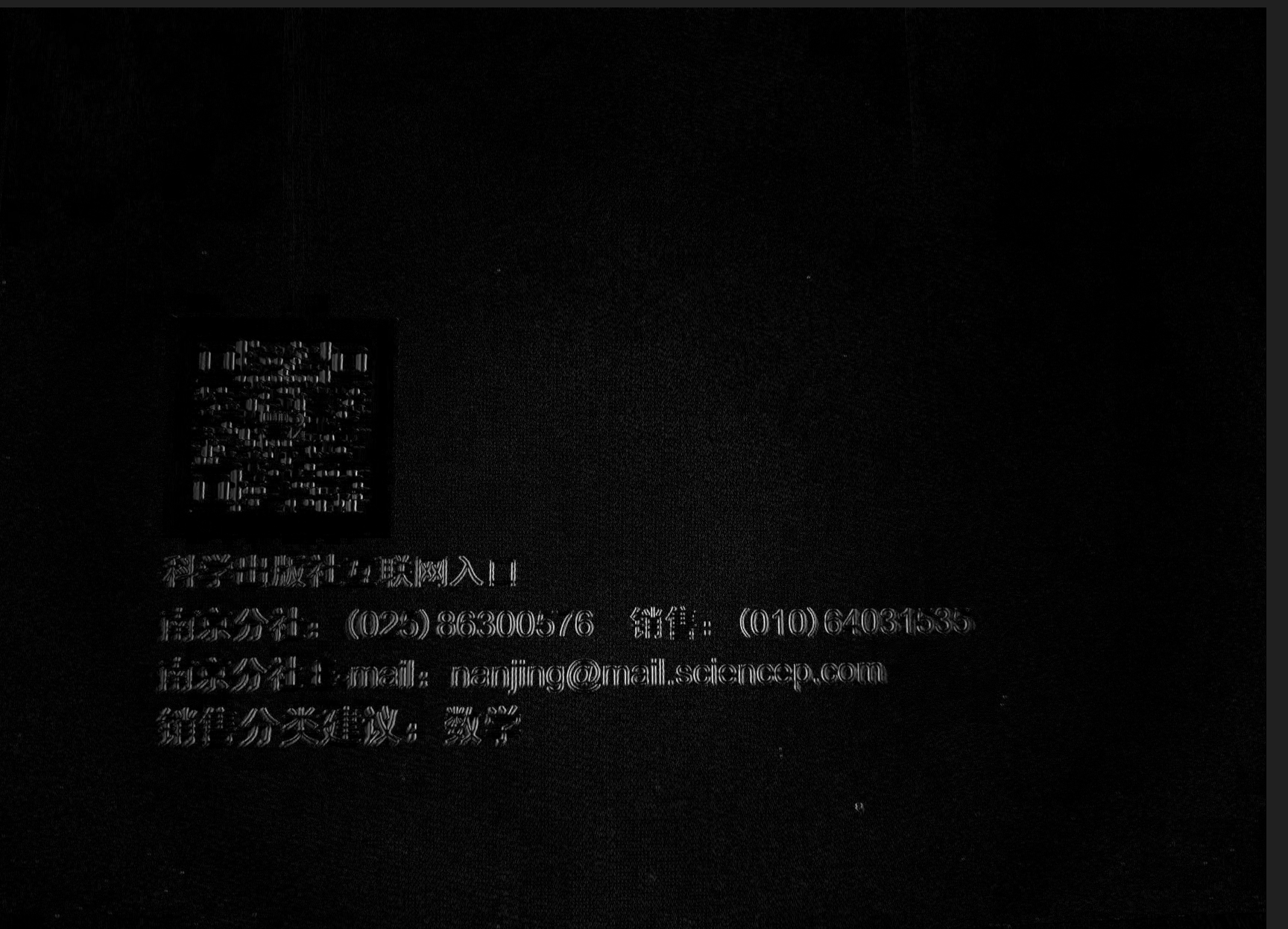
之后我们需要处理这个模板图片。先将其转换为二值图，之后使用opencv的轮廓检测函数cv2.findContours()检测到这十个数字的轮廓，效果如下：



之后建立起轮廓--数字的映射即可。

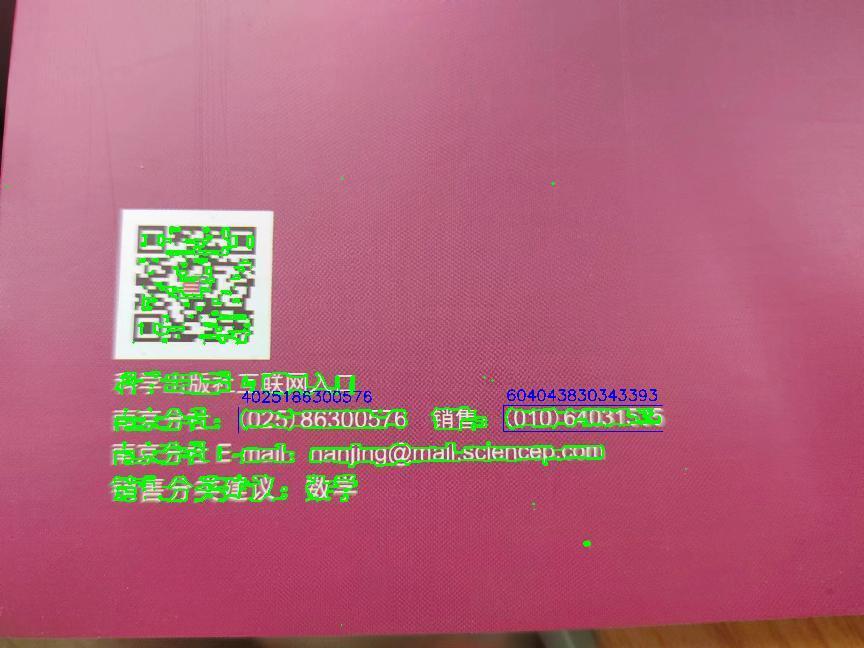
2：之后我们需要处理给定的需要识别的图像，首先我们需要将读取的图像转变为灰度图，并且更改为适合GUI的大小（比例尺不变），理论上我们需要对图像进行顶帽操作，从而突出明亮的区域，也就是转化为二值图后的数字区域。一般来说这是针对数字为白色时的操作，若数字为黑色，则需要进行黑帽操作。

3：之后我们需要进行边缘检测，从而将读取的图像上的各个对象的边缘凸显出来。边缘检测有多种算子可以选择，这里我们选择水平方向的Sobel算子来进行边缘检测即可（一般来说大多数数字是横向排列）。效果如下图：



4.之后我们仍然需要对图像进行处理。首先通过阈值可以有效地区分前景与背景，提取图像的主要特征，或是进行简单的图像二值化处理，便于后续的分析和处理。然后再通过闭操作扩大图像中的明亮区域，增加图像中连通区域的完整性，从而方便我们后续以区域为部分来识别数字。

5.通过算法识别到轮廓（如下图的绿色线框）后，由于算法本身无法直接判断框内的内容是不是数字，我们只能手动去分辩框中的内容。这里我们可以直接通过各个框的长度以及宽高比来判断此框中的内容是否是数字。判断是数字之后，通过将框内的轮廓与先前模板所建立的映射相关联即可识别出数字。具体识别效果如下图所示。

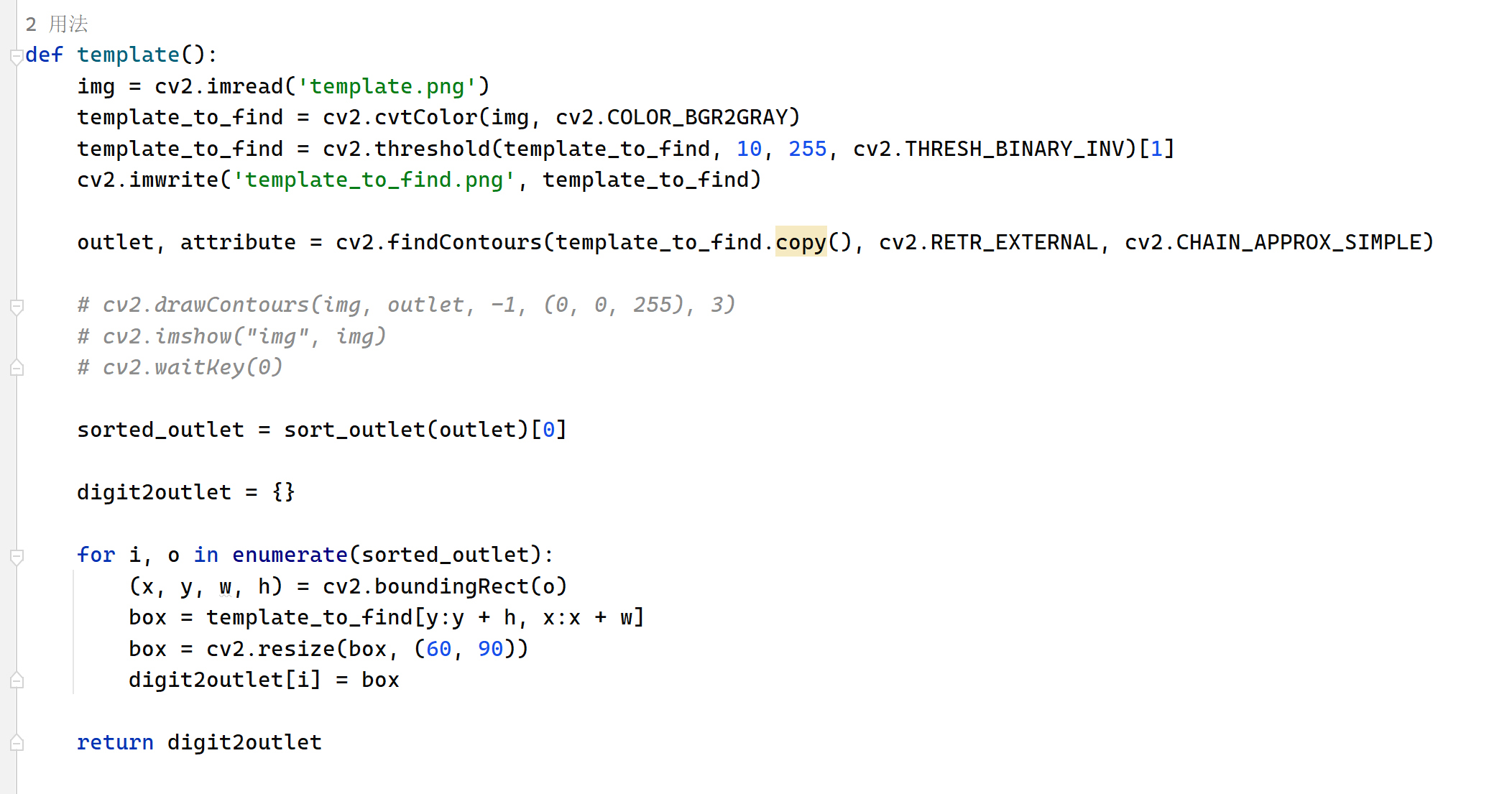


6.同时我考虑到了这个程序需要一个较为简单的GUI界面来进行操作，于是我选定Pyqt作为GUI框架来进行开发，具体UI如下图：

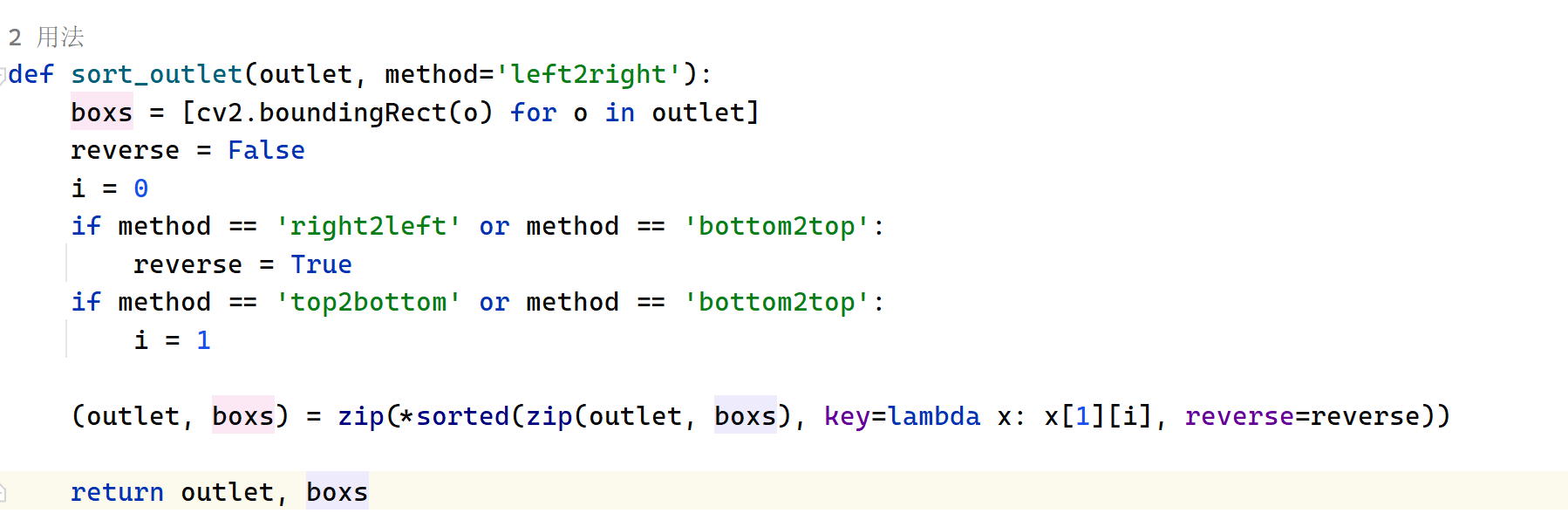


## 程序关键代码

template()函数的具体代码如下图所示：



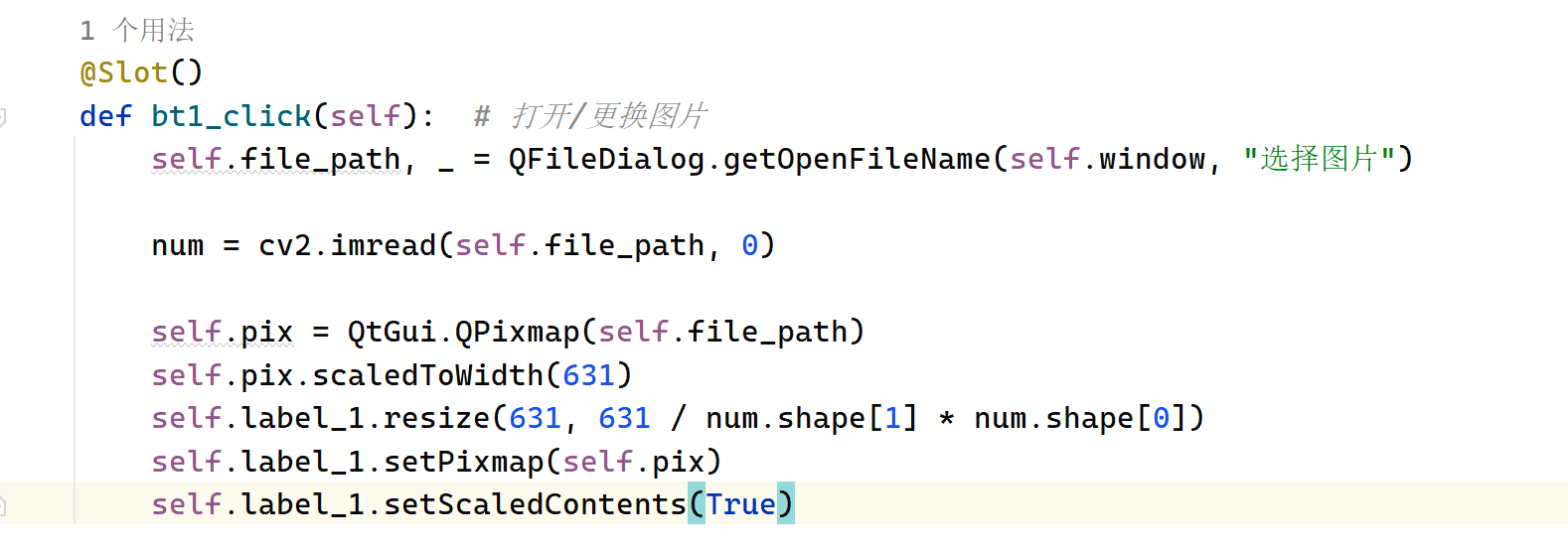
读取已经选择好的模板图片，将其转化为二值图并进行阈值处理，其中白色为要找的模板部分，黑色为背景。之后进行轮廓的检测，再通过sort\_outlet函数按图片中的数字方向（默认从左到右）进行排序。排好序列后建立数字-数字轮廓的映射数组即可。



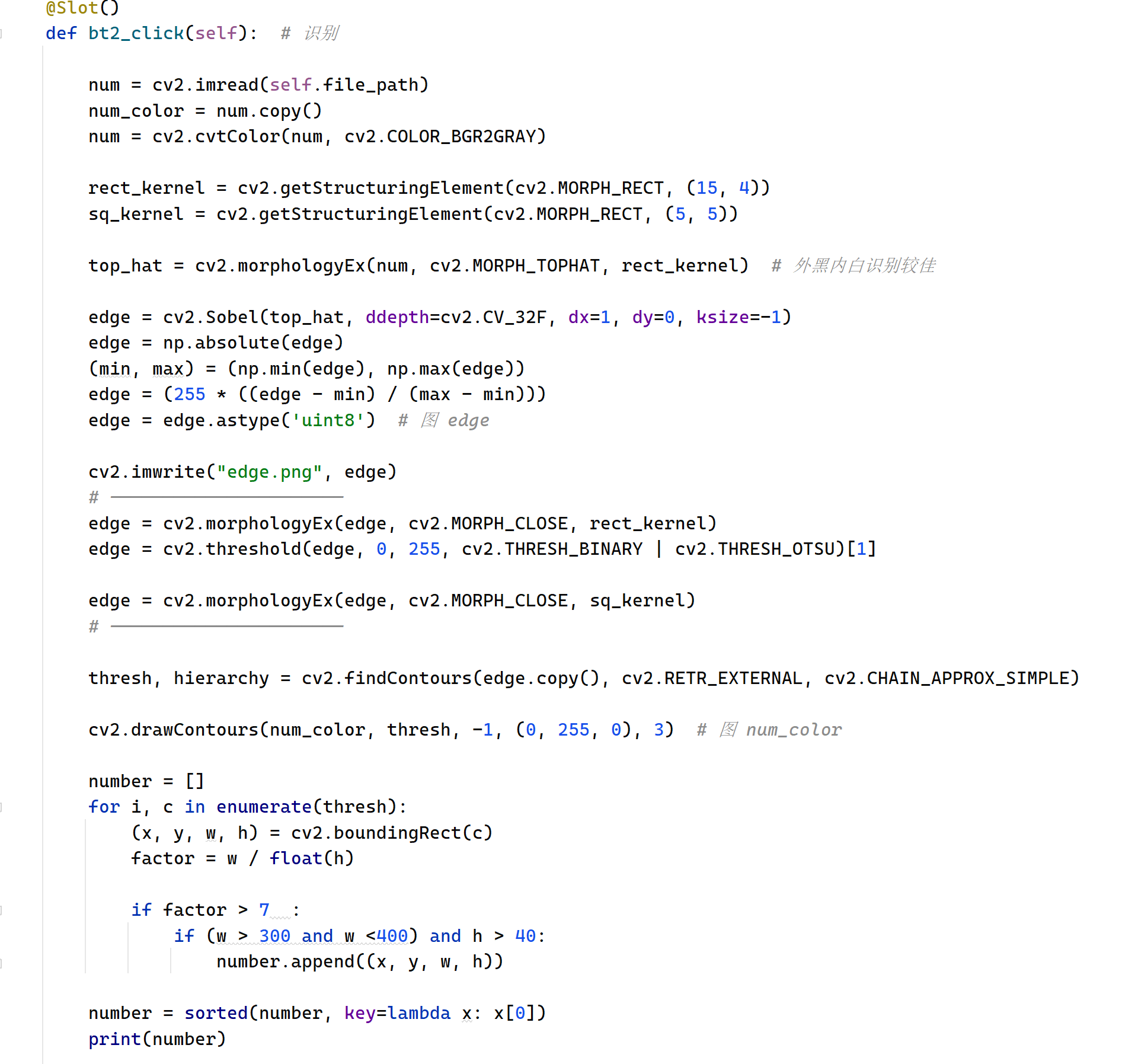
Sort\_outlet函数的排序如图所示，关键在于水平方向和垂直方向的排序选取。

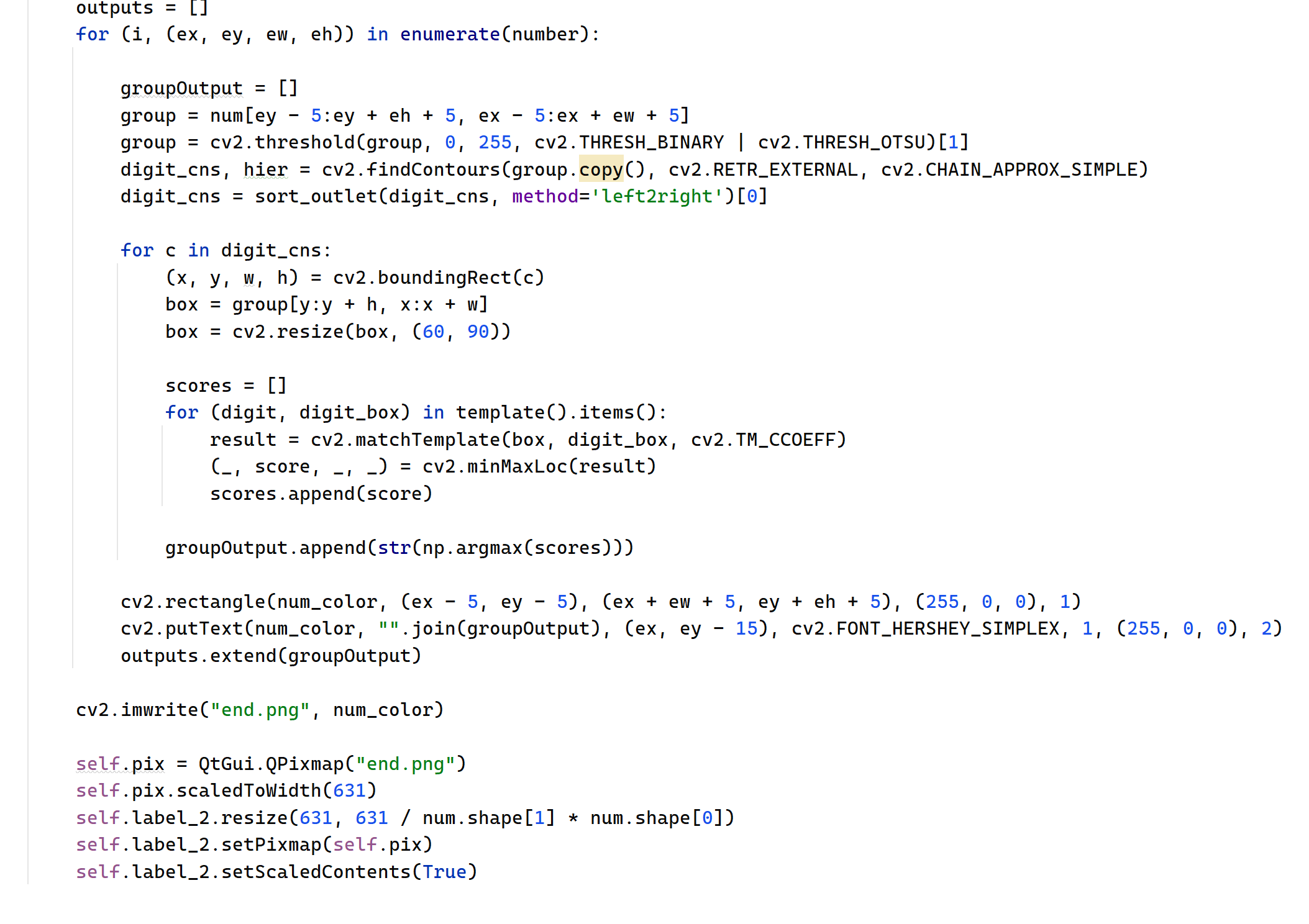
我们使用zip和sorted函数结合一个lambda表达式对outlet和boxs进行排序。key=lambda x: x[1][i]指定了排序的关键字，即每个边界框的第i个属性（根据之前设置可能是宽度或高度），reverse参数决定了排序是升序还是降序。

最后，使用zip(\*...)来解压缩排序后的元组，并返回排序后的outlet和对应的boxs。



与gui对应的打开图片的按钮的绑定函数如图。函数首先将选定的文件路径储存，之后通过opencv读取此文件，在图片宽度定死的前提下将比例尺保留，从而更自然地在GUI中显现出图像。坏处在于缩放后的图片可能会占用GUI垂直方向的其他部分。





识别按钮对应的代码较多。

首先函数读取指定路径 (self.file\_path) 的图像，并保留一份彩色副本 (num\_color) 用于后续绘制轮廓。之后将图像转换为灰度图像，并进行一系列形态学操作来突出数字区域，比如

使用矩形结构元素进行顶帽变换 (cv2.morphologyEx with cv2.MORPH\_TOPHAT) 来增强亮于背景的细节。计算边缘梯度并进行归一化，以便后续处理。同时函数也应用闭运算 (cv2.morphologyEx with cv2.MORPH\_CLOSE) 和阈值处理 (cv2.threshold) 来更好地提取数字的轮廓。

之后我们使用Sobel算子进行边缘检测，使用 cv2.Sobel 函数对 top\_hat 图像进行水平方向（dx=1, dy=0）的边缘检测，得到梯度图像 edge。ddepth=cv2.CV\_32F 指定输出图像的数据类型为32位浮点型，ksize=-1 使用Sobel算子的默认尺寸（3x3）。

接下来，对得到的边缘图像取绝对值，确保梯度值都是正的。然后，通过计算图像中的最小值和最大值，对边缘图像进行归一化，使其像素值范围映射到0-255之间，便于后续处理。

之后，程序使用两次应用形态学的闭运算(cv2.morphologyEx with cv2.MORPH\_CLOSE)，先使用矩形核(rect\_kernel)去除小缝隙，再使用方形核(sq\_kernel)进一步平滑边缘，使得边缘更连续且闭合，有助于后续的轮廓检测。

在第二次闭运算之前，使用全局阈值法（cv2.threshold with cv2.THRESH\_BINARY | cv2.THRESH\_OTSU）对边缘图进行二值化，其中 THRESH\_OTSU 参数会自动选择一个最佳阈值来二值化图像，这一步骤可以有效分离前景和背景。之后在保留的num\_color上绘制出识别出来的轮廓区域。

之后我们通过限定识别区域的宽度以及长宽比来筛选出数字区域。之后使用sorted()函数和lambda表达式按轮廓的x坐标对其进行排序，确保最终列表中的轮廓按照它们在图像中从左到右的顺序排列。

最后这段代码的主要目的是对之前识别出的每个数字区域进行细化处理，识别并组合其中的单个数字，然后在原图上绘制识别结果。

首先遍历之前筛选和排序好的数字区域列表number，对于每一个区域(ex, ey, ew, eh)提取该区域内的图像部分group，并对这部分进行二值化处理，使用cv2.threshold并选取阈值为cv2.THRESH\_OTSU自动获取最优阈值。

在此区域上再次查找轮廓digit\_cns，并使用之前定义的sort\_outlet函数按从左到右的顺序对这些轮廓进行排序。

对于排序后的每个轮廓c，提取其边界框内的图像box，并调整至统一大小(60, 90)。

遍历预先定义好的数字模板集（通过调用template()函数获取），使用cv2.matchTemplate函数计算当前数字图像box与每个数字模板之间的匹配分数。这里采用的是cv2.TM\_CCOEFF方法，它衡量的是相关系数，值越大表示匹配越好。

记录每个模板的匹配分数，然后找出最高分对应的索引，将此索引转换为对应数字字符并添加到groupOutput列表中。

最后在原始彩色图像num\_color上，使用红色边框标出当前数字区域，并在该区域上方用红色字体写上识别出的数字字符串（由groupOutput中的字符拼接而成）。

将当前数字区域识别出的数字字符串列表groupOutput扩展到总的输出列表outputs中。

## 程序运行结果与分析



运行结果如图所示，可以看到对于第一个数字方框程序处理的还算不错，只有括号识别成了数字。第二个方框的处理结果较差，可以看出来大部分的数字识别的都是错误的。主要原因在于识别的轮廓问题以及拍摄的照片的清晰度问题。

## 问题总结与展望

这个程序有如下几个问题：GUI界面过于简陋，识别数字前需要手动指定参数来识别相应的数字区域以及数字识别正确率较低。之后我们可以使用QSS来美化这个程序的GUI,同时也可以使用基于深度学习的方法来识别图中的数字，从而更好地提高识别率的同时也解决识别区域的问题。