2021-2022 学年度第一学期

《人工智能基础》程序设计

班级：<u>计 1901</u> 学号：<u>195150116</u> 姓名：<u>靳子恒</u>

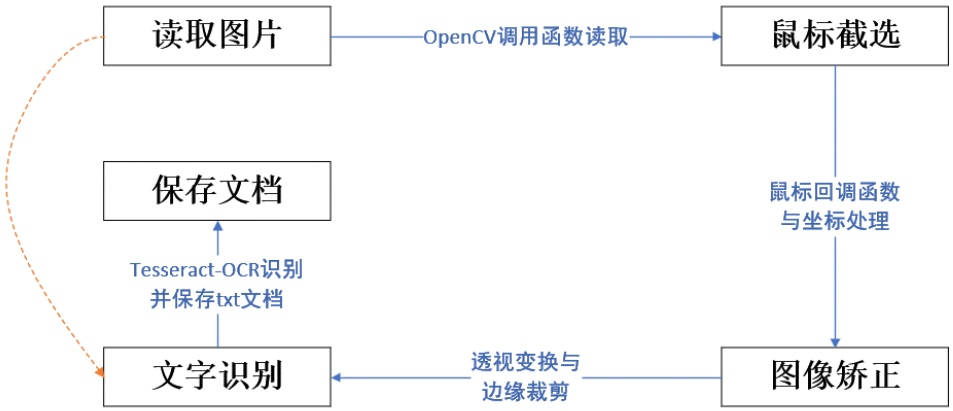
一、 设计内容

文字识别——基于 OpenCV 和 Tesseract-OCR，实现图像截取矫正并进行文字识别

二、 实现过程

（一）.设计思路

利用生活中文字识别的应用场景的启发，模仿一个相似的文字识别程序。



通过 OpenCV 的 imread 和 imshow 函数读取图片并创造一个 GUI 窗口进行图像展示，然后利用鼠标回调函数和鼠标事件（鼠标左键双击）实现鼠标双击绘图截选想要识别的文档部分。

再之后利用 OpenCV 的透视变换里矩阵运算 getPerspectivetransform 函数和 warpPerspective 函数共同完成图像的矩阵运算与转换，与此同时对图像边界进行裁剪，以达到在图像截取过程中不得已截取到文档以外部分的裁剪的目的。

最后使用谷歌的 Tesseract-OCR 引擎进行文字识别，与此同时，对识别内容进行保存，设定好存储路径，保存为 txt 文件。

（二）.实现过程

1. 定义鼠标回调函数并收集源图像的四个顶点坐标点

```
def points_collect(event,x,y,flags,param):
    dic_points = param
    if event == cv2.EVENT_LBUTTONDBLCLK:
        if len(dic_points['ps'])>=4:
            dic_points['ps']=[]
            dic_points['ps'].append((x,y))
        else:
            dic_points['ps'].append((x,y))
    if event == cv2.EVENT_MOUSEMOVE:
        dic_points['p_move']=(x,y)
if __name__ == "__main__":
    file_scan = "C:\\Users\\Forward\\Desktop\\python-ai-master\\Sca
    dic_points = {}
    dic_points["ps"]=[]
    dic_points["p_move"]=()
    cv2.namedWindow('image',cv2.WINDOW_NORMAL)
    cv2.setMouseCallback('image', points_collect,param=dic_points)
```

解析：定义一个鼠标回调函数，利用鼠标响应进行图像截取这一事件的触发。图像截取为调用 OpenCV 绘图函数在下一部分，此处是定义 event 事件为鼠标左键双击 LBUTTONDBLCLK 事件。

当鼠标左键双击后，开始描点绘图截取图像矫正部分并进行鼠标坐标点的记录。

因图像截取为四个坐标点，截取梯形图，因此定义名为 dic_points[]的列表变量，在此函数部分对鼠标双击事件开始后鼠标坐标点的收集。

在绘图过程中，考虑到鼠标点第一个点之后需要显示连线，因此鼠标在移动过程中的坐标也要纳入 dic_points 列表内，为下一步点与点之间连线做好准备，移动的坐标即动点，利用 append 函数将 move 动点坐标附加到 dic_points 列表内。

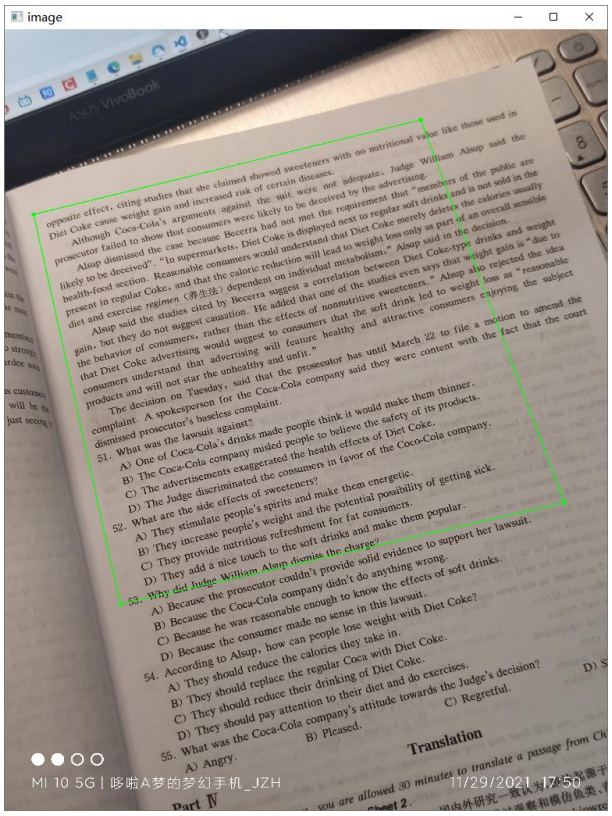
其中，函数参数 param 作用是将定义好的鼠标坐标列表 dic_points 赋给 param，在主函数 setmouseback 部分进行参数回调。

2. 在回调函数基础上进行选点并绘图

```
def drawlines(img,dic_points):
    color = (0,255,0)
    points = dic_points['ps'][:]
    points.append(dic_points['p_move'])

    if len(points)>0 and len(points)<5:
        for i in range(len(points)-1):
            cv2.circle(img,points[i],15,color,cv2.FILLED)
            cv2.line(img,points[i],points[i+1],color,6)
    elif len(points)>=5:
        for i in range(3):
            cv2.circle(img,points[i],15,color,cv2.FILLED)
            cv2.line(img,points[i],points[i+1],color,6)
        cv2.circle(img,points[3],15,color,cv2.FILLED)
        cv2.line(img,points[3],points[0],color,6)
```

解析：此函数就是上一步中所述的 OpenCV 绘图函数，实现过程中起到描点连线，框选想要截选图像部分的功能。该功能是在第一步鼠标响应函数的基础上，当鼠标左键双击，触发事件，开始绘图并记录坐标。如图所示：



OpenCV 描点连线颜色 color 设置为（0,255,0）纯绿色。Circle 函数画圆，然后用 Filled 函数进行填充，已达到画点的目的。Line 函数用于画线，画线函数内利用 point(i)和 point(i+1)函数实现当前点与下一个点的连接，当然此处也间接解释了为什么第一步鼠标回调里要附加（append）鼠标动点坐标到 dic_points[]列表里。

在函数 if-elif 循环部分是考虑到以下情况设置 0-5 区间范围和>=5 的判定值：1.除去鼠标四个固定顶点外还要考虑动点，动点算作第五个坐标点，在鼠标截选图像至要截选最后一个时已经有了四个点（3 顶点 1 动点），此时执行的是 if 语句内的代码；当画第四个顶点的时候，实际有 5 个点，这时候执行 elif 语句，将第四个点和初始点之间建立连接。2.当描完四个顶点并连线后，如果对框选部分不满意，那么再次双击，此时和上一部鼠标响应函数有关，动点+四个顶点+额外双击时，坐标数已经大于 4，那么执行清空数组语句 dic_points[‘ps’]=[]，重新开始绘图。

3. 将绘图中收集到的四个顶点进行排序

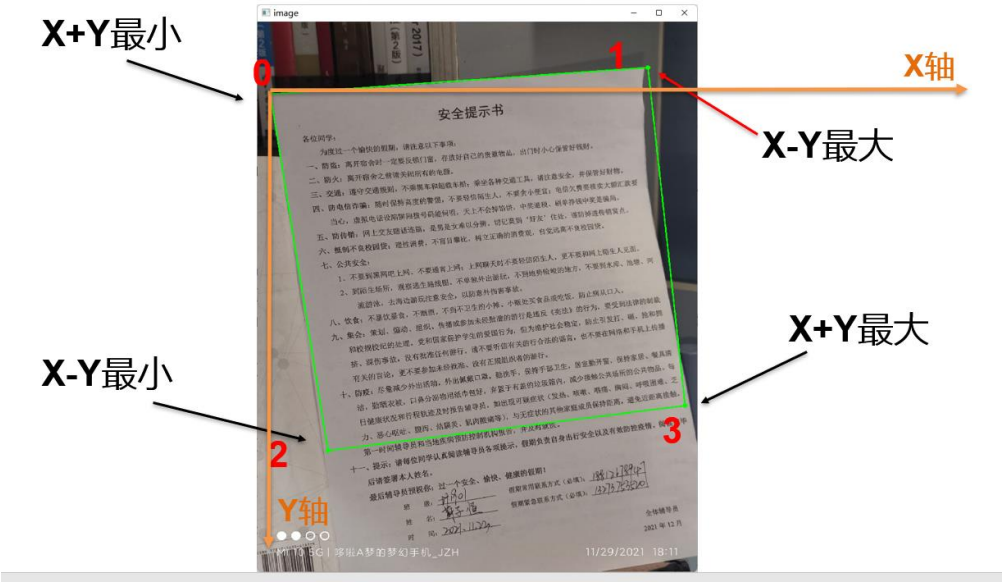
```
def reorder(points):
    points = np.array(points)
    ordered_points = np.zeros([4,2])

    add = np.sum(points,axis=1)
    ordered_points[0] = points[np.argmin(add)]
    ordered_points[3] = points[np.argmax(add)]

    diff = np.diff(points,axis=1)
    ordered_points[1] = points[np.argmin(diff)]
    ordered_points[2] = points[np.argmax(diff)]
    return ordered_points
```

解析：此处对收集好的四个顶点坐标排序是为下一步透视变化做好准备，之所以要坐标排序是因为，我们在截选图像的时候

顺序不一样，可能逆时针可能顺时针，可能从左下开始描点，可能从右上开始描点等等，计算机并不知道这些坐标的顺序，没有个很正的图像的标准，也就无法对截取的图像进行矫正。因此要对坐标进行一个固定的位置排序，排序原理如下图所示：



由此图应该更好理解，坐标排序的依据。定义了两个算法：add 和 diff，前者是计算二维数组（存放好的四个顶点坐标）内 x+y 的值，后者是计算二维数组内 x-y 的值。计算好后，利用 argmin 和 argmax 函数对计算出的最大最小值的 x y 进行索引。

左上角 X+Y 一定是最小的，argmin（add）将索引的那个坐标重新赋给新的数组 orded_point[]里，并放在下标为 0 的位置，以此类推，其他不再赘述。

其中 zero（[4,2]）是创建一个用 0 填充的二维数组，即：[[0,0],[0,0],[0,0],[0,0]],为上述坐标排序做前提准备。

4. 使用透视变换进行图像矫正，并对图像边缘裁剪

```python
def getWarp(img,ordered_points,size_wraped):
    w,h = size_wraped
    p1 = np.float32(ordered_points)
    p2 = np.float32([[0,0],[w,0],[0,h],[w,h]])#float32单精度浮点

    matrix = cv2.getPerspectiveTransform(p1, p2)
    imgOutput = cv2.warpPerspective(img, matrix, (w, h))
    imgCropped = imgOutput[20:imgOutput.shape[0]-20,20:imgOutput.shape[1]-20]
    imgCropped = cv2.resize(imgCropped,(w,h))
    return imgCropped
```

解析：此处是图像校正函数 getwarp。在前三步的基础上，获得源坐标点和目标坐标点，再利用 getPersepectiveTransgorm 函数进行透视矩阵的运算，运算完成后，图像用计算好的结果进行矫正，即 warpPerspective 函数。

此时图像已经矫正完成，再对图像边界进行裁剪，以达到在图像截取过程中不得已截取到文档以外部分的裁剪的目的。用到的函数：imgOutput[20:x-20,20:y-20],他表示对图像截取 x 轴上从第 20 像素开始到 x-20 像素的那一部分，y 轴同理，以此达到图像边界各裁剪 20 像素的目的。

在原代码中还有 shape[0]和 shape[1]函数，前者表示获取图像的高度，后者表示图像的宽度，如果参数是 2 表示图像的通道数。最后 resize 函数将图像矫正后的最终成品赋给变量 imgCropped，供主函数调用。

5. 主函数部分：调用前四步定义好的函数，OpenCV 进行图像路径读取、实现文字识别与识别内容的存储

```python
size_wraped = (960,1020)
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\
while True:
    img = cv2.imread(file_scan)
    drawlines(img,dic_points)
    cv2.imshow('image',img)
    key=cv2.waitKey(100)# & 0xFF
    if key == ord('q'):
        break
    if key == ord('w'):
        key = 0
        if len(dic_points['ps'])==4:
            ordered_points = reorder(dic_points['ps'])
            img_Warped = getWarp(img,ordered_points,size_wraped)
            cv2.imshow("ImageWarped",img_Warped)
            imgWarped_RGB = cv2.cvtColor(img_Warped, cv2.COLOR_BGR2RGB)

            txt = pytesseract.image_to_string(imgWarped_RGB)
            print(txt)
            text_path='C:\\Users\\Forward\\Desktop\\python-ai-master\\Scann
            file_handle=open(text_path,mode='w',encoding= 'utf8')
            file_handle.write(txt)
            file_handle.close()
            result=[text_path]
            result.append(txt)
```

解析：此处是主函数部分，size_wrapped 是设置第四部我们想要设置的矫正后的图像的宽高比。第二句是导入谷歌的 tesseract 语言识别库，是谷歌已经训练好的语言识别包，正因是谷歌的 ocr，因此对中文的识别率并不高。在下边用 pytesseract.imag_to_string 语句实现为图像的文字识别。

其中 waitkey 函数是设置 OpenCV 展示图像函数执行后展示图像的 GUI 的滞留时间，单位是 ms。但后又加了一句& 0xFF，它的作用是不以时间为 GUI 窗口滞留的标准，而是以按键来判断，下边接着，如果是 q 按键则 break 关闭窗口，如果是 w 实现文字识别。

最后一部分就是创建一个保存识别内容的路径，文字识别后将识别内容进行一个存储，存储格式为 txt 文件，命名为 ScannerTxt。

至此，程序语句全部执行完毕！

三、 相关算法

1. 透视变换算法

透视变换（Perspective Transformation）就是将二维的图片投影至一个三维的视平面上，然后再转换到二维坐标下，所以也称为投影映射（Projective Mapping）或者透射变换。简单来说就是二维→三维→二维的一个过程。在很多计算机视觉领域会用到此算法。透视变换矩阵如下所示：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

其中，X，Y，Z 代表透射变换后的三个坐标，x、y 代表透射变换前的 2 维坐标

可以得到公式形式：

$$\begin{cases} X = & a_1x + b_1y + c_1 \\ Y = & a_2x + b_2y + c_2 \\ Z = & a_3x + b_3y + c_3 \end{cases}$$

因为计算出后是一个 3 维坐标，所以我们要利用 Z 将值转换到 2 维坐标中：

$$\begin{cases} x' = \dfrac{X}{Z} = \dfrac{a_1x + b_1y + c_1}{a_3x + b_3y + c_3} \\ y' = \dfrac{Y}{Z} = \dfrac{a_2x + b_2y + c_2}{a_3x + b_3y + c_3} \end{cases}$$

于是 x‘和 y’是二位透透射变换的最终计算结果

其中 c3=1，计算矩阵参数：可以得到通式如下

$$\begin{cases} -a_1x & +a_3xx' - b_1y & +b_3yx' & -c_1 & = -1 \\ -a_2x & +a_3xy' & -b_2y + b_3yy' & -c_2 & = -1 \end{cases}$$

将 4 个点带入通式组成线性方程组，则可用克拉默法则计算出相应参数。

四、 心得体会

1. 通过对人工智能-计算机视觉领域里文字识别的轻研究和学习，让我窥见了人工智能的“智能”之处和算法之难，以及 python 之美。

2. 在学习过程中，让我对 java、c 和 python 不同高级语言间的关系和相似之处理解的更加深刻。

3. 认识和了解了将高级语言应用到实际场景的一个代码实现的过程，以往都是进行数学类计算，很少应用到实际场景。通过此次的学习和锻炼，让我对 python 的简单之处和高级之处有了新的和更深的体会。

4. 除去研究和编写代码，我在一开始对 python 整个环境的配置过程（包括 linux）中和不同高级语言编写工具的练习（vscode、linux

的 vim、pycharm）使用上有了新的认识并且熟练掌握，也增强了环境配置和理解的能力，结合操作系统知识，对我自身整体知识框架有了新的加强和体会。

5. 总结：收获颇深，也有了对自身学习能力的一个判断，希望在将来能够不断突破。