基于Qt库的智慧树题目OCR识别搜索

刘旭锋

信息与通信工程学院，2018210509

摘要：基于Qt库的扫描与识别，将所选图片上传至百度OCR进行图片识别，并将扫描的文字结果返回到界面上，对题目在智慧树题库中进行查询，返回结果，以实现题目答案的搜查。

关键字：Qt库，gui

1 引言

智慧树是全球大型的学分课程运营服务平台。服务的会员学校近3000所，已有超过1700万大学生通过智慧树网跨校修读并获得学分。智慧树网帮助会员高校间，实现跨校课程共享和学分互认，完成跨校选课修读。目前很多大学中都有在智慧树上的网课，在网课中会有随堂测试、章节测试和最后的期末考试。

程序主要解决智慧树的题目查询问题并自动获得答案，主要包括以下几个环节：

1. 截屏获取图片
2. 将扫描的图片上传至百度api
3. 通过百度api扫描得到文本结果，并返回
4. 将返回的结果在智慧树题库中进行扫描，将答案返回

依靠以上几个过程来完成对题目的识别与答案的获取。代码可以参考github上的程序，网址: <https://github.com/architect-of-thought/-python->。

2．问题的提出

目前很多大学中都有在智慧树上的网课，在网课中会有随堂测试、章节测试和最后的期末考试。而考试自然就会涉及到成绩的问题，智慧树上的题目具有随机性，在网络上自动搜索的答案为比准确，有的甚至是错误的。

综合以上的情况，智慧树本身很重要，有没有一款合适的刷成绩查答案的软件，与是便着手准备以此为问题，写一段程序已解决此问题。

3．需要的准备

本代码主要基于Qt库和百度API的OCR（Optical Character Recognition，光学字符识别）技术来实现。因此需要提前安装Qt库以及一些其他的库。

主要有：PySide2 requests baidu-aip slot

4．具体代码的解释

4.1 库准备与置入

import sys

import time

import requests

from PySide2.QtCore import QRect, QSize, QPoint, Signal, Slot, Qt, QByteArray, QBuffer

from PySide2.QtWidgets import QWidget, QApplication, QRubberBand

from PySide2.QtGui import QMouseEvent, QPixmap, QPixelFormat

from mainwin\_ui import Ui\_MainForm

from settingswin\_ui import Ui\_SettingsForm

from aip import AipOcr

4.1.1 Qt库与PySide2

Qt是一个极其强大的用来构建图形界面的库，针对很多的编程语言都有支持。主要用于构建可视化的程序，如可交互的窗口。

而PySide2是Qt为了让Python也能使用Qt，产生了一个项目**Qt for Python**，在Python里这个模块的名字是**PySide2**。与PySide不同的是，PySide是基于Qt4构建，PySide2基于Qt5构建。显然，Pyside2是新版本。

4.1.2 requests

requests是用**python**语言编写，基于urllib，采用 Apache2 Licensed 开源协议的 HTTP 库。比urllib更加方便。主要用途在于向网页发送申请，主要包括：GET请求、POST请求、PUT请求、DELETE请求、HEAD请求、OPTIONS请求

4.1.3 aip (baidu-aip)

与百度对接的aip（图像识别）库

4.1.4 mainwin.ui与settingwin.ui

为代码完成后自动生成的主界面程序，主要包含Ui.mainForm与Ui.SettingForm两个对象，在后续会有类的实例化。

4.2 截屏的操作与相关函数

class Scrot(QWidget):

scrotFinished = Signal(QPixmap)

def \_\_init\_\_(self, parent: QWidget = None):

super(Scrot, self).\_\_init\_\_(parent=parent)

self.rubberBand: QRubberBand = None

self.origin: QPoint = QPoint()

self.screen = QApplication.primaryScreen()

self.setWindowOpacity(0.1)

self.setAttribute(Qt.WA\_TranslucentBackground)

def mousePressEvent(self, event: QMouseEvent):

self.origin = event.pos()

if self.rubberBand is None:

self.rubberBand = QRubberBand(QRubberBand.Rectangle, self)

self.rubberBand.setGeometry(QRect(self.origin, QSize()))

self.rubberBand.show()

def mouseMoveEvent(self, event: QMouseEvent):

if self.rubberBand is not None:

self.rubberBand.setGeometry(QRect(self.origin, event.pos()).normalized())

def mouseReleaseEvent(self, event: QMouseEvent):

if self.rubberBand is not None:

self.rubberBand.hide()

self.close()

time.sleep(0.6)

rect = (min(self.origin.x(), event.x()), min(self.origin.y(), event.y()), abs(event.x() - self.origin.x()),

abs(event.y() - self.origin.y()))

screenShot = self.screen.grabWindow(QApplication.desktop().winId(), \*rect)

self.scrotFinished.emit(screenShot)

def run(self):

self.showFullScreen()

4.2.1 初始化（init）

首先将创建的窗口中所有有关参量设为0或者none，self.setWindowOpacity(0.1)将窗口的透明度设为0.1而不是0的原因是因为在后续的操作中，如果透明度为零将无法截取图片的信息和内容，具体原因并未探究出结果。因此为了让沟洫的操作能够有效地进行，将其设置为0.1。

4.2.2 鼠标点击（mousePressEvent）

具体操作是将起点设置为鼠标点击的点。 rubberband类为一种透明的widght一用于鼠标跟踪，是其通过正方形来跟踪，并显示形状，具体表现为鼠标变为截图的形状，可以开始截图。

4.2.3 鼠标移动（mouseMoveEvent）

主要使rubberband跟踪鼠标的移动即可。

4.2.4 鼠标松开（mouseReleaseEvent）

主要是将rubberband隐藏并关闭，将rubberband所追踪的区域转化为截取获得的的图片，退出并将所得图片保存。

4.3 窗口设置及相关函数

class SettingsWin(QWidget):

apiConfigured = Signal(tuple)

def \_\_init\_\_(self, api\_id: str, api\_key: str, secret\_key: str, parent=None):

super(SettingsWin, self).\_\_init\_\_(parent=parent)

self.ui = Ui\_SettingsForm()

self.ui.setupUi(self)

self.setFixedHeight(self.height())

self.ui.lineEdit.setText(api\_id)

self.ui.lineEdit\_2.setText(api\_key)

self.ui.lineEdit\_3.setText(secret\_key)

self.ui.assertButton.setDisabled(True)

self.ui.pushButton.clicked.connect(self.ok)

self.ui.pushButton\_2.clicked.connect(self.cancel)

def ok(self):

self.apiConfigured.emit((self.ui.lineEdit.text(), self.ui.lineEdit\_2.text(), self.ui.lineEdit\_3.text()))

self.close()

def cancel(self):

self.close()

4.3.1初始化（\_init\_）

首先通过self.ui = Ui\_SettingsForm()将主界面实例化，然后调用setupUi。之后设置窗口的相关参数。将高度设置为原本窗框的高度。同时对窗口的内容设置成api相关的的api key，api id，secret key。之后设置窗口的按键内容。

第一个按键：self.ui.assertButton.setDisabled(True)为断言按键，集如果（）内的条件返回为false则终止程序的运行。此函数为必备函数，在程序中没有用处，因此将（）内天些为Ture即可，这条语句本身可以用作程序的调试。

第二个按键：self.ui.pushButton.clicked.connect(self.ok) 此按键即为向api识别网站发送自己所截取的图片，并进行识别的功效。ok函数会在后续进行介绍；connect与信号槽有关，会在后续介绍。

第三个按键：self.ui.pushButton\_2.clicked.connect(self.cancel) 同样与信息发送有关系，此按键为取消键，即可取消自己的截图。cancel函数会在后续介绍。

4.3.2 ok函数

Ok函数将之前储存的api key，api id，secret key上传到百度网站，与网站建立连接。然后将窗口关闭。

4.3.3 cancel函数

直接关闭窗口。

4.4 主界面与查找搜索操作

由于本部分为主要相关代码，涉及信号-槽的核心部分，且内容相对较多，所以会将代码分段讲解。

4.4.1 init（初始化）

class MainWin(QWidget):

searchSignal = Signal(str, str)

def \_\_init\_\_(self):

super(MainWin, self).\_\_init\_\_()

self.ui = Ui\_MainForm()

self.ui.setupUi(self)

self.setFixedSize(self.width(), self.height())

self.scrot = None

self.settings = None

self.session = requests.Session()

self.session.headers = {

'Content-type': 'application/x-www-form-urlencoded'

}

self.app\_id = '16512590'

self.app\_key = '9cBepWNaqqXxLnUL25IMG72D'

self.secret\_key = 'EpDomzk5i95omRQW9f6qedGFWeXXQYb3'

self.client = AipOcr(self.app\_id, self.app\_key, self.secret\_key)

在初始化的部分中首先将信号函数 searchSignal = Signal(str, str) 进行实例化，这一步会在后续提及。

之后的部分 self.ui = Ui\_MainForm()以及self.ui.setupUi(self) 将Ui.MainForm实例化，并调用setup函数。

将界面的长宽设置为初始长度与宽度，在界面呢现将scort（屏幕截图，在之前有声明）设置为none。requests库的session对象还能为我们提供请求方法的缺省数据，通过设置session对象的属性来实现，并设置session的headers属性，通过update方法，将其余请求方法中的headers属性合并起来作为最终的请求方法的headers。并同时设置 api.key , api.id, secret key，将网站委托交给此api进行ocr识别。

4.4.2 界面按钮设置

# bind

self.ui.scrotButton.clicked.connect(self.onScrotButtonClicked)

self.ui.settingButton.clicked.connect(self.onSettingsButtonClicked)

self.ui.searchButton.clicked.connect(self.onSearchButtonClicked)

self.searchSignal.connect(self.search)

首先是三个按键，分别为截屏、搜索、设置。最后一行代码是将之前定义的searchSingnal与search过程相联系。

@Slot(QPixmap)

def prnt(self, screenShot: QPixmap):

buffer = QBuffer()

buffer.open(QBuffer.ReadWrite)

screenShot.save(buffer, 'png')

data = buffer.data().data()

res = self.client.basicGeneral(data)

if 'words\_result\_num' in res.keys():

print('[ INFO ] 识别成功')

text = ''

result = res['words\_result']

d = {

'单选题': self.ui.rad\_1,

'多选题': self.ui.rad\_2,

'填空题': self.ui.rad\_3,

'问答题': self.ui.rad\_4,

'判断题': self.ui.rad\_14,

'分析题': self.ui.rad\_5,

'解答题': self.ui.rad\_5,

'计算题': self.ui.rad\_5,

'证明题': self.ui.rad\_5,

}

这段代码首先将截图保存，并将结果保存在res中，并判断识别文字与题型是否成功，通过设置将题型设置为：

'单选题'、'多选题'、'填空题'、'问答题'、 '判断题'、 '分析题/解答题/计算题/证明题'，这几个类别也为智慧树中的所有题型种类，如果识别不成功也可以手动修正。

for words in result:

ques\_type = False

for k in d.keys():

if k in words['words']:

ques\_type = True

print('[ INFO ] 题目类型:', k)

d[k].setChecked(True)

d[k].repaint()

break

if not ques\_type:

text += words['words']

text = text.replace('(', '（').replace(')', '）').replace('?', '？').replace(',', '，')

print('[ INFO ] 题目:', text)

self.ui.questLineEdit.setText(text)

self.ui.questLineEdit.repaint()

self.ui.searchButton.click()

else:

print('[ INFO ] 识别失败')

@Slot(tuple)

def setApiKey(self, newPair: tuple):

self.app\_id, self.app\_key, self.secret\_key = newPair

def onSettingsButtonClicked(self):

self.settings = SettingsWin(self.app\_id, self.app\_key, self.secret\_key)

self.settings.apiConfigured.connect(self.setApiKey)

self.settings.show()

def onScrotButtonClicked(self):

self.scrot = Scrot()

self.scrot.scrotFinished.connect(self.prnt)

self.scrot.run()

def onSearchButtonClicked(self):

self.ui.searchButton.setDisabled(True)

self.ui.searchButton.repaint()

type\_btn = self.ui.typeBtnGroup.checkedButton()

text = self.ui.questLineEdit.text()

if len(text) <= 1:

self.ui.searchButton.setDisabled(False)

self.ui.searchButton.repaint()

return

这里引入了一个槽函数，首先设置了selfApiKey，然后定义了新函数onSettingsButtonClicked(self)，onScrotButtonClicked(self)，onSearchButtonClicked(self) 这几个函数为之前的connect函数的触发条件：

之前的函数：

self.ui.scrotButton.clicked.connect(self.onScrotButtonClicked)

self.ui.settingButton.clicked.connect(self.onSettingsButtonClicked)

self.ui.searchButton.clicked.connect(self.onSearchButtonClicked)

其中第一个函数onSettingsButtonClicked(self)将窗口的self. app id, self. app key, self. secret\_key进行了设置，同时将selfApiKey设置为了连接的参量，并将结果显示。

第二个函数：onScrotButtonClicked(self)

这个函数将之前定义的socrt类实例化，调用run（）函数，是截图的范围充满屏幕，以满足可以在屏幕的任何位置进行截图。

第三个函数：onSearchButtonClicked(self):

这个函数将之前设置的搜索按钮的状态设置为可点击状态。在点击后检测接受到的文本，如果没有文本返回则将按键的状态设置为不可点击。type\_btn这个变量作为检测按键是否处于可使用状态，集可以反映由网页返回的文本是否为空，如果不为空集可以通过未接调用执行之后的代码：

typ = '' if type\_btn is None else type\_btn.objectName()[4:]

self.searchSignal.emit(typ, text)

这两行代码可以向网页发送之前api返回的识别文本，进行进一步的搜索。

@Slot(str, str)

def search(self, typ: str, text: str):

print('[ INFO ] 正在查找')

self.ui.textBrowser.setText('正在查找...')

self.ui.textBrowser.repaint()

if typ == '':

res = self.session.post('http://mooc.forestpolice.org/zhs/0/' + text).json()

else:

res = self.session.post('http://mooc.forestpolice.org/zhs/0/' + text, data={'type': typ}).json()

print(res['code'])

if res['code'] == -1:

print('[FAILED] 查找失败')

self.ui.textBrowser.setText('未找到答案\n可以尝试从后往前删除一些内容来重新搜索\n以便提高搜索成功率')

self.ui.searchButton.setDisabled(False)

else:

print('[ OK ] 查找成功')

print('[ INFO ]', res['data'])

self.ui.textBrowser.setText(res['data'].replace('#', '\n'))

self.ui.searchButton.setDisabled(False)

在这里同样是一个槽函数，对应之前的 self.searchSignal.emit(typ, text)

首先是search过程，分为两种情况来搜索题库的网址，其中第一种是在已知题目类型（即typ变量时），直接上到智慧树题目库的该类别的题目的网址中进行检索，另一种为typ变量未检测出需要手动添加时，则将题目类别对赢得网址后缀添加到题库的网址并前往该网址中进行查找。

查找使用的为python内最简单的bmp算法，并将查找的结果存贮在字典变量code中并返回，如果code==-1即意味着查找失败，这在后台界面输出查找失败，并在主窗口界面中输出以下内容:

“未找到答案

可以尝试从后往前删除一些内容来重新搜索

以便提高搜索成功率”

由于查找本身使用的是bmp算法，再加上百度ocr的识别本身可能存在一定的误差，所以删除一些文本在进行搜查可能提高搜查成功的概率

如果查找成功，则在后台输出查找成功，并将结果同时输出。并将结果返回在主窗口上。

4.5 主函数

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

mainwin = MainWin()

mainwin.show()

sys.exit(app.exec\_())

将之前定义的MainWin类实例化，然后调用show函数既可以运行程序。

在运行结束后退出程序即可

5．信号-槽的使用

信号与槽是一种Qt中非常常用的高级接口，主要应用于对象之间的通信，如本代码中的与百度网页进行api连接，这是Qt的一个核心特性之一。Qt是信号与槽中非常重要的机制，也是Qt所特有的机制。

信号是指当某个信号对其客户或所有者发生的内部状态发生改动，信号被一个对象发射。只有定义过这个信号的类及其派生类能够发射这个信号。当一个信号被发射时，和其相关联的槽将被即时执行，就如同一个正常的函数调用相同。如果存在多个槽和某个信号相关联，那么，当这个信号被发射时，这些槽将会一个接一个地执行，不过执行的顺序是随机的、不确定的。信号的声明是在头文件中进行的，QT中使用signals关键字。

槽是指槽是普通的C++成员函数，能被正常调用，唯一的特别之处是非常多信号能和其相关联。当和其关联的信号被发射时，这个槽就会被调用。槽能有参数，但槽的参数不能有缺省值。槽也能够声明为虚函数。槽既可以在头文件中声明，也可以在程序中用关键字slots修饰。本程序中也主要使用的此方式。

在gui的用户界面中，当用户操作一个窗口部件是，需要其他窗口的响应或者可以激活其他操作。在没有槽函数时通常用回调函数来实现，而对于Qt库则可以通过槽函数来解决复杂的指针函数，是的程序编写变得简单。

所有的QObject都能够包含信号与槽，当其对象发生改变时，信号（singnal）就可以有该对象发射（emit）出，这个过程就是信息的封装。

信号与槽的关联主要为手动关联，即为connect函数：通过调用QObject对象的connect函数将某个对象的信号与另外一个对象的槽函数相关联，这样当发射者发射信号时，接受者的槽函数将被调用。

另外一种方法为自动关联，即将关联函数整合到槽名中，自动关联的槽名有字符，发射信号的部件和信号的名称组成，本次程序中主要是用的是第二种方法。在实际的连接中**一个信号可以和多个槽相连（槽会一个接一个地被调用，但是调用的顺序是不确定的）；多个信号可以连接一个槽（只要任意一个信号产生，这个槽就回被调用）；**

值得一提的是，使用QT开发时，通常使用connect函数只传递四个参数：connect(Sender,SIGNAL(signal),Receiver,SLOT(slot))；

但是在实际上是有第五个参数的，只是通常该函数已经给第五个参数赋值了而已，我们所使用的是默认值。

实际上connect函数应该是如下形式：

connect(sender,SIGNAL(signal),receiver,SLOT(slot)，Qt::DirectConnection);

6．程序使用演示

首先打开main.py程序，用cmd运行，打开主窗口显示如下：

在打开窗口之后，打开智慧树网址，进入课程，选择答题，章节测试或者期末考试题目都可以。之后点击选取方框，屏幕透明度会降低，只后通过鼠标框选自己所要查找答案的题目（只需要题目即可，不需要框选答案，框选答案有可能导致查询失败）

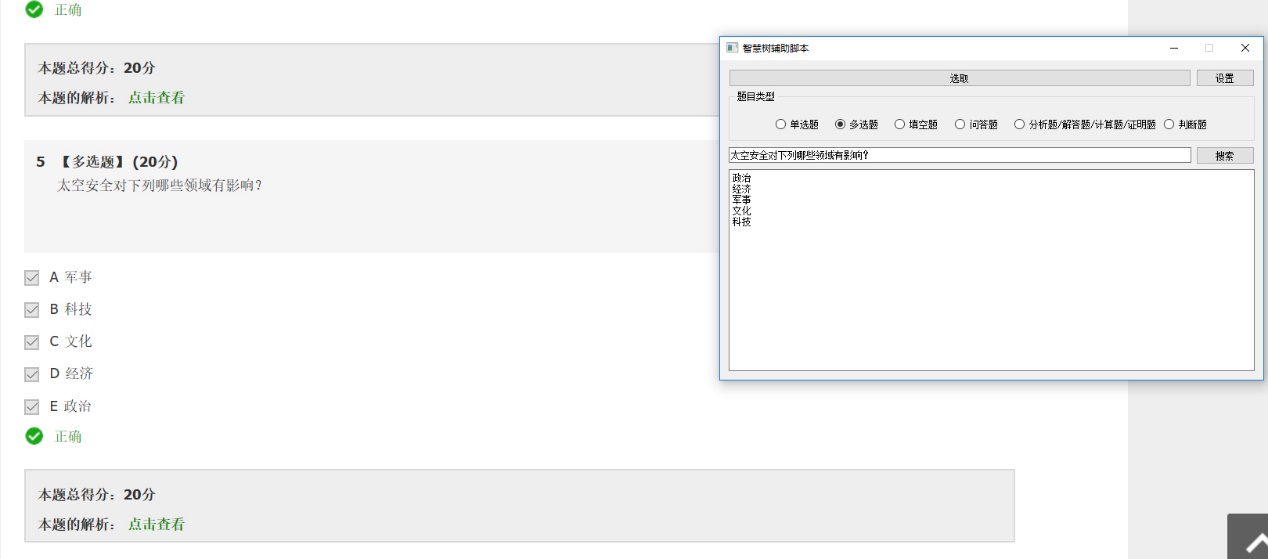
框选完成后请等待程序的运行，程序会自动识别所选择的题目类型，如果识别有错误，可以手动更改，之后会在下面的方框中返回出答案的查询结果。

如果查询失败，则可以尝试删一下文本后继续重新搜索。





除单选题外，本程序也可以查询别的题目，如果查询结果为多选题，会将答案依次排列，在方框内。



7．程序移植性与收获

本程序主要分为两个部分，截屏并识别图像的文字内容与将文字上传并查询的过程，两个主要的过程都有移植的能力。

文字识别的部分可以作为很多文字扫描的基础，可以将扫描后获得的文字加以输出，这样效率更高，远高于手打或者普通扫描的速度，并且可以作为很多程序的第一段铺垫。缺点在于需要利用百度的ocr，所以对网络有要求，只能在联网的时候使用。

上传查询的部分可以用于一些智能查询的方面，可以将所查询的文本在百度或者其他搜索引擎上以实现搜索功能。可能会在之后用于实现智能搜索并返回综合后德都的结果的功能，以减少搜索中信息的分析过程，节约搜索时间，提高搜索效率。

通过本次编程的实践，我理解了所谓的编程的思路。之前课本中所讲的知识只告诉们原理而没有告诉我们具体的使用情景，而向python这样封装较好的语言，有太多细枝末节的知识点，单靠记忆本身很难去熟练掌握。所以这个时候当真正需要去解决问题时就需要我们从问题出发去了解所需要的代码，这样才能去解决一些实际的问题。通过过程中的学习与问题的查询与积累才可以熟练掌握这类代码。

通过本次的实践我学习了python中Qt库的信号-槽向网络发中请求的原理，并理解了在gui编程中操作，触发之间的关系。Python相对之前学习的C++来说是一门更加简洁的语言，这一点在编程中也深有体会，这样的简化在实际的使用中也方便了编程人员。这次实践只进行了图形界面的编程，对于剩下的部分并未有更深的探索，希望在之后的学习中可以有所了解。