

****

Python课程设计

|  |  |
| --- | --- |
| 题目 | 员工刷脸考勤 |
| 学生姓名 | 肖涛 |
| 学生学号 | 0905160212 |
| 指导教师 | 支国明、蒋富 |
| 学 院 | 信息科学与工程学院 |
| 专业班级 | 通信工程1602班 |

注：本word文档设计了导航标题，为了方便阅读，可在 **视图**

上勾选 **导航窗格**

## 需求分析

选题“员工刷脸考勤”，要求采用python语言开发，可以通过摄像头添加员工面部信息，这里就涉及到两个具体的个问题，一个是应该以什么样的数据来标识每一个员工的面部信息，二是持久化地保存这些信息到数据库中去。更细地，还涉及表的设计；另一个基本要求是通过摄像头识别员工面部信息来完成考勤，这个问题基本可以通过遍历数据库里的员工面部数据与当前摄像头里的员工面部数据的比对来实现，但有一个问题就是假如摄像头里有多张人脸改怎么处理。扩展要求是导出每日的考勤表，可以拆分为两个部分，一个是存储考勤信息，一个是展示考勤信息。

我们希望达到的目标是：

（1）仿照通用型软件界面设计的原则，所有的操作都在菜单栏里实现，一部分区域用于展示摄像头实时读取并由程序加工后的视频流信息，另一部分区域做控制台输出，打印相关信息，比如提示员工面部信息添加成功、添加失败及其原因，提示员工打卡成功、打卡失败及其原因；添加面部信息时人是必须和程序进行交互的，比如输入一些相关的信息，这个时候程序是阻塞的；但是在打卡的时候，程序是不阻塞的，如果不点击关闭打卡，它会一直在打卡的模式，等待并识别每一个前来打卡的员工，这比较符合现实的使用场景。

（2）建表来存储员工信息和考勤信息，每次新建录入员工面部信息时，要求输入工号、姓名，并查无重后方可录入，录入时只取距离屏幕最近的员工的面部信息，这是考虑到实际打卡都是依次进行而不是一群人一群人地打卡，录入时有两种模式可供选择，自动模式：一旦识别到人脸就自动捕获截图，连续截图达到10张就结束录入；手动模式：点击菜单结束录入，不一定要得到10张。结束录入后就开始从刚才捕获的员工面部提取人脸特征数据并连同之前输入的员工姓名等数据作为一行记录保存到数据库中，如果没有捕获到员工面部或者刚才捕获的面部信息不是同一个人，这行记录就被丢弃。

考勤时，须满足三个条件：面部信息已录入、在打卡时间段内、未重复打卡，只有打卡成功，打卡人姓名及工号、打卡日期及时间才会被当成一行记录保存到数据库并在控制台输出打卡成功信息，否则会在控制台输出失败及其原因信息。

总而言之：我们的设计目标是规范化、人性化。

## 总体设计

为了完成上述目标一，程序的界面初始化分为三部分，第一部分初始化菜单栏，第二部分初始化左边控制台，第三部分初始化右边展示面板，使这三部分相互独立；数据逻辑部分的初始化分为两部分，第一部分是数据库部分的初始化，如果数据库/表不存在就新建，存在则加载相关数据，第二部分是初始化一些需要循环使用的变量，比如新建录入时的员工姓名、工号、截图数目计数器等,每当完成录入时这些数据都应该被重置成初始化以待下一次录入，把这些初始化语句写成一个函数可以提高代码复用度。

上述目标二主要是一些限制性条件，可以通过添加判断语句来实现，比如对输入id的合法性检验：

while self.id == ID\_WORKER\_UNAVIABLE:

self.id = wx.GetNumberFromUser(message="请输入您的工号(-1不可用)",prompt="工号", caption="温馨提示", value=ID\_WORKER\_UNAVIABLE, parent=self.bmp,max=100000000,min=ID\_WORKER\_UNAVIABLE)

for knew\_id in self.knew\_id:

if knew\_id == self.id:

self.id = ID\_WORKER\_UNAVIABLE

wx.MessageBox(message="工号已存在，请重新输入", caption="警告")

其中ID\_WORKER\_UNAVIABLE是id的初始化值-1，不可用，self.knew.id是从数据库里加载出来的id列表，如果id非法（已重复或者不在0~100000000，就会一直有新的弹窗来提示输入id。

再比如对拒绝多张人脸时、只处理距离屏幕最近的员工的面部信息：

if len(dets) != 0:

biggest\_face = dets[0]

#取占比最大的脸

maxArea = 0

for det in dets:

w = det.right() - det.left()

h = det.top()-det.bottom()

if w\*h > maxArea:

biggest\_face = det

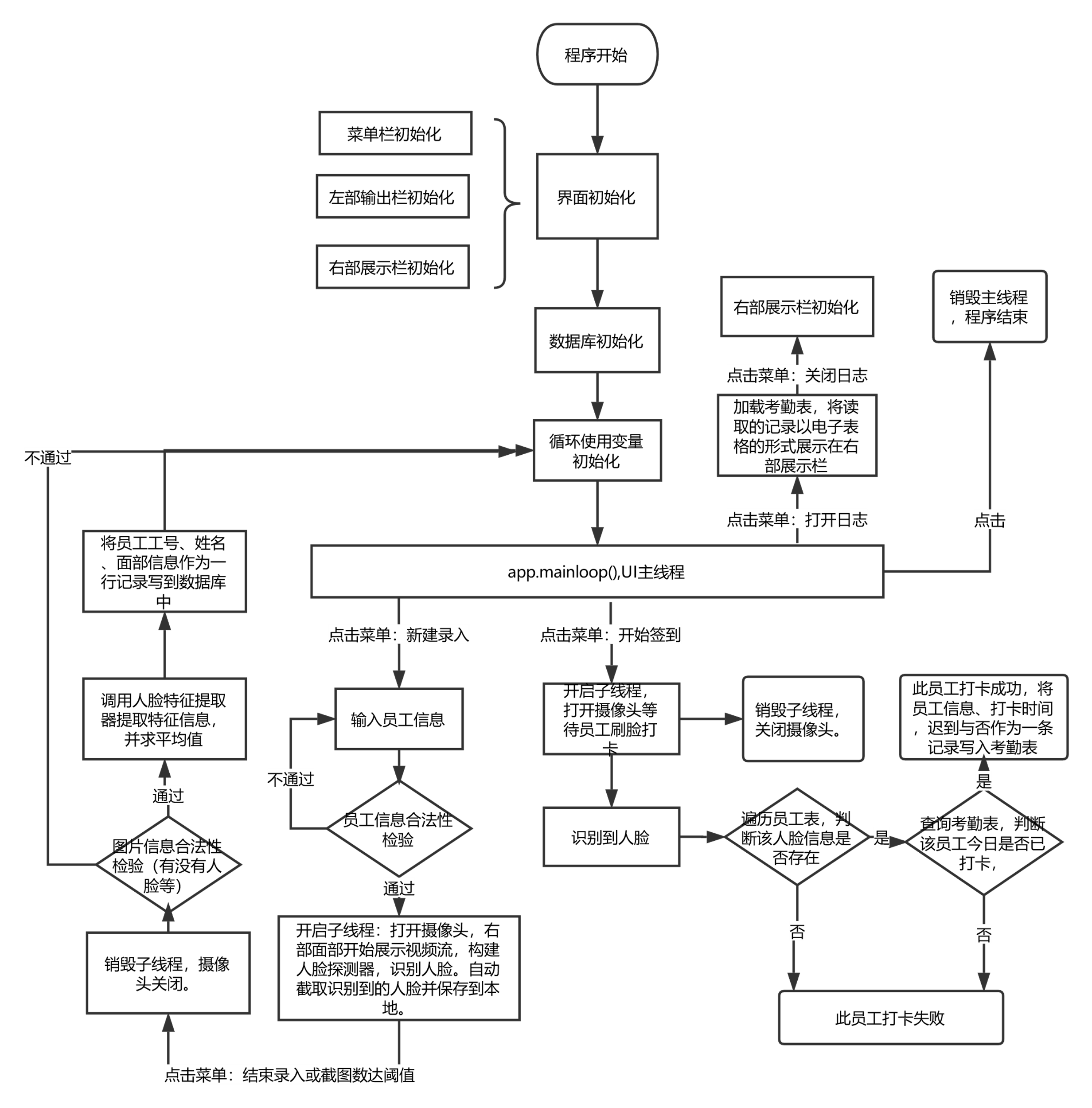
maxArea = w\*h

dets是侦测到的所有面部数组，biggest\_face是距离屏幕最近的面部。

本次课设的完整过程如下：选了员工刷脸考勤系统以后，我和小组成员刘佳璇查阅了大量资料，了解到dlib这个库，于是我们利用dlib库自带的人脸预测器、特征提取器，然后计算128d特征来完成人脸识别，再加上我们之前构想的逻辑，大致完成了课设，由于完成得早，我们大致在课设安排时间的一半时就给老师检查，虽然基本要求和拓展要求都已经实现，但是还有许多可以完善的地方，老师给出了许多宝贵的意见：

1. 把数据保存的地方由csv文件改成数据库。
2. 减少弹出式界面，用frame替换的方式解决之。
3. 刷脸打卡的界面不需要打卡者交互，全自动打卡。

最后检查的时候这些问题都被解决了。

程序框图：

注：图片如看不清也可在线预览：<https://www.processon.com/view/link/5bbcc953e4b08faf8c7324a1>

本程序的设计思想大致可分为以下几个方面

1. 面向对象的原则，整个程序的主体就是一个WAS(WorkAttendanceSystem)类,所有的实现都围绕这个类展开。
2. 界面和数据逻辑分离的原则，WAS类的初始化过程包括界面的初始化和数据初始化，两者相互独立。
3. 代码封装原则，多次调用的语句集写成接口供调用，没有冗余的代码。
4. 接口隔离原则:使用多个专门的接口,而不是使用单一的总接口。

## 详细设计

函数清单

注：所有类内的函数的第一个参数为self，表明该函数属于该类，后面不再赘述

1. def \_\_init\_\_(self)

WAS类的构造函数，主要是完成一些初始化操作，如初始化菜单、信息打印面板、主展示面板以及初始化加载数据库、初始化循环使用的变量。

1. def initMenu(self):

完成菜单的初始化显示，点击事件绑定。

1. def initInfoText(self):

完成左边信息提示面板的初始化显示。

1. def initGallery(self):

完成右边主展示面板的初始化显示。

1. def initDatabase(self):

数据库的初始化，建立数据库连接（如果数据库inspurer.db不存在则先新建），如果数据库中不存在员工信息worker\_info和考勤logcat这两个表，则依次创建。

1. def loadDataBase(self,type):

该模块函数完成从数据库读取数据的操作，包括读取员工信息和考勤信息，第二个参数type用于标识是加载员工信息还是考勤信息，一方面，可以统一接口，打开数据库和得到游标、关闭连接是一样的，将两个读取接口合二为一，提高代码复用度；另一方面，可以减少加载的工作量，减少IO,提高程序运行速度；最后，因为读取信息前对上一次读取的信息列表做了清空处理，用type标识可以避免读取一个表时对另一个表造成的误操作。

1. def insertARow(self,Row,type):

该模块函数完成写数据库操作，第二个参数为准备写的一条记录，第三个参数type表示要对哪一个表进行写操作。

1. def adapt\_array(self,arr):

将提取的人脸特征信息（列表）压缩，入口参数就是待压缩的数据，出口参数是压缩后的数据，用于写入数据库。

1. def convert\_array(self,text):

将读取出来的数据解压缩成人脸特征信息，入口参数是待解压得数据，出口参数是解压后的数据。

1. def return\_euclidean\_distance(feature\_1, feature\_2):

计算两个人脸的欧式距离，入口参数是两个人脸的特征数据，出口参数是判定的结果，欧式距离大于0.4判为不同，不大于判为相同。

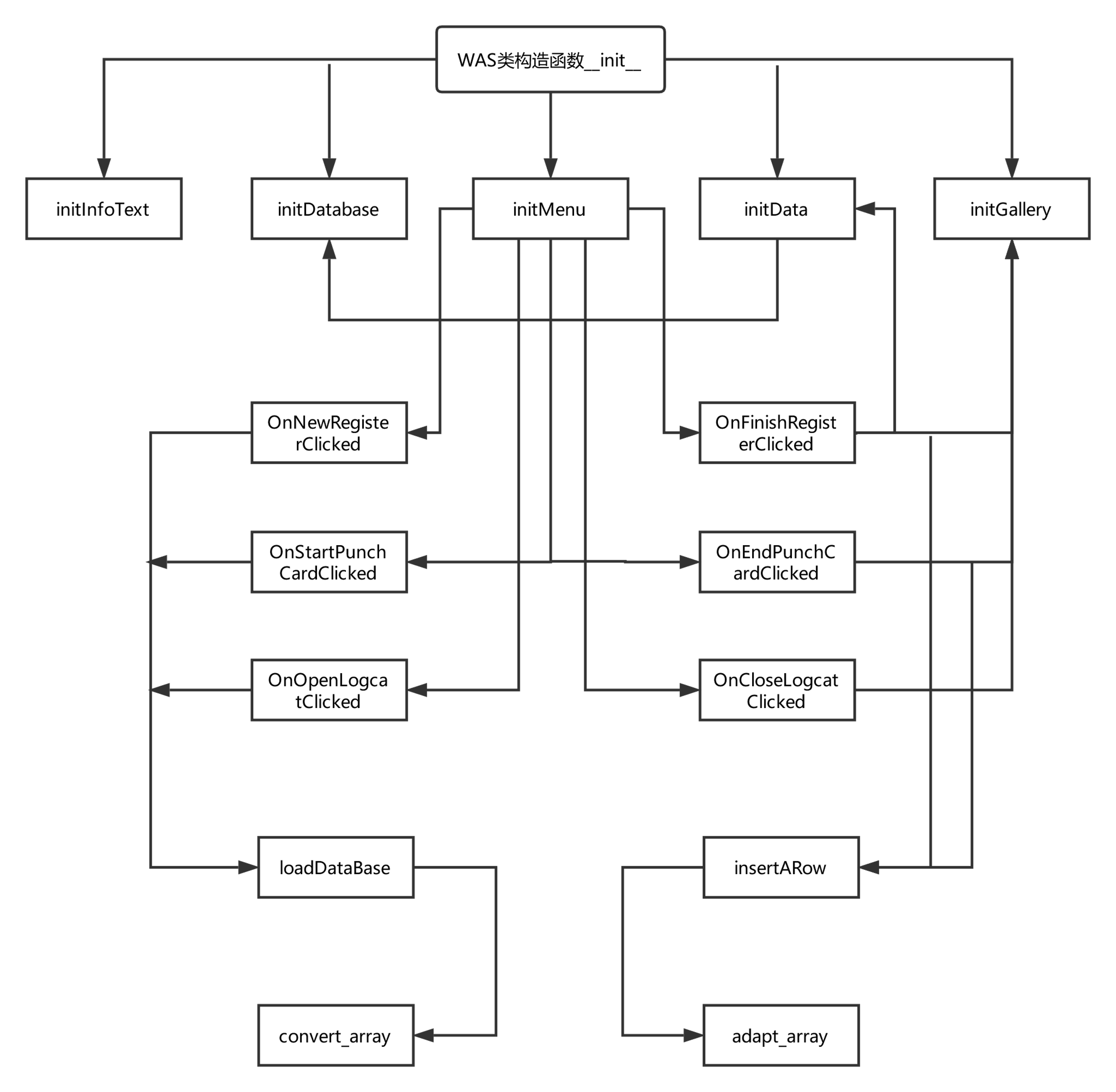
1. def OnNewRegisterClicked(self,event):

见名知义，菜单新建录入的监听事件，参数event为事件信息，其他几个菜单的(OnFinishRegisterClicked,OnStartPunchCardClicked, OnEndPunchCardClicked, OnOpenLogcatClicked,OnCloseLogcatClicked)类似，在此不再赘述。

1. def getDateAndTime(self):

得到当前日期和时间，并组装成特定格式作为出口参数返回。

函数调用关系:箭头指向被调用者

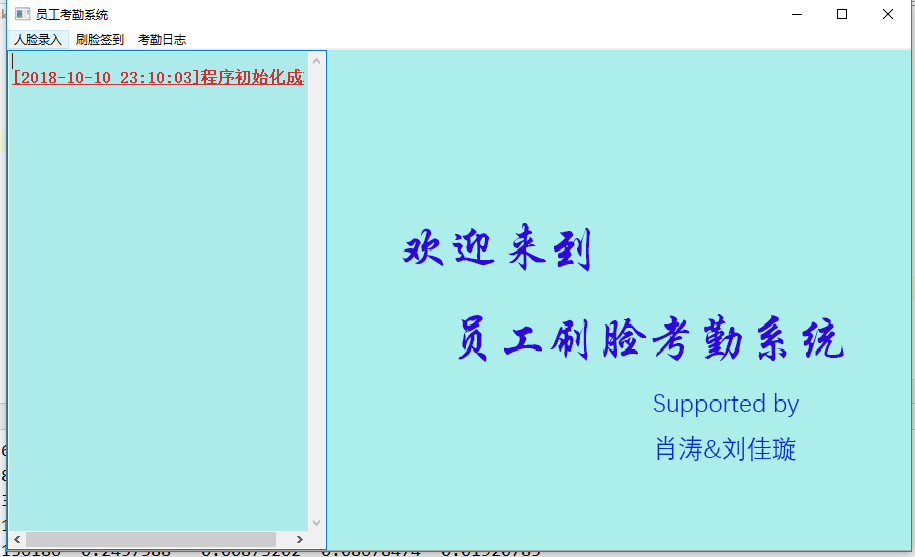


在线预览地址：

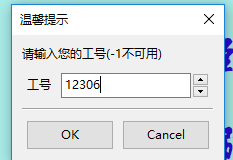
<https://www.processon.com/view/link/5bbe0b0de4b0534c9bfbecb4>

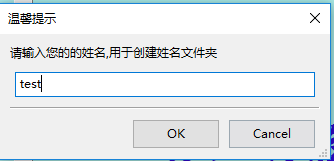
## 程序运行结果测试与分析

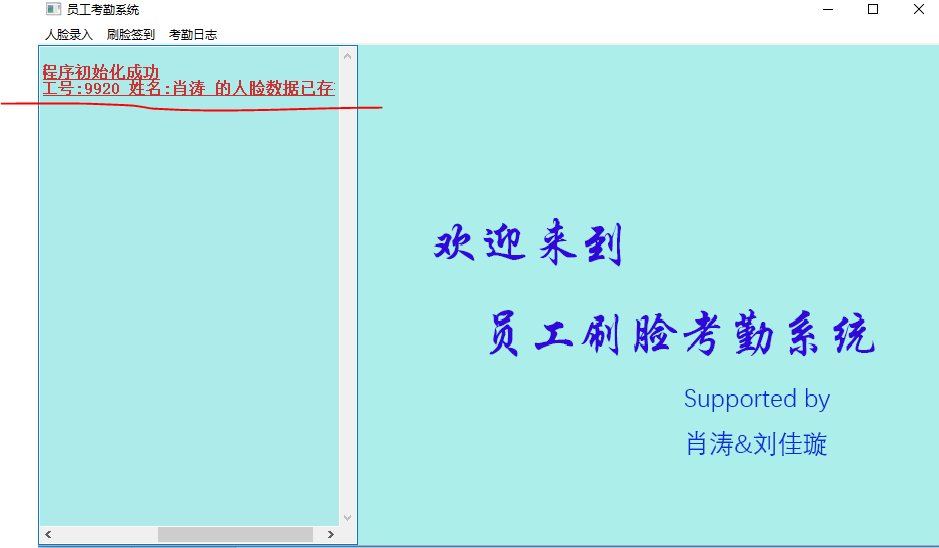
1. 程序主界面



1. 新建录入







我们看到，信息栏有人脸数据重复警告，本次录入取消。

于是我们把数据库数据删了重来。

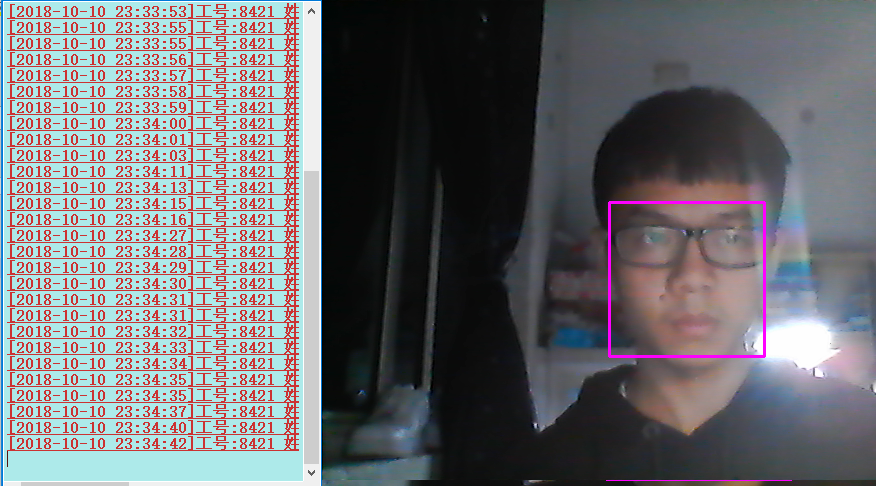


下面是打印的日志信息(为保证格式，复制到记事本中截的图)



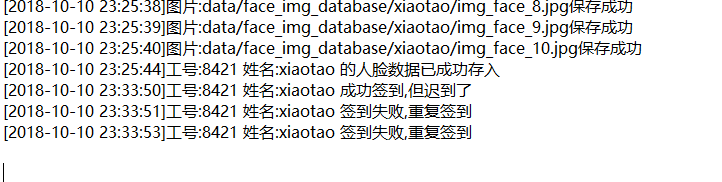
看到已经录入成功了。

1. 开始打卡



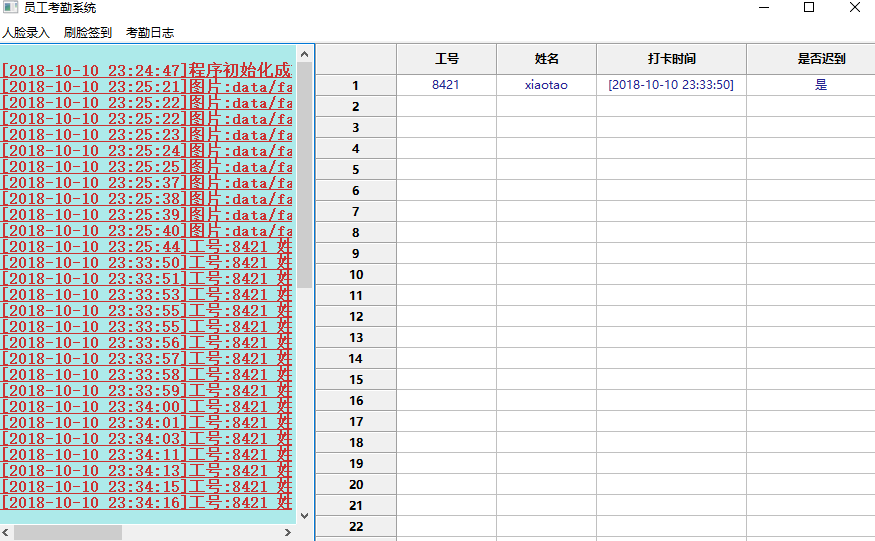


提示信息打印如下



迟到与否的临界时间是9:00

1. 展示日志





只有已经录入且第一次成功签到才会写进到数据库，无论迟到与否。

## 实验心得

遇到的问题及解决办法:

1. opencv自带的视频窗口不支持太多的UI扩展,怎么把视频流嵌到自定义的界面上去是一个棘手的问题，于是我想到了实现实时截图，然后把图片显示到面板上去，这里有一个延时的问题，opencv只允许整秒整秒地设置延时时间,设置成0s,程序会卡死，只好设置成1s，于是我们看到的‘视频流’会有少许的延时和卡顿,后面看到opencv还有一个设置刷新率fps的接口，但是尝试了也没用，估计和延时有冲突，最最后面只好开了一个子线程来做这些处理，防止主线程阻塞。
2. 识别人脸的时候如果有多张人脸入境，最开始我是只取所有识别到的人脸的第一张人脸，但是无法保证在许多时刻点都是同一个人脸,因此做了改进，取距离屏幕最近的人脸，具体做法是求最大的人脸面积，一般来说距离屏幕越近面积越大。
3. 其他的大都是编写程序过程中遇到的问题，比如什么中文设置问题，变量没有按照预期演化，这个时候，一般百度或者google，在csdn或者stackoverflow能找到解决办法，或者在pycharm开启debug模式，追踪变量等，在此不在赘述。

本人承担的任务：

人脸识别、UI设计及编写代码、日志展示。

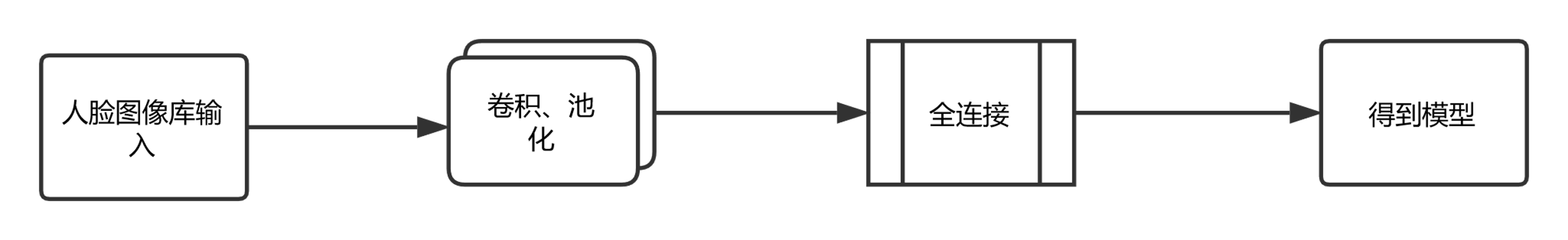
因时间关系没有完成的问题

总而言之，此员工考勤系统最核心的就是人脸识别，在此我直接用的是python 第三方库dlib自带的人脸预测器、特征提取器，对于同一人多份特征的处理也仅仅是取平均脸而已，平均脸的缺点有

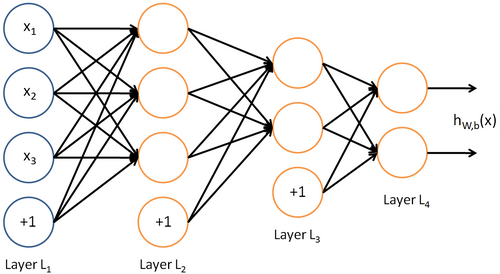
1. 受环境影响大，如果在光线弱的环境录入人脸，在光线好的地方可能就会出现识别错误，但是这个问题可以用取光线弱和光线好的环境下的人脸取平均来解决，显然比较麻烦。
2. 计算量比较大，每一条特征有128个数据，10张截图求平均就有1280次加法和128次除法。

而利用卷积神经网络来实现人脸识别可以有效地提高人脸识别准确度，后续我将学习用卷积神经网络和聚类来构建训练模型。

比如模型训练阶段



其中神经网络部分：



然后根据模型去识别人脸，最高识别准确率可达99%以上。

## 代码

#coding=utf-8

import wx

import wx.grid

import sqlite3

from time import localtime,strftime

import os

from skimage import io as iio

import io

import zlib

import dlib # 人脸识别的库dlib

import numpy as np # 数据处理的库numpy

import cv2 # 图像处理的库OpenCv

import \_thread

ID\_NEW\_REGISTER = 160

ID\_FINISH\_REGISTER = 161

ID\_START\_PUNCHCARD = 190

ID\_END\_PUNCARD = 191

ID\_OPEN\_LOGCAT = 283

ID\_CLOSE\_LOGCAT = 284

ID\_WORKER\_UNAVIABLE = -1

PATH\_FACE = "data/face\_img\_database/"

# face recognition model, the object maps human faces into 128D vectors

facerec = dlib.face\_recognition\_model\_v1("model/dlib\_face\_recognition\_resnet\_model\_v1.dat")

# Dlib 预测器

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor('model/shape\_predictor\_68\_face\_landmarks.dat')

def return\_euclidean\_distance(feature\_1, feature\_2):

feature\_1 = np.array(feature\_1)

feature\_2 = np.array(feature\_2)

dist = np.sqrt(np.sum(np.square(feature\_1 - feature\_2)))

print("欧式距离: ", dist)

if dist > 0.4:

return "diff"

else:

return "same"

class WAS(wx.Frame):

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

wx.Frame.\_\_init\_\_(self,parent=None,title="员工考勤系统",size=(920,560))

self.initMenu()

self.initInfoText()

self.initGallery()

self.initDatabase()

self.initData()

def initData(self):

self.name = ""

self.id =ID\_WORKER\_UNAVIABLE

self.face\_feature = ""

self.pic\_num = 0

self.flag\_registed = False

self.puncard\_time = "09:00:00"

self.loadDataBase(1)

def initMenu(self):

menuBar = wx.MenuBar() #生成菜单栏

menu\_Font = wx.Font()#Font(faceName="consolas",pointsize=20)

menu\_Font.SetPointSize(14)

menu\_Font.SetWeight(wx.BOLD)

registerMenu = wx.Menu() #生成菜单

self.new\_register = wx.MenuItem(registerMenu,ID\_NEW\_REGISTER,"新建录入")

self.new\_register.SetBitmap(wx.Bitmap("drawable/new\_register.png"))

self.new\_register.SetTextColour("SLATE BLUE")

self.new\_register.SetFont(menu\_Font)

registerMenu.Append(self.new\_register)

self.finish\_register = wx.MenuItem(registerMenu,ID\_FINISH\_REGISTER,"完成录入")

self.finish\_register.SetBitmap(wx.Bitmap("drawable/finish\_register.png"))

self.finish\_register.SetTextColour("SLATE BLUE")

self.finish\_register.SetFont(menu\_Font)

self.finish\_register.Enable(False)

registerMenu.Append(self.finish\_register)

puncardMenu = wx.Menu()

self.start\_punchcard = wx.MenuItem(puncardMenu,ID\_START\_PUNCHCARD,"开始签到")

self.start\_punchcard.SetBitmap(wx.Bitmap("drawable/start\_punchcard.png"))

self.start\_punchcard.SetTextColour("SLATE BLUE")

self.start\_punchcard.SetFont(menu\_Font)

puncardMenu.Append(self.start\_punchcard)

self.end\_puncard = wx.MenuItem(puncardMenu,ID\_END\_PUNCARD,"结束签到")

self.end\_puncard.SetBitmap(wx.Bitmap("drawable/end\_puncard.png"))

self.end\_puncard.SetTextColour("SLATE BLUE")

self.end\_puncard.SetFont(menu\_Font)

self.end\_puncard.Enable(False)

puncardMenu.Append(self.end\_puncard)

logcatMenu = wx.Menu()

self.open\_logcat = wx.MenuItem(logcatMenu,ID\_OPEN\_LOGCAT,"打开日志")

self.open\_logcat.SetBitmap(wx.Bitmap("drawable/open\_logcat.png"))

self.open\_logcat.SetFont(menu\_Font)

self.open\_logcat.SetTextColour("SLATE BLUE")

logcatMenu.Append(self.open\_logcat)

self.close\_logcat = wx.MenuItem(logcatMenu, ID\_CLOSE\_LOGCAT, "关闭日志")

self.close\_logcat.SetBitmap(wx.Bitmap("drawable/close\_logcat.png"))

self.close\_logcat.SetFont(menu\_Font)

self.close\_logcat.SetTextColour("SLATE BLUE")

logcatMenu.Append(self.close\_logcat)

menuBar.Append(registerMenu,"&人脸录入")

menuBar.Append(puncardMenu,"&刷脸签到")

menuBar.Append(logcatMenu,"&考勤日志")

self.SetMenuBar(menuBar)

self.Bind(wx.EVT\_MENU,self.OnNewRegisterClicked,id=ID\_NEW\_REGISTER)

self.Bind(wx.EVT\_MENU,self.OnFinishRegisterClicked,id=ID\_FINISH\_REGISTER)

self.Bind(wx.EVT\_MENU,self.OnStartPunchCardClicked,id=ID\_START\_PUNCHCARD)

self.Bind(wx.EVT\_MENU,self.OnEndPunchCardClicked,id=ID\_END\_PUNCARD)

self.Bind(wx.EVT\_MENU,self.OnOpenLogcatClicked,id=ID\_OPEN\_LOGCAT)

self.Bind(wx.EVT\_MENU,self.OnCloseLogcatClicked,id=ID\_CLOSE\_LOGCAT)

def OnOpenLogcatClicked(self,event):

self.loadDataBase(2)

grid = wx.grid.Grid(self,pos=(320,0),size=(600,500))

grid.CreateGrid(100, 4)

for i in range(100):

for j in range(4):

grid.SetCellAlignment(i,j,wx.ALIGN\_CENTER,wx.ALIGN\_CENTER)

grid.SetColLabelValue(0, "工号") #第一列标签

grid.SetColLabelValue(1, "姓名")

grid.SetColLabelValue(2, "打卡时间")

grid.SetColLabelValue(3, "是否迟到")

grid.SetColSize(0,100)

grid.SetColSize(1,100)

grid.SetColSize(2,150)

grid.SetColSize(3,150)

grid.SetCellTextColour("NAVY")

for i,id in enumerate(self.logcat\_id):

grid.SetCellValue(i,0,str(id))

grid.SetCellValue(i,1,self.logcat\_name[i])

grid.SetCellValue(i,2,self.logcat\_datetime[i])

grid.SetCellValue(i,3,self.logcat\_late[i])

pass

def OnCloseLogcatClicked(self,event):

self.initGallery()

pass

def register\_cap(self,event):

# 创建 cv2 摄像头对象

self.cap = cv2.VideoCapture(0)

# cap.set(propId, value)

# 设置视频参数，propId设置的视频参数，value设置的参数值

# self.cap.set(3, 600)

# self.cap.set(4,600)

# cap是否初始化成功

while self.cap.isOpened():

# cap.read()

# 返回两个值：

# 一个布尔值true/false，用来判断读取视频是否成功/是否到视频末尾

# 图像对象，图像的三维矩阵

flag, im\_rd = self.cap.read()

# 每帧数据延时1ms，延时为0读取的是静态帧

kk = cv2.waitKey(1)

# 人脸数 dets

dets = detector(im\_rd, 1)

# 检测到人脸

if len(dets) != 0:

biggest\_face = dets[0]

#取占比最大的脸

maxArea = 0

for det in dets:

w = det.right() - det.left()

h = det.top()-det.bottom()

if w\*h > maxArea:

biggest\_face = det

maxArea = w\*h

# 绘制矩形框

cv2.rectangle(im\_rd, tuple([biggest\_face.left(), biggest\_face.top()]),

tuple([biggest\_face.right(), biggest\_face.bottom()]),

(255, 0, 0), 2)

img\_height, img\_width = im\_rd.shape[:2]

image1 = cv2.cvtColor(im\_rd, cv2.COLOR\_BGR2RGB)

pic = wx.Bitmap.FromBuffer(img\_width, img\_height, image1)

# 显示图片在panel上

self.bmp.SetBitmap(pic)

# 获取当前捕获到的图像的所有人脸的特征，存储到 features\_cap\_arr

shape = predictor(im\_rd, biggest\_face)

features\_cap = facerec.compute\_face\_descriptor(im\_rd, shape)

# 对于某张人脸，遍历所有存储的人脸特征

for i,knew\_face\_feature in enumerate(self.knew\_face\_feature):

# 将某张人脸与存储的所有人脸数据进行比对

compare = return\_euclidean\_distance(features\_cap, knew\_face\_feature)

if compare == "same": # 找到了相似脸

self.infoText.AppendText(self.getDateAndTime()+"工号:"+str(self.knew\_id[i])

+" 姓名:"+self.knew\_name[i]+" 的人脸数据已存在\r\n")

self.flag\_registed = True

self.OnFinishRegister()

\_thread.exit()

# print(features\_known\_arr[i][-1])

face\_height = biggest\_face.bottom()-biggest\_face.top()

face\_width = biggest\_face.right()- biggest\_face.left()

im\_blank = np.zeros((face\_height, face\_width, 3), np.uint8)

try:

for ii in range(face\_height):

for jj in range(face\_width):

im\_blank[ii][jj] = im\_rd[biggest\_face.top() + ii][biggest\_face.left() + jj]

# cv2.imwrite(path\_make\_dir+self.name + "/img\_face\_" + str(self.sc\_number) + ".jpg", im\_blank)

# cap = cv2.VideoCapture("\*\*\*.mp4")

# cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 2)

# ret, frame = cap.read()

# cv2.imwrite("我//h.jpg", frame) # 该方法不成功

# 解决python3下使用cv2.imwrite存储带有中文路径图片

if len(self.name)>0:

cv2.imencode('.jpg', im\_blank)[1].tofile(

PATH\_FACE + self.name + "/img\_face\_" + str(self.pic\_num) + ".jpg") # 正确方法

self.pic\_num += 1

print("写入本地：", str(PATH\_FACE + self.name) + "/img\_face\_" + str(self.pic\_num) + ".jpg")

self.infoText.AppendText(self.getDateAndTime()+"图片:"+str(PATH\_FACE + self.name) + "/img\_face\_" + str(self.pic\_num) + ".jpg保存成功\r\n")

except:

print("保存照片异常,请对准摄像头")

if self.new\_register.IsEnabled():

\_thread.exit()

if self.pic\_num == 10:

self.OnFinishRegister()

\_thread.exit()

def OnNewRegisterClicked(self,event):

self.new\_register.Enable(False)

self.finish\_register.Enable(True)

self.loadDataBase(1)

while self.id == ID\_WORKER\_UNAVIABLE:

self.id = wx.GetNumberFromUser(message="请输入您的工号(-1不可用)",

prompt="工号", caption="温馨提示",

value=ID\_WORKER\_UNAVIABLE,

parent=self.bmp,max=100000000,min=ID\_WORKER\_UNAVIABLE)

for knew\_id in self.knew\_id:

if knew\_id == self.id:

self.id = ID\_WORKER\_UNAVIABLE

wx.MessageBox(message="工号已存在，请重新输入", caption="警告")

while self.name == '':

self.name = wx.GetTextFromUser(message="请输入您的的姓名,用于创建姓名文件夹",

caption="温馨提示",

default\_value="", parent=self.bmp)

# 监测是否重名

for exsit\_name in (os.listdir(PATH\_FACE)):

if self.name == exsit\_name:

wx.MessageBox(message="姓名文件夹已存在，请重新输入", caption="警告")

self.name = ''

break

os.makedirs(PATH\_FACE+self.name)

\_thread.start\_new\_thread(self.register\_cap,(event,))

pass

def OnFinishRegister(self):

self.new\_register.Enable(True)

self.finish\_register.Enable(False)

self.cap.release()

self.bmp.SetBitmap(wx.Bitmap(self.pic\_index))

if self.flag\_registed == True:

dir = PATH\_FACE + self.name

for file in os.listdir(dir):

os.remove(dir+"/"+file)

print("已删除已录入人脸的图片", dir+"/"+file)

os.rmdir(PATH\_FACE + self.name)

print("已删除已录入人脸的姓名文件夹", dir)

self.initData()

return

if self.pic\_num>0:

pics = os.listdir(PATH\_FACE + self.name)

feature\_list = []

feature\_average = []

for i in range(len(pics)):

pic\_path = PATH\_FACE + self.name + "/" + pics[i]

print("正在读的人脸图像：", pic\_path)

img = iio.imread(pic\_path)

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

dets = detector(img\_gray, 1)

if len(dets) != 0:

shape = predictor(img\_gray, dets[0])

face\_descriptor = facerec.compute\_face\_descriptor(img\_gray, shape)

feature\_list.append(face\_descriptor)

else:

face\_descriptor = 0

print("未在照片中识别到人脸")

if len(feature\_list) > 0:

for j in range(128):

#防止越界

feature\_average.append(0)

for i in range(len(feature\_list)):

feature\_average[j] += feature\_list[i][j]

feature\_average[j] = (feature\_average[j]) / len(feature\_list)

self.insertARow([self.id,self.name,feature\_average],1)

self.infoText.AppendText(self.getDateAndTime()+"工号:"+str(self.id)

+" 姓名:"+self.name+" 的人脸数据已成功存入\r\n")

pass

else:

os.rmdir(PATH\_FACE + self.name)

print("已删除空文件夹",PATH\_FACE + self.name)

self.initData()

def OnFinishRegisterClicked(self,event):

self.OnFinishRegister()

pass

def punchcard\_cap(self,event):

self.cap = cv2.VideoCapture(0)

# cap.set(propId, value)

# 设置视频参数，propId设置的视频参数，value设置的参数值

# self.cap.set(3, 600)

# self.cap.set(4,600)

# cap是否初始化成功

while self.cap.isOpened():

# cap.read()

# 返回两个值：

# 一个布尔值true/false，用来判断读取视频是否成功/是否到视频末尾

# 图像对象，图像的三维矩阵

flag, im\_rd = self.cap.read()

# 每帧数据延时1ms，延时为0读取的是静态帧

kk = cv2.waitKey(1)

# 人脸数 dets

dets = detector(im\_rd, 1)

# 检测到人脸

if len(dets) != 0:

biggest\_face = dets[0]

# 取占比最大的脸

maxArea = 0

for det in dets:

w = det.right() - det.left()

h = det.top() - det.bottom()

if w \* h > maxArea:

biggest\_face = det

maxArea = w \* h

# 绘制矩形框

cv2.rectangle(im\_rd, tuple([biggest\_face.left(), biggest\_face.top()]),

tuple([biggest\_face.right(), biggest\_face.bottom()]),

(255, 0, 255), 2)

img\_height, img\_width = im\_rd.shape[:2]

image1 = cv2.cvtColor(im\_rd, cv2.COLOR\_BGR2RGB)

pic = wx.Bitmap.FromBuffer(img\_width, img\_height, image1)

# 显示图片在panel上

self.bmp.SetBitmap(pic)

# 获取当前捕获到的图像的所有人脸的特征，存储到 features\_cap\_arr

shape = predictor(im\_rd, biggest\_face)

features\_cap = facerec.compute\_face\_descriptor(im\_rd, shape)

# 对于某张人脸，遍历所有存储的人脸特征

for i, knew\_face\_feature in enumerate(self.knew\_face\_feature):

# 将某张人脸与存储的所有人脸数据进行比对

compare = return\_euclidean\_distance(features\_cap, knew\_face\_feature)

if compare == "same": # 找到了相似脸

print("same")

flag = 0

nowdt = self.getDateAndTime()

for j,logcat\_name in enumerate(self.logcat\_name):

if logcat\_name == self.knew\_name[i] and nowdt[0:nowdt.index(" ")] == self.logcat\_datetime[j][0:self.logcat\_datetime[j].index(" ")]:

self.infoText.AppendText(nowdt+"工号:"+ str(self.knew\_id[i])

+ " 姓名:" + self.knew\_name[i] + " 签到失败,重复签到\r\n")

flag = 1

break

if flag == 1:

break

if nowdt[nowdt.index(" ")+1:-1] <= self.puncard\_time:

self.infoText.AppendText(nowdt + "工号:" + str(self.knew\_id[i])

+ " 姓名:" + self.knew\_name[i] + " 成功签到,且未迟到\r\n")

self.insertARow([self.knew\_id[i],self.knew\_name[i],nowdt,"否"],1)

else:

self.infoText.AppendText(nowdt + "工号:" + str(self.knew\_id[i])

+ " 姓名:" + self.knew\_name[i] + " 成功签到,但迟到了\r\n")

self.insertARow([self.knew\_id[i], self.knew\_name[i], nowdt, "是"], 2)

self.loadDataBase(2)

break

if self.start\_punchcard.IsEnabled():

self.bmp.SetBitmap(wx.Bitmap(self.pic\_index))

\_thread.exit()

def OnStartPunchCardClicked(self,event):

# cur\_hour = datetime.datetime.now().hour

# print(cur\_hour)

# if cur\_hour>=8 or cur\_hour<6:

# wx.MessageBox(message='''您错过了今天的签到时间，请明天再来\n

# 每天的签到时间是:6:00~7:59''', caption="警告")

# return

self.start\_punchcard.Enable(False)

self.end\_puncard.Enable(True)

self.loadDataBase(2)

\_thread.start\_new\_thread(self.punchcard\_cap,(event,))

pass

def OnEndPunchCardClicked(self,event):

self.start\_punchcard.Enable(True)

self.end\_puncard.Enable(False)

pass

def initInfoText(self):

#少了这两句infoText背景颜色设置失败，莫名奇怪

resultText = wx.StaticText(parent=self, pos = (10,20),size=(90, 60))

resultText.SetBackgroundColour('red')

self.info = "\r\n"+self.getDateAndTime()+"程序初始化成功\r\n"

#第二个参数水平混动条

self.infoText = wx.TextCtrl(parent=self,size=(320,500),

style=(wx.TE\_MULTILINE|wx.HSCROLL|wx.TE\_READONLY))

#前景色，也就是字体颜色

self.infoText.SetForegroundColour("ORANGE")

self.infoText.SetLabel(self.info)

#API:https://www.cnblogs.com/wangjian8888/p/6028777.html

# 没有这样的重载函数造成"par is not a key word",只好Set

font = wx.Font()

font.SetPointSize(12)

font.SetWeight(wx.BOLD)

font.SetUnderlined(True)

self.infoText.SetFont(font)

self.infoText.SetBackgroundColour('TURQUOISE')

pass

def initGallery(self):

self.pic\_index = wx.Image("drawable/index.png", wx.BITMAP\_TYPE\_ANY).Scale(600, 500)

self.bmp = wx.StaticBitmap(parent=self, pos=(320,0), bitmap=wx.Bitmap(self.pic\_index))

pass

def getDateAndTime(self):

dateandtime = strftime("%Y-%m-%d %H:%M:%S",localtime())

return "["+dateandtime+"]"

#数据库部分

#初始化数据库

def initDatabase(self):

conn = sqlite3.connect("inspurer.db") #建立数据库连接

cur = conn.cursor() #得到游标对象

cur.execute('''create table if not exists worker\_info

(name text not null,

id int not null primary key,

face\_feature array not null)''')

cur.execute('''create table if not exists logcat

(datetime text not null,

id int not null,

name text not null,

late text not null)''')

cur.close()

conn.commit()

conn.close()

def adapt\_array(self,arr):

out = io.BytesIO()

np.save(out, arr)

out.seek(0)

dataa = out.read()

# 压缩数据流

return sqlite3.Binary(zlib.compress(dataa, zlib.Z\_BEST\_COMPRESSION))

def convert\_array(self,text):

out = io.BytesIO(text)

out.seek(0)

dataa = out.read()

# 解压缩数据流

out = io.BytesIO(zlib.decompress(dataa))

return np.load(out)

def insertARow(self,Row,type):

conn = sqlite3.connect("inspurer.db") # 建立数据库连接

cur = conn.cursor() # 得到游标对象

if type == 1:

cur.execute("insert into worker\_info (id,name,face\_feature) values(?,?,?)",

(Row[0],Row[1],self.adapt\_array(Row[2])))

print("写人脸数据成功")

if type == 2:

cur.execute("insert into logcat (id,name,datetime,late) values(?,?,?,?)",

(Row[0],Row[1],Row[2],Row[3]))

print("写日志成功")

pass

cur.close()

conn.commit()

conn.close()

pass

def loadDataBase(self,type):

conn = sqlite3.connect("inspurer.db") # 建立数据库连接

cur = conn.cursor() # 得到游标对象

if type == 1:

self.knew\_id = []

self.knew\_name = []

self.knew\_face\_feature = []

cur.execute('select id,name,face\_feature from worker\_info')

origin = cur.fetchall()

for row in origin:

print(row[0])

self.knew\_id.append(row[0])

print(row[1])

self.knew\_name.append(row[1])

print(self.convert\_array(row[2]))

self.knew\_face\_feature.append(self.convert\_array(row[2]))

if type == 2:

self.logcat\_id = []

self.logcat\_name = []

self.logcat\_datetime = []

self.logcat\_late = []

cur.execute('select id,name,datetime,late from logcat')

origin = cur.fetchall()

for row in origin:

print(row[0])

self.logcat\_id.append(row[0])

print(row[1])

self.logcat\_name.append(row[1])

print(row[2])

self.logcat\_datetime.append(row[2])

print(row[3])

self.logcat\_late.append(row[3])

pass

app = wx.App()

frame = WAS()

frame.Show()

app.MainLoop()