人脸识别系统

课程：　　数字图像处理

指导老师：　　王亮

学号：　　S201961352

姓名：　　许　梦　文

目　录

前　言 1
第 1 章　人脸识别系统概述 1
第 2 章　人脸检测技术 1
§2.1 基于 Harr 级联的人脸检测 2
§2.2 基于 face_recognition 开源库的人脸检测 ..... **错误!未定义书签。**
第 3 章　人脸识别技术 5
§3.1 构造人脸数据集 5
§3.2 模型训练 7
§3.3 实时人脸识别 9
第 4 章　基于 PyQt5 的应用平台 10
总　结 12
附　录 14

前　言

近年来人类生物特征越来越广泛地用于身份识别，如 DNA 识别技术、指纹识别技术、虹膜识别技术、语音识别技术、人脸识别技术等。基于人类生物特征的识别技术具有安全可靠、特征唯一、不易伪造、不可窃取等优点。本文基于 OpenCV 图像处理库，利用《数字图像处理》课程所学的知识，实现简单地人脸识别技术。希望能够学以致用，通过实践来练习课堂上所学的知识。文章大致分为以下几个部分：人脸的检测、数据集准备、数据训练、人脸识别、讨论总结等。

综上所述，本文想要实现的功能就是：

- 实现人脸检测、对齐，构造数据集。
- 选择合适的人脸识别模型进行训练识别。
- 与实际应用相结合，实现一个与人脸识别相关的小应用。

1、人脸识别系统概述

人脸识别技术现在已经愈发成熟，有 OpenCV 的 Harr 分类器、Dlib 库等等，他们可以高精度实现人脸识别。本文希望运用已经成熟的技术搭建一个人脸识别系统的应用程序，通过摄像头采集图像，实现在线识别，并可以在线添加新的使用者，可以运用在电脑开机人脸检测等场景。本系统编程语言使用 Python3.7，在 PyCharm 平台上开发，主要依赖于 OpenCV 库函数，通过对采集到图像帧进行预处理、人脸检测、裁剪、最后再进行识别；再此基础上，再添加基于 PyQt 的应用程序框架，对系统进行完善，美化。

主要实现过程：

1) 使用摄像头采集图像显示，并检测出人脸
2) 实现在线生成数据集，对模型进行训练、识别
3) 将程序封装在应用程序中

2、人脸检测技术

早期人们的主要研究对象是人脸识别，即只根据人脸来识别身份，但是

随着应用环境越来越复杂，人脸识别越来越困难。往往需要先将人脸提取出来进行裁剪、旋转、对齐等，再进行人脸识别。本章主要采用 Harr 开源库、和 face_recognition 库分别进行人脸检测实验，并对比好坏。

2.1 基于 Harr 级联的人脸检测

Harr 的简单概念，当我们需要对某些东西进行分类时，很习惯的会想到观察它的细节与其他类别实物有什么不同。同理，从图像里面找出人脸也需要提取出图像的细节，这些被提取出来的细节成为特征，专业释义：从图像数据中提取特征。虽然像素是根据环境变化的，任意像素都可能影响多个特征，但是特征要比像素少的多。例如，边、顶点和细线都能生成具有判别性的特征。可以这么简单的理解，人脸上眼睛和鼻子的分布会造成图像上不同的阴影，但是大多数人的这些阴影像素分布又是相同的，Harr 就是通过提取这些特征来进行人脸检测。OpenCV 提供了 Harr 级联分类器和跟踪器，并将其保存成指定的文件格式。

获取 Harr 级联数据，在 OpenCV 源代码的副本中有一个 data/harrcascades 文件夹。该文件夹包含了所有 OpenCV 的人脸检测的 XML 文件，这些文件可以检测静止图像、视频和摄像头所得到的人脸图像，但是这些级联有个弊端就是必须用于正面、直立的人脸图像。

本系统使用 Python 语言基于 OpenCV3.0 函数库编写，首先进行摄像头采集图像，然后对图像做灰度化等相应处理，最后调用 detectMultiScale()函数对人脸进行检测。程序 detect_face_harr.py 主要程序如图 2-1 所示。

基于 Harr 级联的人脸检测主要函数有：cv2.VideoCapture(0)是获取摄像头的函数，笔记本内置摄像头参数为 0，外置的其他摄像头也可以选择其他参数；ret, frame = camera.read()函数是捕获图像并返回两个参数，ret 表示读取是否成功，frame 表示读取回来的图像；读取图像后将其转换为灰度图像，因为 OpenCV 中人脸检测需要灰度的色彩空间，然后就是调用我们的级联文件 CascadeClassifier(“haarcascade_frontalface_default.xml”)；接下来利用我们的 detectMultiScale()函数进行人脸检测，输入参数 scaleFactor 和 minNeighbor 分别表示人脸检测过程中每次迭代时图像的压缩率以及每个人脸矩形保留近

邻数目最小值，其中还有人脸最大、最小的设置，为了减小误差，我们基于采集整体图像为 640x480 将最小人脸设置为 minSize=(60,60)；最终利用 rectangle 函数在原图像上画出人脸。程序中显示了两张图片，一个是采集的全局图片，一个是分割出来的脸部图片如图 2-2 所示。

```python
def Detect_Face():
    # opencv自带的人脸识别分类器
    face_cascade = cv2.CascadeClassifier('.\opencv_module\haarcascade_frontalface_default.xml')
    # 调用笔记本内置摄像头，所以参数为0，如果有其他的摄像头可以调整参数为1，2
    camera = cv2.VideoCapture(0)
    while (True):
        ret, frame = camera.read()#获取图像
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # 灰度化
        faces = face_cascade.detectMultiScale(gray, 1.3, 5,minSize=(60,60))  # 人脸检测
        for (x, y, w, h) in faces:
            img = cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
            f = cv2.resize(gray[y:y + h, x:x + w], (200, 200))
            cv2.imshow("face", f)
        cv2.imshow("camera", frame)
        if cv2.waitKey(int(1000 / 12)) & 0xff == ord("q"):
            break
    camera.release()
    cv2.destroyAllWindows()
```

图 2-1. 基于 Harr 的人脸检测程序

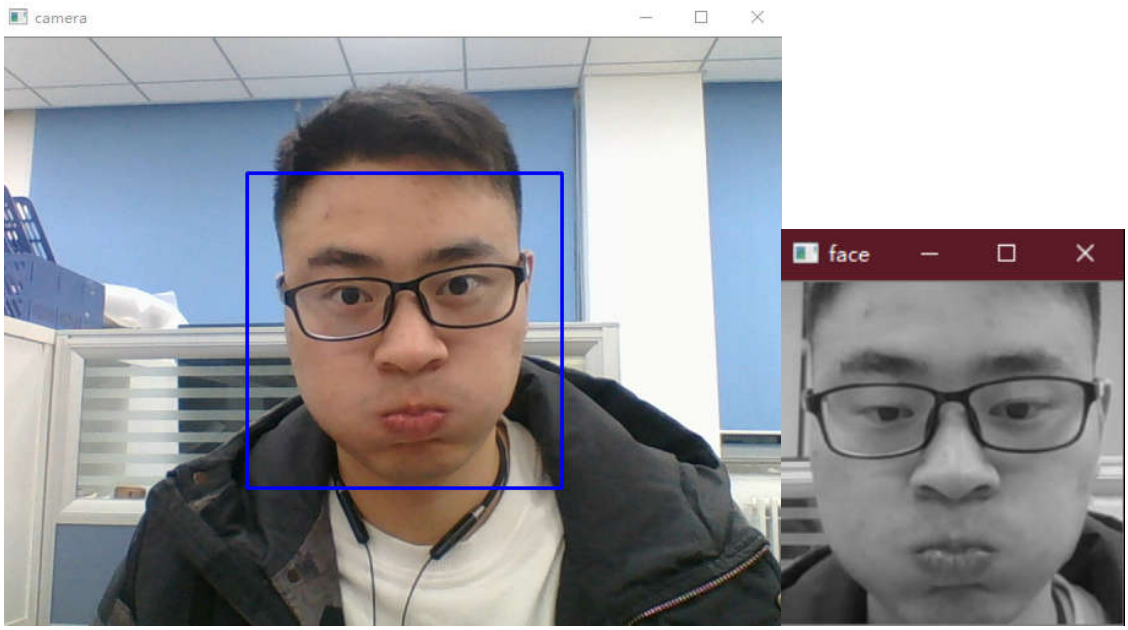

图 2-2. 人脸检测图像

2.2 基于 face_recognition 开源库的人脸检测

基于 Harr 级联的人脸检测优点就是简单易理解，但是这些级联有个弊端就是必须用于正面、直立的人脸图像，当被检测人歪着头就很容易检测不到。而 face_recognition 应用了 dlib——一个现代 C++工具包，其中包含了一些机器学习算法来帮助完成复杂的基于 C++的应用，可以完成大量任务，发现图片中所有的脸，发现并处理图片中的脸部特征，识别图片中的脸，实时的人脸识别，而且对于歪着的脸也有很高的检测能力。

人脸对齐，face alignment 指在标定人脸位置后对人脸上的特征进一步定位，可以对人脸检测的侧脸进行校正，旋转，3d 变换等，也可以对特征点进行特定的表情变化。常用的有 5 点和 68 点特征。如 DCNN，TCDCN，MTCNN 等方法。我们通过获取两眼的中心坐标，计算出倾斜角度，计算出仿射矩阵，再进行旋转变换即可。考虑到背景问题，旋转后空出来的背景用灰色常数值填充。

我们将使用 face_recognition 库对捕获图像采集人脸，并进行对齐处理。程序 detect_face_recognition.py 就是实现了人脸检测，并对齐。如图 2-3 是人脸对齐程序：

```
def face_alignment(faces):
    faces_aligned = []# 预测关键点
    for face in faces:
        rec = dlib.rectangle(0, 0, face.shape[0], face.shape[1])
        shape = predictor(np.uint8(face), rec)
        # left eye, right eye, nose, left mouth, right mouth
        order = [36, 45, 30, 48, 54]
        for j in order:
            x = shape.part(j).x
            y = shape.part(j).y
        eye_center = ((shape.part(36).x + shape.part(45).x) * 1. / 2,  # 计算两眼的中心坐标
                      (shape.part(36).y + shape.part(45).y) * 1. / 2)
        dx = (shape.part(45).x - shape.part(36).x)
        dy = (shape.part(45).y - shape.part(36).y)
        angle = math.atan2(dy, dx) * 180. / math.pi     # 计算角度
        RotateMatrix = cv2.getRotationMatrix2D(eye_center, angle, scale=1) # 计算仿射矩阵
        # 进行仿射变换，即旋转
        RotImg = cv2.warpAffine(face, RotateMatrix, (face.shape[0], face.shape[1])
                                ,borderMode=cv2.BORDER_CONSTANT, borderValue=(125, 125, 125))
        cv2.imshow("Align_face", RotImg)
        faces_aligned.append(RotImg)
    return faces_aligned
```

图 2-3. 人脸对齐程序

predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat"),

这里主要用了预测关键点函数，获得眼睛、鼻子、嘴巴的坐标。接下来就是一些数学计算，算出两眼中心坐标的斜率，计算出偏斜角度，利用cv2.warpAffine()函数进行旋转，并对空余部分进行灰度填充。至于人脸检测，与基于 Harr 的程序大同小异，具体见附录 detect_face_recognition.py。我们在测试中显示三张图像分别是：整张图片，采集到的人脸，对齐后的人脸，如图 2-4 所示。

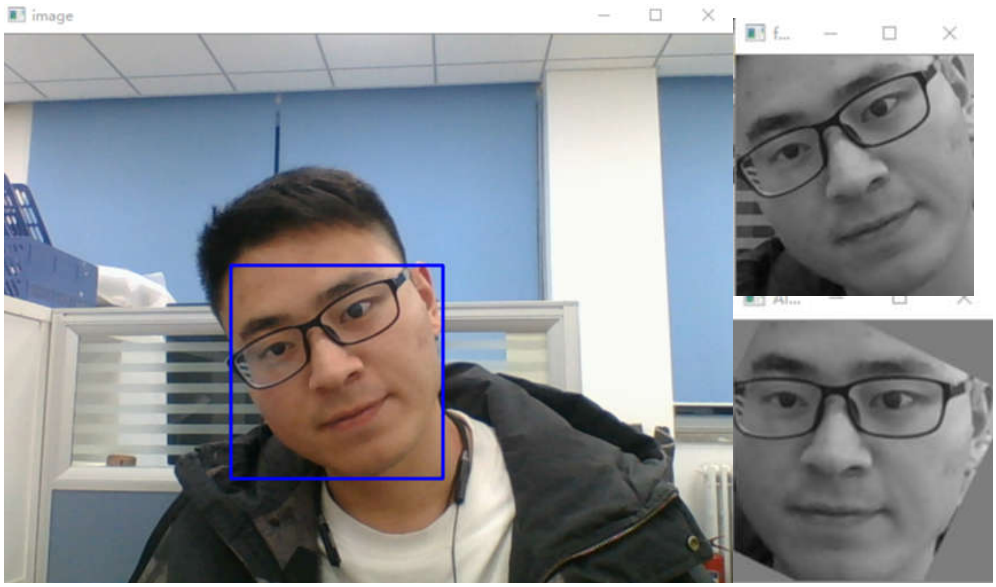

图 2-4. 基于 face_recognition 开源库的人脸检测、对齐图片

通过比较两种人脸检测方法，基于 face_recognition 开源库的人脸检测能识别到场景更差的人脸，效果更好。下面我们将根据这些图片做成我们自己的数据集，并进行人脸识别训练。

3、人脸识别技术

本章节主要分为三个部分：先采用上一章节内容采集人脸数据做成数据集，再利用数据集训练 Harr 人脸识别模型，最终在摄像头读取实时识别人脸。本章的所有程序依赖环境与上一章相同。

3.1 构造人脸数据集

人脸数据集的要求每个人脸单独一个文件夹，并将文件夹以人名命名，在总的文件夹下记录每个人名，方便人脸识别时，显示人名。人脸数据的采

集是基于上一章的人脸检测内容，只是将检测到的人脸切割出来存储为 TXT 文件。同第二章节一样，人脸检测也有两种方案，但是本章只是用 Harr 级联检测作为示例。我们在测试阶段使用 pycharm 平台开发，数据集制作流程如图 3-1 所示。

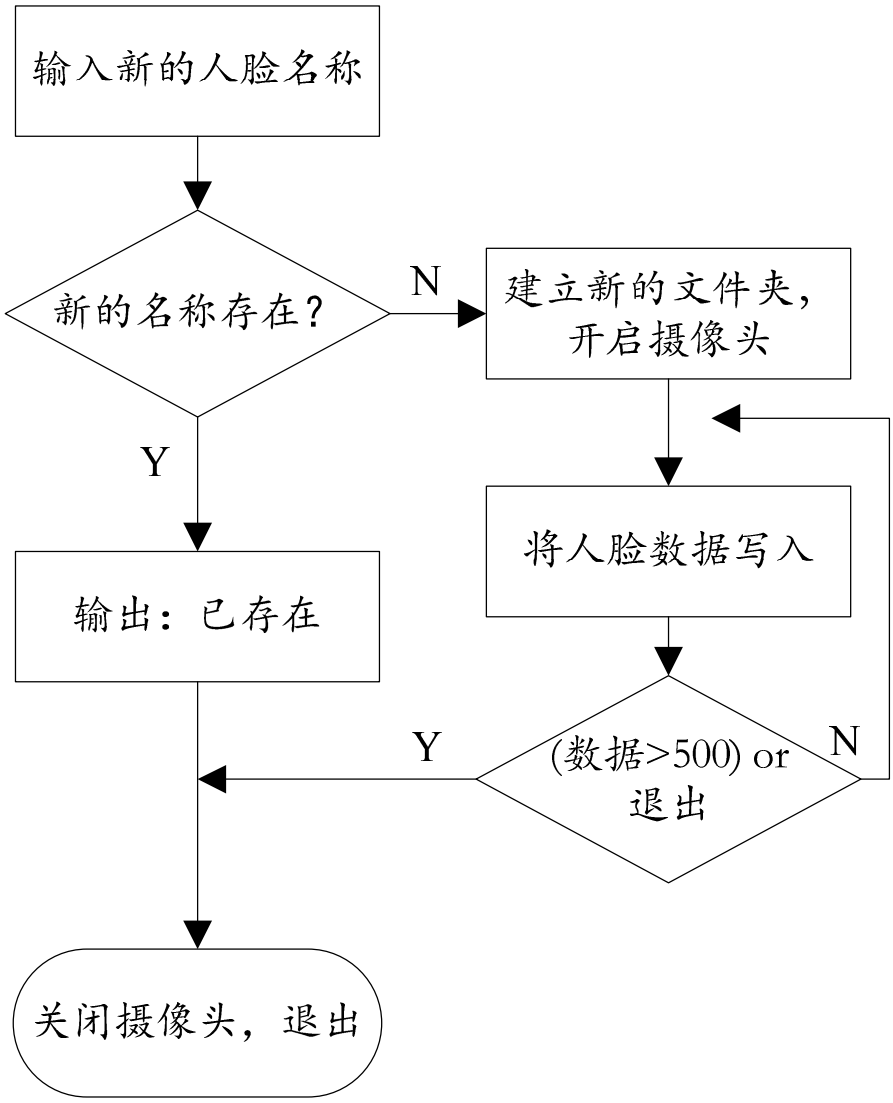

图 3-1. 人脸数据采集流程图

程序中的关于文件的处理程序是基于 Python 的 OS 库函数实现，函数 os.path.exists(file_name)判断名称是否存在，os.mkdir(file_name)函数用于创建一个新的文件夹。其主要程序见附录 face_data_collect.py。

所以，在数据采集过程中我们只需要先输入名称，等待一些时间等待数据自动采集完成即可，数据收集过程如图 3-2 所示。需要注意的是：尽量只

有一个人对准摄像头，因为使用的是 Harr 级联检测，尽量是头保持端正。

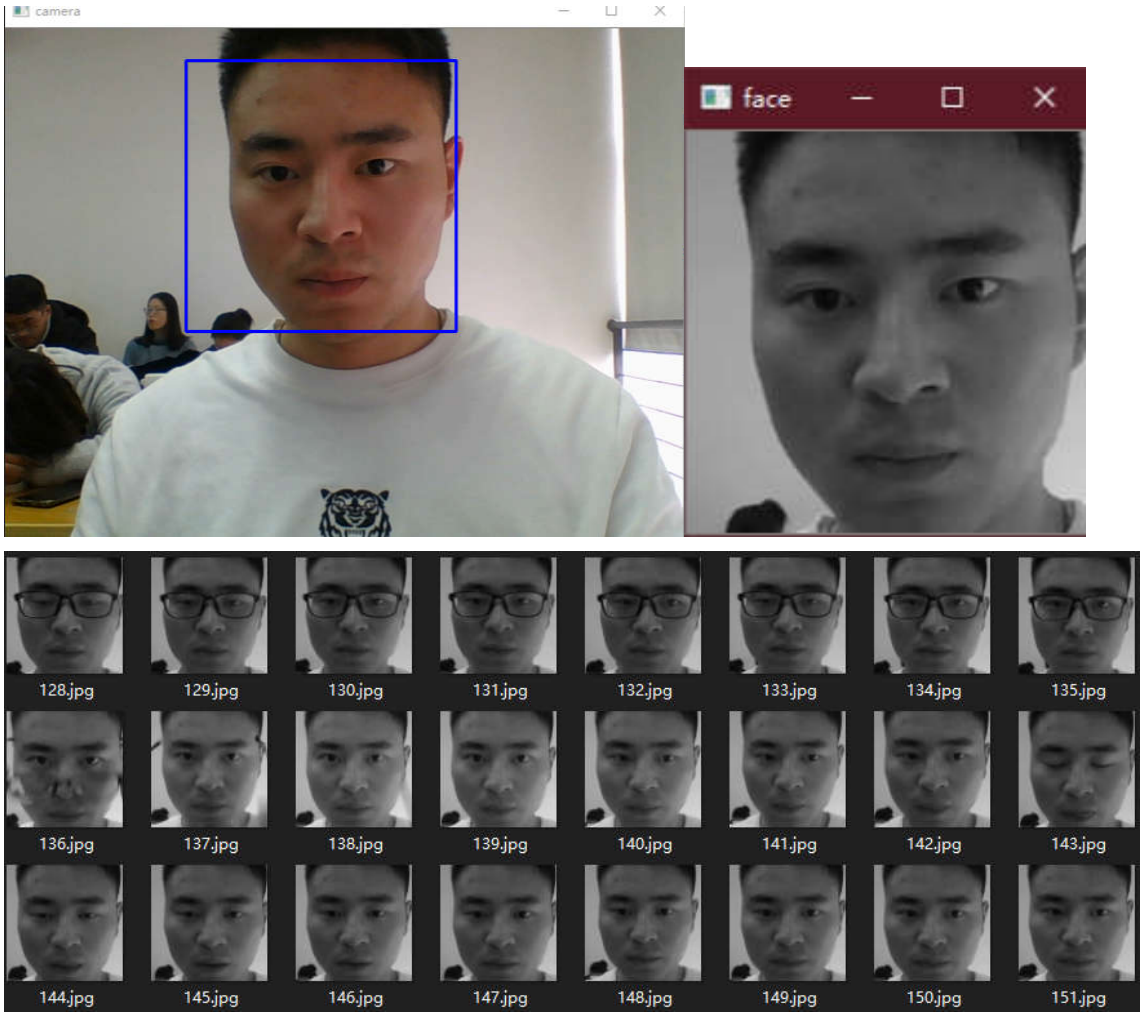

图 3-2. 人脸数据收集过程图

3.2 模型训练

模型训练是将上一章所采集的数据作为输入，训练出关于我们自己的模型，主要包含两部分内容，图像加载和训练识别模型。

人脸库确定之后需要进行训练，即让计算机“学习”这些人脸样本。这时面临的一个问题就是如何把训练样本读进内存中。opencv 手册中明确说明 LBPHFace 的训练函数的入口参数是一个图像容器，容器中包含所有训练图像。那么如何创建一个这样的容器并把训练样本全部放进去呢？加载图像就是为了解决这个问题，为数据集添加标签等。主要包含读取样本文件，并加载到一个列表里，返回值包括 2 部分，【文件列表，对应标签】，标签用来对

照姓名。最终把所有的名称写入文件保存。加载图像流程图如图 3-3 所示。加载数据主要利用 numpy 库和 os 库中的函数读取数据、打开文件等具体程序见附录 train_face.py 中的 read_image(path)函数。这样处理以后就可以将返回值直接输入训练模型。

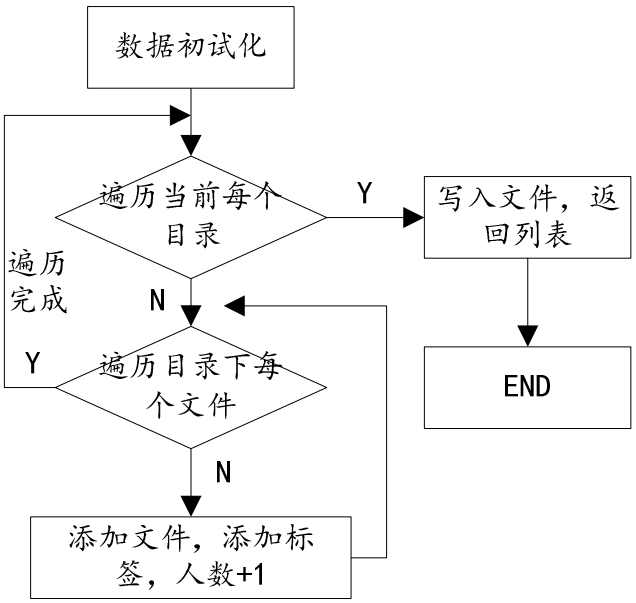

图 3-3. 数据加载流程图

OpenCV 中目前支持的 3 种人脸识别的方案：特征脸 EigenFace、Fisher 脸 FisherFace、LBP 直方图 LBPHFace。分别调用函数 createEigenFaceRecognizer、createFisherFaceRecognizer、createLBPHFaceRecognizer 建立模型。对于 EigenFace 两个输入参数，分别为 PCA 主成分的维数 num_components 和预测时的阈值 threshold，主成分这里没有一个选取的准则，要根据输入数据的大小而决定，通常认为 80 维主成分是足够的。除了这两个输入参数外，还有 eigenvalues 和 eigenvectors 分别代表特征值和特征向量，mean 参数为训练样本的平均值，projections 为训练数据的预测值，labels 为预测时的阈值。对于 FisherFace，和 EigenFace 非常相似，也有 num_components 和 threshold 两个参数和其他 5 个参数，FisherFace 的降维是 LDA 得到的。默认值为 c-1，如果设置的初始值不在（0，c-1]的范围内，会自动设定为 c-1。特别需要强调的是，EigenFace 和 FisherFace 的训练图像和测试图像都必须是灰度图，而且是

经过归一化裁剪过的。对于 LBPHFace，LBP 简单和效果是大家都很喜欢的，参数包括半径 radius，邻域大小即采样点个数 neighbors，x 和 y 方向的单元格数目 grid_x,grid_y，还有两个参数 histograms 为训练数据得到的直方图，labels 为直方图对应的标签，这个方法也要求训练和测试的图像是灰度图。本系统选用 LBP 直方图 LBPHFace 方案。当然我们也不需要每次使用都进行一次训练，可以把训练好的模型通过 save 函数保存成一个文件，下次使用的时候只需 load 即可。训练数据如图 3-4 所示。

```
def train_model(path, epoch=5):

    print('Training faces. It will take a few seconds. Wait ...')
    faces, ids = read_image(path)
    for s in range(1, epoch):
        print('Training ...  %dth' % s)
        if os.path.exists('trainer.yml'):#是否存在文件
            recognizer.read('trainer.yml')  #读取模型参数
        recognizer.train(faces, np.array(ids))  #训练模型
        recognizer.write('trainer.yml')         #保存模型参数
    print('Complete the training !')
```

图 3-4. LBPHFace 方案模型训练

3.3 实时人脸识别

有了前面的基础，实时人脸识别也就简单了，首先我们对摄像头捕获帧进行基本处理，灰度化、人脸检测、人脸对齐等，然后加载人脸识别模型，加载训练好的参数，最后将处理后的图像输入到训练好的模型中，利用人脸预测函数 params = recognizer.predict(roi_gray)，返回参数有两个，params[0]表示识别后的标签， params[1]表示置信度评分。对图像或视频中检测到的人脸进行分析，并从两方面来确定：是否识别到目标；目标真正被识别到的置信度的衡量，这也称为置信度评分，在实际应用中可以通过设置阈值来进行筛选，置信度高于该阈值的人脸将会被丢弃。通常 params[1] 低于 50 以下是好的识别参考值，高于 80 的参考值都会认为是低的置信度评分。我们可以设当置信评分低于 70 时，在图像上输出检测人名。人脸实时识别过程如图 3-5 所示。

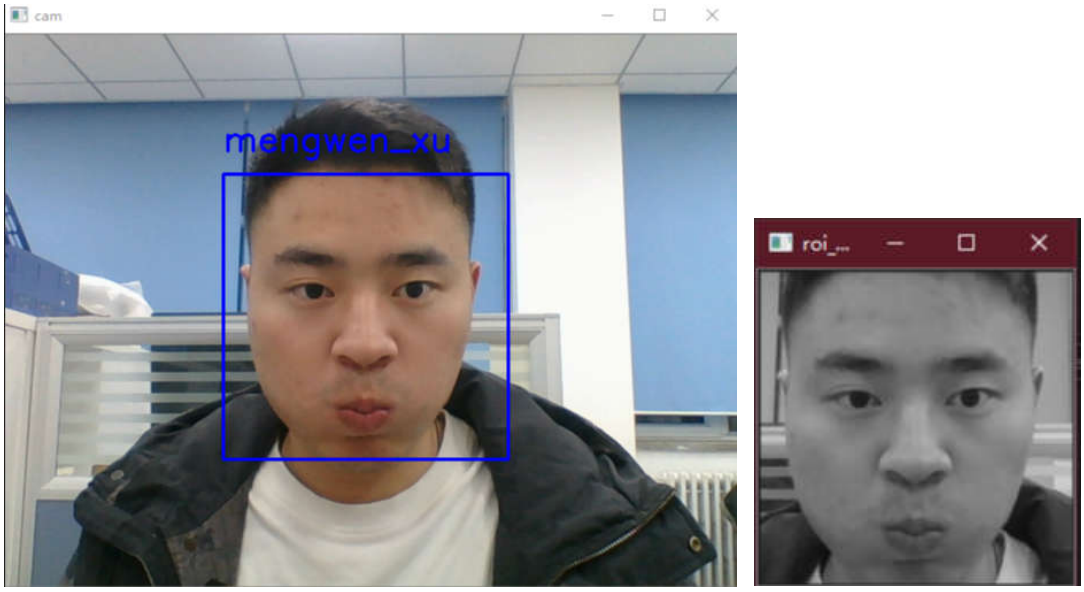

图 3-5. 人脸实时识别过程

4、基于 PyQt5 的应用平台

在上面几个过程中我们已经实现了人脸实时检测和识别，并且在后台输出信息，但是这个界面效果不太好看，使用起来还要一个文件一个文件的去点，不方便。所以我们使用 PyQt5 库搭建 GUI 界面，并于之前的程序相结合打造一款不错的应用。

PyQt5 是一套 Python 绑定 Digia QT5 应用的框架，Qt 库是最强大的 GUI 库之一。为什么选择 qt 还有一个重要原因就是在 python 中依然可以使用 Qt Designer 来设计界面，这会很方便我们的调试过程。设计应用界面如图 4-1 所示。我们只需要编写简单的响应函数就可以把现在的界面和之前的程序串接起来。首先是我们的训练模型，因为在程序调试过程中我们也是直接调用的函数，所以这里直接在按键响应函数下调用即可，如图 4-2 所示。

```
def btnTrainData_Clicked(self):
    #调用训练数据模型
    self.printf('Training faces. It will take a few seconds. Wait ...')
    train_model(picture_path, 5)
    self.printf('Complete the training ! ')
```

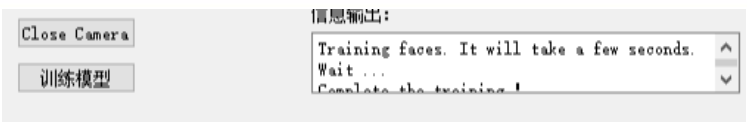

图 4-2. 训练模型函数的串接与结果

图 4-1. 界面预览图

另外还有最重要的就是人脸识别显示，这里对原函数做些简单改变，原来是调用 opencv 里面的显示函数，这时不需要了，直接把得到的图像返回。在 GUI 界面接收到图像再把它显示到 Label 控件上。有几点需要注意，图片需要镜像显示；opencv 的颜色格式与 pyqt 也不同；图像大小要根据界面控件大小调整。我们在“Open Camera”按钮响应函数中设置定时器打开，定时每 30ms 调用刷新函数采集图像。调用程序如图 4-3 所示。

```
@QtCore.pyqtSlot()
def _queryFrame(self):
    # 循环捕获图片
    self.frame = recognition.face_rec(self.camera)
    img_rows, img_cols, channels = self.frame.shape#输出图像形状
    bytesPerLine = channels * img_cols
    cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB, self.frame)#转变成RGB颜色格式
    QImg = QImage(self.frame.data, img_cols,
                  img_rows,bytesPerLine, QImage.Format_RGB888)#装变成qt颜色格式
    #根据形状适应大小，并输出在Label上
    self.label.setPixmap(QPixmap.fromImage(QImg).scaled(self.label.size(),
                         Qt.KeepAspectRatio, Qt.SmoothTransformation))
```

图 4-3. 人脸识别与 GUI 串接程序

到这里算是基本完成了我们的人脸识别系统的设计，在开发环境界面我们可以实现人脸检测、人脸对齐等单项操作，而在 GUI 界面中我们将他们合

为一体。总体操作界面如图 4-4 所示。

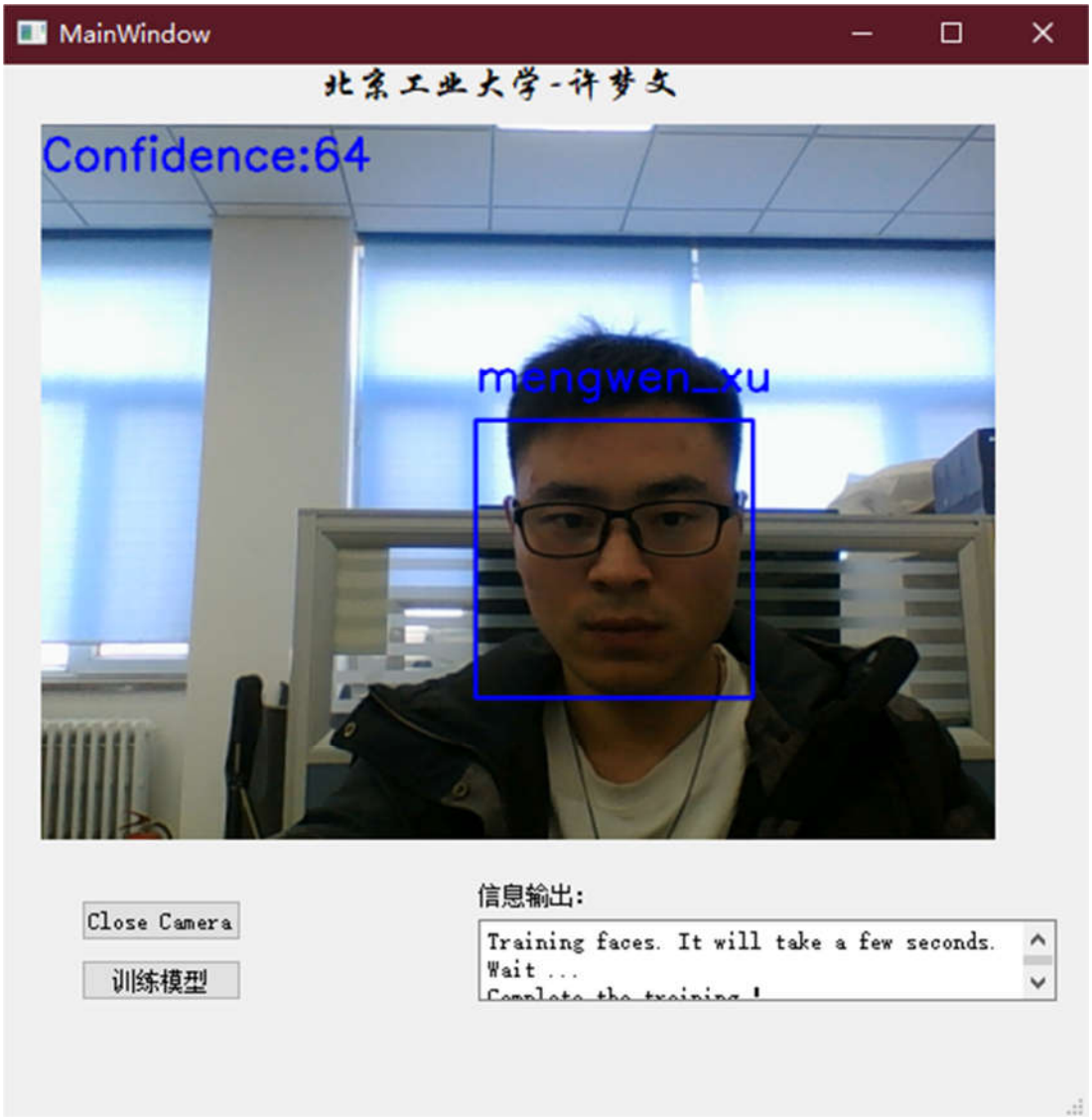

图 4-4. 系统总体界面

总结

人脸识别具有重大的理论意义和应用意义，它是一项结合了多学科，多领域知识方法的技术。长期以来，如何利用计算机进行准确，快速的人脸识别，一直是图像处理的研究热点与难点。社会的发展促进了身份认证技术市场的急速扩大，人脸识别因其自身的优点，在身份认证中的使用日益频繁。 人脸识别技术具有广泛的社会需求和市场前景。本文通过人脸识别的过程，层层搭建完成了简单的人脸识别系统。也比较了在问题中的不同解决办法，详细

介绍了人脸识别系统的构成与工作以及其中的一些关键性问题。概述了人脸识别技术的应用及其难点，研究内容与主要方法，比较了人脸检测的不同方法，介绍了人脸识别中会使用到的各种分类器。详细介绍了人脸识别的流程、、特征脸方法（PCA），使用 LBPHFace 方法设计出人脸识别程序，并集成在 GUI 界面上。

最后，特别感谢王亮老师对我们这一学期的教导，您的课整堂课思路清晰，环节紧凑，重难点突出。再次感谢老师的教导。

附录

Detect_face_harr.py

```
import cv2
# minW = camera.get(3)  # 图像宽度  480
# minH = camera.get(4)  # 图像高度  640
#开启摄像头，采集图像，并截取脸部存起来作为数据集
def Detect_Face():
    # opencv 自带的人脸识别分类器
    face_cascade = cv2.CascadeClassifier('.\opencv_module\haarcascade_frontalface_default.xml')
    # 调用笔记本内置摄像头，所以参数为 0，如果有其他的摄像头可以调整参数为 1，2
    camera = cv2.VideoCapture(0)
    while (True):
        ret, frame = camera.read()#获取图像
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # 灰度化
        faces = face_cascade.detectMultiScale(gray, 1.3, 5,minSize=(60,60))  # 人脸检测
        for (x, y, w, h) in faces:
            img = cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
            f = cv2.resize(gray[y:y + h, x:x + w], (200, 200))
            cv2.imshow("face", f)
        cv2.imshow("camera", frame)
        if cv2.waitKey(int(1000 / 12)) & 0xff == ord("q"):
            break
    camera.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    Detect_Face()
```

Detect_face_recognition.py

```
import dlib
import face_recognition
import math
import numpy as np
import cv2
#人脸检测+对齐
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
def rect_to_bbox(rect):
    """获得人脸矩形的坐标信息"""
    # print(rect)
    x = rect[3]
    y = rect[0]
    w = rect[1] - x
    h = rect[2] - y
    return (x, y, w, h)

def face_alignment(faces):
    faces_aligned = []# 预测关键点
    for face in faces:
        rec = dlib.rectangle(0, 0, face.shape[0], face.shape[1])
        shape = predictor(np.uint8(face), rec)
        # left eye, right eye, nose, left mouth, right mouth
        order = [36, 45, 30, 48, 54]
        for j in order:
            x = shape.part(j).x
            y = shape.part(j).y
        eye_center = ((shape.part(36).x + shape.part(45).x) * 1. / 2,  # 计算两眼的中心坐标
                      (shape.part(36).y + shape.part(45).y) * 1. / 2)
        dx = (shape.part(45).x - shape.part(36).x)
        dy = (shape.part(45).y - shape.part(36).y)
        angle = math.atan2(dy, dx) * 180. / math.pi        # 计算角度
        RotateMatrix = cv2.getRotationMatrix2D(eye_center, angle, scale=1) # 计算仿射矩阵
```

```
            # 进行仿射变换，即旋转
            RotImg = cv2.warpAffine(face, RotateMatrix, (face.shape[0], face.shape[1])
                                    ,borderMode=cv2.BORDER_CONSTANT, borderValue=(125, 125,125))
            faces_aligned.append(RotImg)
        return faces_aligned


def test( ):
    camera = cv2.VideoCapture(0)
    while (True):
        read, img = camera.read()
        # 定位图片中的人脸
        face_locations = face_recognition.face_locations(img)
        # 提取人脸区域的图片并保存
        src_faces = []
        src_face_num = 0
        for (i, rect) in enumerate(face_locations):
            src_face_num = src_face_num + 1
            (x, y, w, h) = rect_to_bbox(rect)
            detect_face = img[y:y + h, x:x + w]
            detect_face = cv2.cvtColor(detect_face, cv2.COLOR_BGR2GRAY)
            src_faces.append(detect_face)
            cv2.imshow("face", detect_face)
            #用框框圈住人脸
            img = cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
            cv2.imshow("image", img)
        # 人脸对齐操作并保存
        faces_aligned = face_alignment(src_faces)
        face_num = 0
        for faces in faces_aligned:
            face_num = face_num + 1
            cv2.imshow("Align_face", faces)
        if cv2.waitKey(int(1000/12)) & 0xff == ord("q"):
            break
    camera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    test( )
    print(" SUCCEED !!! ")
```

face_data_collect.py

```
import cv2
import os

#开启摄像头，采集图像，并截取脸部存起来作为数据集
def generate_data():
    # opencv 自带的人脸识别分类器
    face_cascade = cv2.CascadeClassifier('.\opencv_module\haarcascade_frontalface_default.xml')
    count = 0   # 计数，采取图像照片
    new_name = input('\n enter user id:')
    file_name = './data_collection/' + new_name
    if not os.path.exists(file_name):   # 如果目录不存在
        os.mkdir(file_name)# 以新的名字创建新的目录
        # 调用笔记本内置摄像头，所以参数为 0，如果有其他的摄像头可以调整参数为 1，2
        camera = cv2.VideoCapture(0)
        while (True):
            ret, frame = camera.read()#获取图像
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)   # 灰度化
            faces = face_cascade.detectMultiScale(gray, 1.3, 5,minSize=(60,60))   # 人脸检测
            for (x, y, w, h) in faces:
                img = cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
                f = cv2.resize(gray[y:y + h, x:x + w], (200, 200))
                cv2.imshow("face", f)
                cv2.imwrite(file_name + '/%s.pgm' % str(count), f)
```

```
                count += 1
            cv2.imshow("camera", frame)
            if count >= 500:
                break
            if cv2.waitKey(int(1000 / 12)) & 0xff == ord("q"):
                break
        camera.release()
        cv2.destroyAllWindows()
    else:
        print("名称已存在")

if __name__ == "__main__":
    generate_data()
```

train_face.py

```
import numpy as np
import os
import cv2
picture_path = 'data_collection'# 人脸数据路径
recognizer = cv2.face.LBPHFaceRecognizer_create()#cv 中自带的识别模型
detector = cv2.CascadeClassifier('.\opencv_module\haarcascade_frontalface_default.xml')#cv 中的检测人脸文件
face_names = []
#读取样本文件，并加载到一个列表里，返回值包括 2 部分，[文件列表，对应标签],标签用来对照姓名用。
def read_image(path,sz=None):
    pr_img=[] #图像列表
    pr_flg=[] #对应标签
    pr_count=0 #初始化检测到的人数
    for dirname,dirnames,filenames in os.walk(path):#遍历当前程序目
        for subdirname in dirnames: #遍历程序文件夹下的各个目录
            subject_path=os.path.join(dirname,subdirname)
  #这时候 subdirname 是循环处理 data_collection 文件夹下的文件夹，这时如果有新的面孔加进来需要更新标签名字
            face_names.append(subdirname)
            for filename in os.listdir(subject_path): #遍历文件夹下文件
                try:
                    filepath=os.path.join(subject_path,filename)
                    im=cv2.imread(filepath,cv2.IMREAD_GRAYSCALE) #读取文件下 PGM 文件
                    if im.shape!=(200,200): #判断像素是否 200
                        im=cv2.resize(im,(200,200))
                    pr_img.append(np.asarray(im,dtype=np.uint8)) #添加图像
                    pr_flg.append(pr_count)#添加标签
                except:
                    print("io error")
            pr_count+=1 #另一个人的标签
    with open('pace_names.txt', 'w') as f: #把名字写入文件
        f.write(' '.join(face_names)) #每个元素间加一个空格转换为字符串
    return [pr_img, pr_flg]

def train_model(path, epoch=5):
print('Training faces. It will take a few seconds. Wait ...')
faces, ids = read_image(path)
if os.path.exists('trainer.yml'):#是否存在文件
        recognizer.read('trainer.yml')   #读取模型参数
for s in range(1, epoch):
    print('Training ...   %dth' % s)
    recognizer.train(faces, np.array(ids))   #训练模型

recognizer.write('trainer.yml')            #保存模型参数
print('Complete the training  ! ')


if __name__ == "__main__":
    train_model(picture_path, 10)
```

recognition.py

```
import cv2
import face_data_collect
import dlib
import face_recognition
import detect_face_recognition as dfr
def rect_to_bbox(rect):
    """获得人脸矩形的坐标信息"""
    x = rect[3]
    y = rect[0]
    w = rect[1] - x
    h = rect[2] - y
    return (x, y, w, h)

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer.yml')
face_cascade = cv2.CascadeClassifier('.\opencv_module\haarcascade_frontalface_default.xml')
font = cv2.FONT_HERSHEY_SIMPLEX
idnum = 0

def face_rec():
    # 下面读取摄像头图像，用矩形标识检测到脸部和训练后结果比对，打印出对应标签所对应名字
    with open('pace_names.txt', 'r') as f:
        face_names = f.read()
    names = face_names.split()
    camera = cv2.VideoCapture(0)
    while(True):
        read, img = camera.read()
        face_locations = face_recognition.face_locations(img)   # 定位图片中的人脸
        src_faces = []
        src_face_num = 0
        for (i, rect) in enumerate(face_locations):
            src_face_num = src_face_num + 1
            (x, y, w, h) = rect_to_bbox(rect)#读取单个人脸的初始坐标与宽度，高度
            detect_face = img[y:y + h, x:x + w]#截取出人脸部分
            detect_face = cv2.cvtColor(detect_face, cv2.COLOR_BGR2GRAY)#灰度化
            src_faces.append(detect_face)
            cv2.imshow("face", detect_face)
            # 用框框圈住人脸，注意位置偏移
            img = cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
            cv2.imshow("image", img)
        # 人脸对齐操作并保存
        faces_aligned = dfr.face_alignment(src_faces)
        face_num = 0
        for roi_gray in faces_aligned:
            face_num = face_num + 1
            cv2.imshow("Align_face", roi_gray)


        # #harr 方法检测人脸
        #由于读取的摄像头大小为 640x480，所以设置人脸最小为 60x60
        # faces = face_cascade.detectMultiScale(img, 1.3, 5,minSize=(60,60))
        # for (x, y, w, h) in faces:
        #     img = cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        #     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        #     roi_gray = gray[y:y + h,x:x + w]#注意别弄错了位置


            try:
                roi_gray = cv2.resize(roi_gray, (200,200), interpolation=cv2.INTER_LINEAR)
                cv2.imshow("roi_gray", roi_gray)
                params = recognizer.predict(roi_gray)
                print("Label:%s,Confidence:%.2f" % (params[0], params[1]))
                if params[1] < 45:#低于 50 以下的才勉强可以
                    cv2.putText(img, names[params[0]],(x,y-20),cv2.FONT_HERSHEY_SIMPLEX, 1,255,2)
```

```
                except:
                    continue

            cv2.imshow("cam",img)
            if cv2.waitKey(int(1000/12)) & 0xff == ord("q"):
                break
        camera.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
        face_rec( )
```