西安工业大学

XI`AN TECHNOLOGICAL UNIVERSITY

人工智能大作业

基于 YOLOV5 算法与质心追踪算法的戴口罩人脸检测识别系统

专　　业：软件工程

班　　级：20060213

姓　　名：李小龙 刘郅喆 李玉鑫
林堉泽 刘盛林 李泽鹏

2023 年 5 月 27 日

小组成员分工安排

（1）小组负责人：李小龙 2020032936

分工：负责组员的整体任务调度，系统整体架构设计，以及核心算法的设计和实现。

（2）小组成员 1：刘郅喆 2020032941

分工：负责核心算法的部分设计，系统原型设计，Qt 前端设计。

（3）小组成员 2：李玉鑫 2020032937

分工：负责查找相关论文资料，整体论文内容，撰写论文，进行系统测试。

（4）小组成员 2：林堉泽 2020032939

分工：负责算法训练和调试，数据库设计和实现。

（5）小组成员 2：刘盛林 2020032940

分工：负责算法训练，后端业务逻辑实现。

（6）小组成员 2：李泽鹏 2020031822

分工：负责系统调试和测试，提交 bug 并进行日志管理。

摘要

自 2019 年新型冠状病毒肺炎爆发以来，全球抗击疫情形势异常严峻。为了控制疫情的快速蔓延，政府要求出入公共场所必须要佩戴口罩，而这对现有的人脸识别系统识别准确率就造成了很大的困难。针对这一问题，本文提出一种基于 YOLOv5 目标检测算法与质心追踪算法(Centriod tracking)的戴口罩人脸检测识别系统。

本系统采用质心追踪算法来锁定视频中的图像，追踪视频首帧和后续帧的人脸信息，结合 YOLOv5 算法检测并提取人脸特征，提高了人脸识别系统的性能和精准性。同时，利用 FaceNet 算法实现对戴口罩人脸的识别。在前端界面设计方面，本系统使用 PySide2 模块和 Qt 工具，实现了界面与代码分离，降低了耦合性，提高了开发效率。

本文主要介绍了本系统的算法原理、原型设计、功能模块设计、数据库设计、实现方法以及应用前景等。经过 12 小时的训练，本系统的识别准确率可达到 89.8%，并且随着训练时间的增加，识别准确率还有进一步提升的空间。

【关键词】：YOLOv5 算法、质心追踪算法、FaceNet 算法、PySide2、Qt

Based on YOLOv5 Algorithm and Centroid Tracking Algorithm Wear a Mask Face Detection and Recognition System

Abstract

Since the outbreak of the COVID-19 in 2019, the global fight against the epidemic has been extremely grim. In order to control the rapid spread of the epidemic, the government has required masks to be worn in public places, which has caused great difficulties in the recognition accuracy of the existing face recognition system. To solve this problem, this paper proposes a face mask detection and recognition system based on the YOLOv5 target detection algorithm and Centriod tracking algorithm.

This system uses centroid tracking algorithm to lock the image in the video, track the face information of the first and subsequent frames of the video, and combine with YOLOv5 algorithm to detect and extract the face features, improving the performance and accuracy of the face recognition system. At the same time, FaceNet algorithm is used to recognize faces wearing masks. In the aspect of front-end interface design, this system uses PySide2 module and Qt tool to realize the separation of interface and code, reduce the coupling and improve the development efficiency.

This paper mainly introduces the algorithm principle, prototype design, functional module design, database design, implementation method and application prospect of this system. After 12 hours of training, the recognition accuracy of this system can reach 89.8%, and with the increase of training time, there is room for further improvement of the recognition accuracy.

【Key Words】: YOLOv5 Algorithm,Centroid Tracking Algorithm,FaceNet Algorithm, PySide2,Qt

目录

第一章 绪论

本章主要介绍了本文的研究背景、国内外研究现状、研究内容和应用前景。首先，分析了新冠肺炎疫情下戴口罩人脸检测识别的重要性和意义，以及相关领域的应用需求。其次，综述了国内外关于戴口罩人脸检测识别的研究现状，指出了目前存在的问题和挑战。然后，阐述了本文的研究内容，即基于 YOLOv5 算法和质心追踪算法的戴口罩人脸检测识别系统的设计与实现。最后，展望了本文的应用前景，主要包括门禁考勤、金融支付和公共安防等方面。

1.1 研究背景

人脸识别是一种基于人脸特征进行身份识别的技术，它具有方便快捷、安全可靠、无需接触等优点，广泛应用于门禁考勤、金融支付、公共安防等领域。然而，自 2019 年底以来，新冠肺炎疫情在全球范围内爆发并持续蔓延，给人类社会带来了巨大的危机和挑战。为了防止病毒的传播，各国政府和卫生机构都推荐或要求人们在公共场所佩戴口罩。这对人脸识别技术提出了新的要求和挑战，即如何在戴口罩的情况下，仍然能够准确地检测和识别人脸。

目前，市场上存在的大多数人脸识别系统都是基于深度学习的卷积神经网络[1](CNN)模型，它们通常需要大量的清晰完整的人脸图像作为训练数据，以提高模型的泛化能力和识别精度。然而，在戴口罩的情况下，人脸图像的有效信息被大幅度减少，导致模型的性能下降。此外，由于口罩的形状、颜色、材质等因素的多样性，以及不同人群对口罩佩戴方式的差异，使得戴口罩人脸图像具有较高的复杂度和多样性，给模型的训练和测试带来了更大的难度。

因此，如何设计一个能够在戴口罩情况下，实现高效准确的人脸检测识别系统，是当前亟待解决的问题。本文基于 YOLOv5 算法与质心追踪算法，提出了一种戴口罩人脸检测识别系统。YOLOv5 算法是一种基于锚点框回归[2]和分类的端到端目标检测算法[3]，它具有速度快、精度高、模型轻量等特点。质心追踪算法是一种基于质心距离匹配的目标追踪算法，它具有简单有效、鲁棒稳定、适应性强等特点。本

文将 YOLOv5 算法与质心追踪算法相结合，实现了对戴口罩人脸的检测和追踪，并利用 OpenCV 库、Dlib 库和 FaceNet 算法实现了对戴口罩人脸的对齐、特征提取和相似度计算，从而完成了对戴口罩人脸的识别。

1.2 研究现状

1.2.1 国外研究现状

人脸识别是计算机视觉领域的一个重要研究方向，它涉及到人脸检测、人脸对齐、人脸特征提取、人脸匹配等多个子任务。随着深度学习技术的发展，人脸识别的性能也得到了显著的提升。目前，国外的人脸识别技术主要分为两大类：基于传统机器学习的方法和基于深度神经网络的方法。

基于传统机器学习的方法主要利用手工设计的特征描述符，如局部二值模式[4]、方向梯度直方图[5]、局部相位量化[6]等，来提取人脸图像的纹理信息，并结合不同的分类器，如支持向量机[7]、K 近邻[8]、随机森林[9]等，来实现人脸识别。这类方法的优点是计算量相对较小，适合于低端设备或实时应用。但是，这类方法也存在一些缺点，如特征描述符对光照、表情、姿态等变化敏感，导致识别准确率下降；特征描述符难以捕捉人脸图像的高层语义信息，导致识别效果受限。

基于深度神经网络的方法主要利用卷积神经网络或其他类型的神经网络，如循环神经网络[10]、注意力机制[11]等，来自动学习人脸图像的特征表示，并结合不同的损失函数，如交叉熵损失[12]、三元组损失[13]、余弦相似度损失[14]等，来优化人脸识别的目标函数。这类方法的优点是能够提取人脸图像的深层语义信息，具有较强的泛化能力和鲁棒性。但是，这类方法也存在一些缺点，如计算量较大，需要大量的训练数据和计算资源；网络结构和参数选择较为复杂，需要专业的知识和经验。

在新冠肺炎疫情期间，由于佩戴口罩成为了防控疫情的重要措施之一，给现有的人脸识别系统带来了新的挑战。国外针对戴口罩人脸识别问题，有以下几种主要的研究方向：

基于多模态信息融合的方法。这类方法主要利用除了可见光图像之外的其他模态信息，如红外图像、深度图像、声音等，来增强戴口罩人脸识别系统的性能。例如，S. Wang[15]等提出了一种基于红外图像和可见光图像的跨模态人脸识别方法，利

用一个共享的卷积神经网络来提取两种模态的人脸特征，并通过一个对齐模块来实现两种模态的特征对齐和融合。

基于数据增强的方法。这类方法主要利用一些图像处理技术，如遮挡、旋转、缩放、裁剪、噪声等，来模拟不同的口罩佩戴情况，从而扩充训练数据集，提高戴口罩人脸识别系统的泛化能力。例如，A. Majumder[16]等提出了一种基于 GAN 的数据增强方法，利用生成对抗网络(GAN)来生成不同样式和颜色的口罩，并将其叠加到人脸图像上，从而生成大量的戴口罩人脸图像。

基于特征选择或降维的方法。这类方法主要利用一些特征选择或降维技术，如主成分分析、线性判别分析、稀疏表示等，来提取或保留人脸图像中与身份相关的特征，从而降低口罩对人脸识别系统的影响。例如，M. H. Alkhatib[17]等提出了一种基于稀疏表示的方法，利用稀疏表示技术来重建戴口罩人脸图像中被遮挡的部分，并将重建后的图像用于人脸识别。

1.2.2 国内研究现状

国内对于人脸识别技术也有着广泛和深入的研究，尤其是在深度神经网络方面取得了一系列的突破和创新。目前，国内的人脸识别技术主要分为两大类：基于传统机器学习的方法和基于深度神经网络的方法。

基于传统机器学习的方法主要沿用了国外的一些经典算法，如 LBP、HOG、LPQ 等，并对其进行了一些改进和优化，以适应不同的应用场景和需求。例如，王晓峰[18]等提出了一种基于 LBP 和 SVM 的人脸识别方法，通过引入局部方向梯度信息和局部二值模式信息来增强人脸图像的特征描述能力，并结合 SVM 分类器来实现人脸识别。

基于深度神经网络的方法主要借鉴了国外的一些先进模型，如 VGG、ResNet、Inception 等，并对其进行了一些改造和创新，以提高人脸识别的性能和效率。例如，孙毅[19]等提出了一种基于 ResNet 和 Center Loss 的人脸识别方法，通过引入 Center Loss 损失函数来增强人脸特征之间的类内紧密性和类间分散性，并结合 ResNet 网络来提取人脸特征。

在新冠肺炎疫情期间，由于佩戴口罩成为了防控疫情的重要措施之一，给现有

的人脸识别系统带来了新的挑战。国内针对戴口罩人脸识别问题，有以下几种主要的研究方向：

基于多任务学习的方法。这类方法主要利用多任务学习技术，将人脸识别任务与其他相关任务，如口罩检测、口罩分类、人脸属性识别等，同时进行学习，从而提高戴口罩人脸识别系统的性能和鲁棒性。例如，李晓宇[20]等提出了一种基于多任务学习的戴口罩人脸识别方法，利用一个共享的卷积神经网络来提取人脸特征，并分别用三个不同的全连接层来实现口罩检测、口罩分类和人脸识别三个任务。

基于知识蒸馏的方法。这类方法主要利用知识蒸馏技术，将一个大型的预训练模型，如 VGG、ResNet 等，作为教师模型，将一个小型的轻量化模型，如 MobileNet、ShuffleNet 等，作为学生模型，并通过一定的方式，如软标签、硬标签、特征图等，来传递教师模型的知识给学生模型，从而提高戴口罩人脸识别系统的效率和准确率。例如，张宇[21]等提出了一种基于知识蒸馏的戴口罩人脸识别方法，利用一个预训练的 ResNet-50 作为教师模型，一个自定义的轻量化网络作为学生模型，并通过软标签和特征图来进行知识蒸馏。

1.3 研究内容

本文的主要研究方向是基于 YOLOv5 目标检测算法与质心追踪算法的戴口罩人脸检测识别系统。本文的研究内容如下：

首先，分析了新冠肺炎疫情对人脸识别系统的影响和挑战，以及国内外对于戴口罩人脸识别问题的研究现状和存在的问题，确定了本文的研究目标和意义。

其次，介绍了本文所采用的 YOLOv5 目标检测算法和质心追踪算法的原理和特点，以及它们在戴口罩人脸识别系统中的作用和优势。

再次，设计并实现了本文的戴口罩人脸检测识别系统的原型和功能模块以及前后端和数据库，详细说明了每个模块的功能和实现方法及细节。

最后，利用自行构建的戴口罩人脸数据集，对本文的戴口罩人脸识别系统进行了训练和测试，并与其他几种常见的人脸识别方法进行了对比和分析，验证了本文方法的有效性和优越性。

1.4 应用前景

目前，在后疫情时代下，戴口罩出行是当前最有效的防护方式。因此，在地铁、门禁等应用人脸识别的场合卸下口罩才能进行准确的识别无疑升高了病毒的传播速率。

1.4.1 门禁考勤

目前，人脸识别系统多应用于门禁考勤。对该系统及进行开发，通过数据库存储客户所录入的人脸识别信息。

公司企业选择使用人脸识别系统，不仅能有效的防范陌生人或者推销人员等随便进出公司办公室或者办公楼，而且能为员工打造一个安全与体验感良好的办公环境。而且对于公司企业来讲，人脸识别考勤系统所具有的活体识别检验等功能可以有效防止员工代打卡或者蒙混过关的行为。并且，具体的考勤记录也会自动地传至系统后台，然后能形成多样化的考勤报表，可供相关管理人员进行使用。这不仅能为公司领导提供实时与准确的公司员工考勤数据，还能提升公司企业的人事部门的工作效率。

社区的人脸识别门禁系统具有多种优势，能实现智能化的管控社区人员进出与增强社区的安全系数。在社区各个大门处或单元楼门口处安装使用人脸识别门禁，对于已完成人脸信息录入的业主或住户能直接通过“刷脸”轻松进出。而对于没有做人脸信息录入的陌生人或者推销人员、不明身份的人员等是无法通过人脸识别门禁系统。这无疑极大地增强了社区的安全系数与管理水平。当然，针对于社区业主的亲朋好友等访客可以由业主或住户验证身份后录入人脸，然后可以轻松刷脸通行。在访客结束行程后还可以删除权限。所以人脸识别门禁系统对于社区来讲，能实现有效的防止陌生人或不明身份人员等随意进出社区，让社区生活的安全体验感更高。

1.4.2 金融支付

在金融领域人脸识别技术同样大放异彩。支付宝的刷脸登录付款大家已经不会陌生了，通过刷脸可以进行账户登录并支付。多家银行升级了 ATM 机，其搭载的人脸识别技术全部采用最新的红外双目摄像头活体检测技术，能够完全抵御照片、

换脸视频、翻拍、面具攻击，除此之外，还具备静默活体、动作活体、唇语活体等活体检验方式，另外取款除了采用人脸识别功能，用户还需要输入手机号码或身份证号进行身份确认，最后再依靠密钥进行取款。

1.4.3 公共安防

在公共安防方面，在公交车、地铁站、飞机场和高铁火车站以及宾馆酒店等场景，人脸识别系统现在也是被广泛的运用。在这些场景的进出口出安装和使用人脸识别系统，利用系统的硬件摄像头设备等对进出人员进行人脸捕捉、抓拍识别验证。不仅在能在公共交通出行中提供“刷脸”通行服务，还能把抓拍的人脸信息与公安平台对接，能为公安部门提供相关的人员信息并协助好人员防控等事宜。

现在人们出行基本都戴口罩，这就降低了目前人脸识别系统在公共安防上的识别准确率。所以对于现有的人脸识别系统进行改进与升级是必要的。戴口罩人脸识别系统可以提高识别的准确率，进而帮助公安系统提高“天眼”的识别准确率

1.5 本章小结

本章通过分析新冠肺炎疫情下戴口罩人脸检测识别的重要性和意义，以及相关领域的应用需求，确定了本文的研究目标和意义。通过综述国内外关于人脸检测识别的研究现状，指出了目前存在的问题和挑战，为本文提出了创新点和优势。通过阐述本文的研究内容，即基于 YOLOv5 算法和质心追踪算法的戴口罩人脸检测识别系统的设计与实现，为本文后续的理论分析和实验验证奠定了基础。通过展望本文的应用前景，主要包括门禁考勤、金融支付和公共安防等方面，为本文提供了实际意义和价值。

第 2 章 基础理论知识介绍

随着深度学习技术的发展，人脸检测方法作为目标检测的经典应用，具有数据量、数据集丰富的优点。人脸数据处于非约束环境下的检测是一类足够复杂的任务，运用深度学习来解决此类任务依然具有极大的挑战。本章主要介绍了本文涉及到的基础理论知识，即卷积神经网络的基本概念、组成部分、激活函数和训练方法，以及卷积神经网络在人脸检测识别领域的应用。

2.1 卷积神经网络

作为深度学习中出类拔萃的算法之一，卷积神经网络[22](CNN)是传统多层感知神经网络（MLP）的一个改进。卷积神经网络非常适合处理高维输入，并且在图像分类方面一直表现出比 MLP 更好的性能。CNN 与 MLP 具有相似的拓扑结构，但基于 MLP 的结构对 CNN 进行了一些修改。CNN 中的卷积层由许多可识别的神经元阵列和卷积核滤波器组成，其中每个方形神经元内的实数相当于 MLP 中的连接。卷积层在前一层上执行“卷积”操作，其中内核过滤器可以被视为训练权重。基本的卷积神经网络结构如图 2.1 所示。

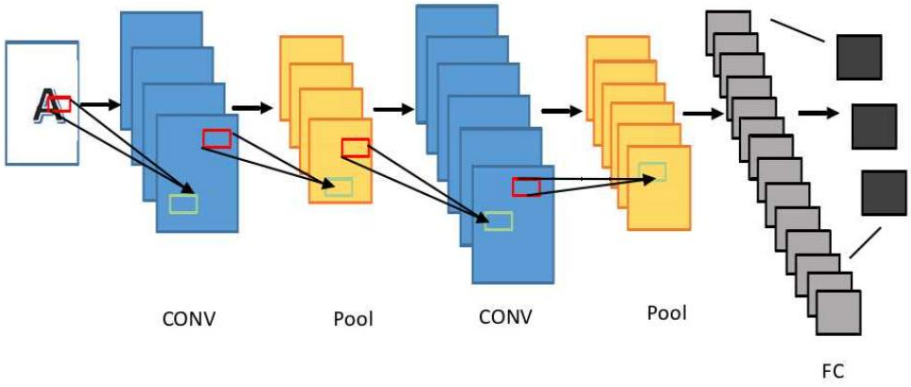

图 2.1 基本的卷积神经网络结构

多通道图像数据作为输入数据，传递到第一个卷积层后，卷积层输出的特征图通过激活函数处理后输出，该输出的结果通常应用于正确的分类预测。随后进入池化层，来减少参数量。卷积层在整个网络中的作用是提取特征，卷积层越深，提取的特征便具有更大的感受野[23]和更多的语义信息。越浅的网络，提取的特征则具有更多的位置信息。如前所述，CNN 中的输出层为多层感知机的全连接结构。来自

卷积层的输出被向量化后，输入多层感知机以进行预测。在全连接层的最后，根据任务不同，加入相应的结构以对齐到输出的格式。

2.1.1 卷积层

卷积层与多层感知机网络具有相似的拓扑结构，但基于多层感知机的结构对卷积层进行了一些修改。以一张 32×32×3 的图像为例，卷积层的输入就是一张 32×32×3 的矩阵，完全不用做任何改动性的操作。每层卷积的输出数据均称为特征图（feature-map），但浅层卷积结构所得到的特征图与深层特征图具有不同的含义。浅层特征图由于感受野较小，具有更多的位置信息；而深层特征图则具有高聚合性，弱化了位置信息的同时，具有更多的语义信息。在卷积运算中，需要引入几个新的概念：卷积核(kernel,常常简称为卷积，有时也称为滤波器)，根据实际需要自行定义卷积核的尺寸。步长，即卷积核在输入特征图上需要移动的像素。填充，即向输入特征图的周围填入固定的数据(比如 0)，使用填充可以调整输出特征图的大小。卷积运算通过将卷积核在输入数据或输入特征图上依次滑动预设步长的长度来进行。每滑动一次,卷积核就和所在的滑窗位置上所对应的输入数据（特征图）做一次对位相乘运算(将各个位置上卷积核的元素和输入特征图对应位置的元相乘)，相乘后将所得到的乘法结果进行相加作为该卷积核在该位置的特征输出。最终得到卷积运算的输出作为该层卷积完整的输出特征图。在 CNN 中,卷积核除了有权重参数之外,也存在偏置。偏置通常起到增加噪声作用使模型增加泛化性能，它会被加到应用卷积核的所有元素上。与传统神经网络不同的是，卷积层的计算是含有空间信息的。

2.1.2 池化层

在 CNN 中，可以在几个卷积层之后添加一个池化层，用于提取隐藏表示的特定特征，同时不会影响特征的提取。池化层的特点为，对当前的特征图数据进行降维处理而无需其他训练与学习操作，所以该过程没有要学习的参数。在池化运算中，针对当前通道，仅仅为当前矩阵的维度发生变化，而整体的特征图通道并不会发生变化。池化层仅压缩原始特征层的信息，该方法的优点在于，微小的偏差并不会对最终结果产生影响。因此该结构对模型鲁棒性的提升起到了关键作用。池化方法有

很多种类， 比较常用的是最大池化（Max pooling）和平均池化（Average Pooling）。例如，创建 m×m 最大池化窗口的维度，以提取对应最大池化窗口区域内像素的最大亮度值，以进一步增强来自先前卷积层的滤波图像的表征。除了最大池操作，平均池化方法也常用于窗口区域中的平均特征值。假定一个（4,4）的特征图，设置池化比例为（2,2），该特征图的池化操作如图 2.2 所示。

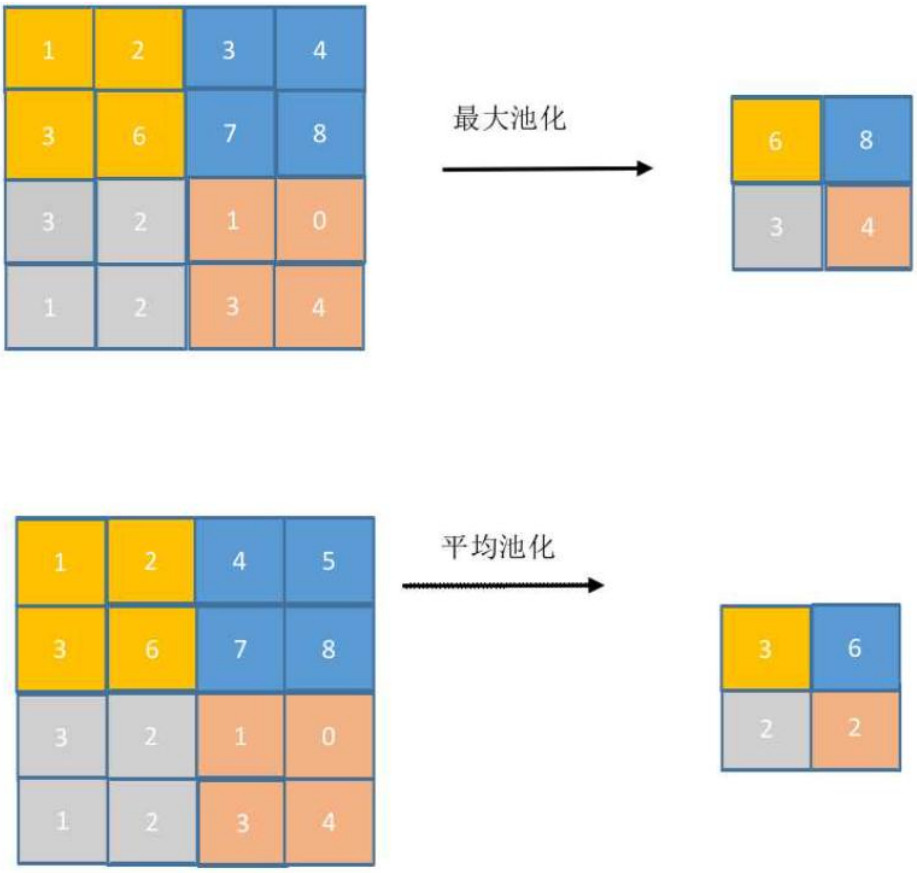

图 2.2 池化操作

2.1.3 全连接层

一般情况下，全连接层至少包含三层：输入层、一个隐藏层和输出层。全连接层的特点为：相邻两层之间的每一个神经元节点都与其上一层的所有神经元结点相连。该做法意在将上游结构提取到的特征进行个性化处理。

2.1.4 激活函数

卷积层和池化层的操作均属于线性变换，那么在处理实际生活中不可线性变换

的数据时，我们需要引入非线性因素。因此，引入非线性可导的激活函数，一方面方便计算，另一方面可以增加网络的深度为整体的梯度下降提供理论依据。激活函数在神经网络中是不可缺失的，选择适合的激活函数会提高网络对特征信息的表达能力，下面我们介绍三种比较常见的激活函数。

(1) Sigmoid 函数

Sigmoid 函数由于具有归一化的特性，所以经常用在二分分类问题或者多分类问题的最后一层。其公式表示如 2.1 所示：

$$S(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

它的优点在于，它是输出在(0,1)之间的单调连续函数，适合作为神经网络的输出层。它的缺点在于，由于该函数具有软饱和性，较为容易落入饱和区中。

(2) Tanh 函数

Tanh 函数只是在 Sigmoid 的基础上做了平移和拉伸，和 Sigmoid 对比而言，梯度值更大，由此可以加快训练速度，均属于饱和激活函数，其公式表示如 2.2 所示：

$$T(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{2.2}$$

Tanh 函数的输出范围为(-1,1)，并且经过原点，但幂运算的问题依然没有得到有效解决。

(3) ReLU 函数

ReLU 函数是近些年人们最常用的一个非线性函数，其其公式表示如 2.3 所示:

$$R(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases} \tag{2.3}$$

ReLU 函数的优点在于，争取解决了梯度消失的问题；由于该函数求导形式简单，为一个简单的阶跃函数，所以在神经网络运算过程中，该函数收敛速度远远快于上述两种函数。

2.1.5 卷积神经网络训练

当前神经网络结构之所以在众多领域上都取得了叹为惊人的成绩，在很大程度上取决的是网络训练学习过程的进步，深度神经网络的训练过程分为两个阶段：前向传播（feed-forward propagation）与反向传播[24](back-forwardpropagation)。整体训

练过程如图 2.6 所示。

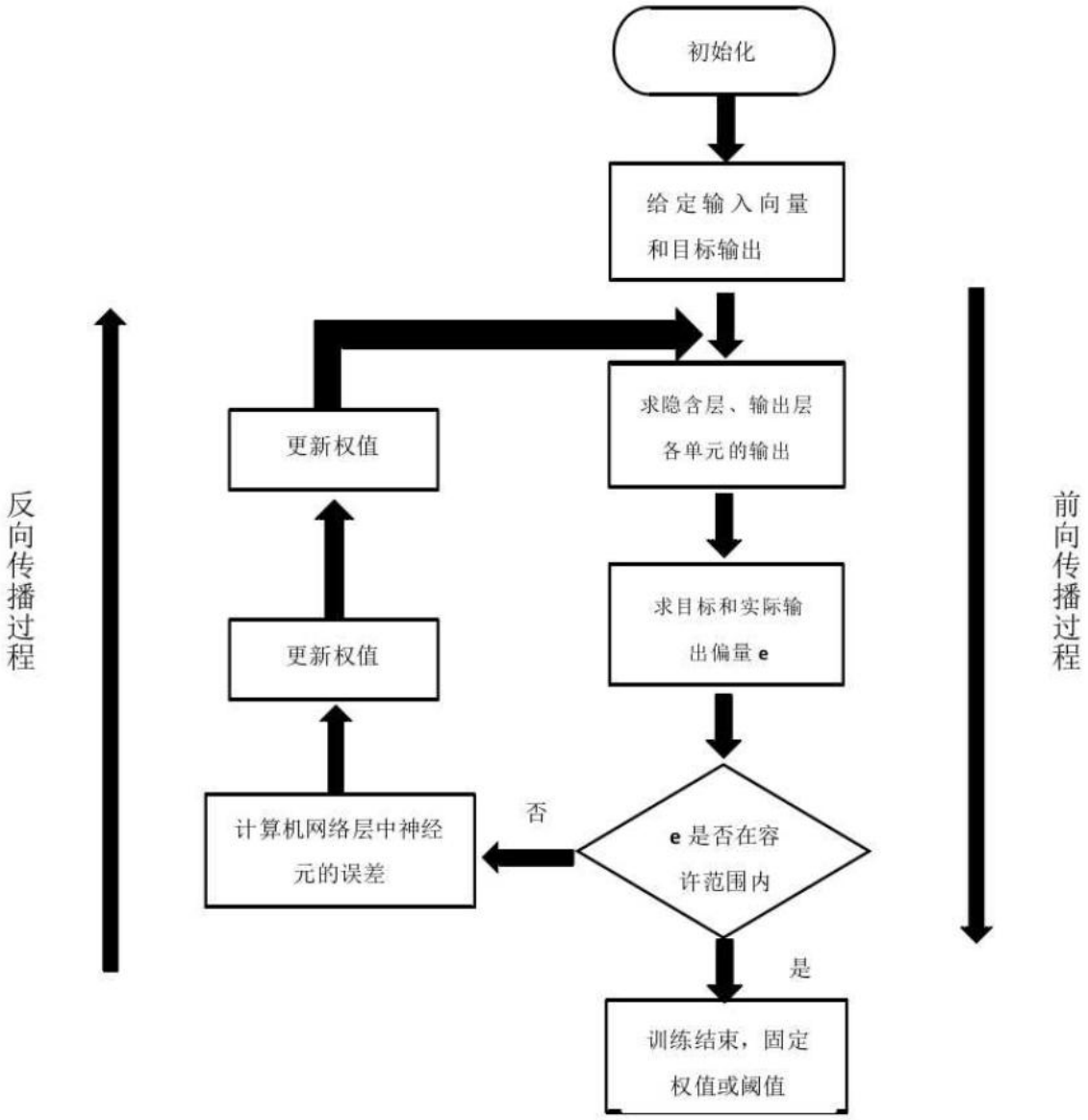

图 2.6 卷积神经网络的训练过程

(1) 前向传播阶段

在阶段中，网络中的每个节点加上一个偏差值作为输入，然后其对应的输出通过一个非线性函数计算激活函数，当前馈传播通过一个或多个隐藏层到达输出层时，获得预测目标$\hat{y}$ 以计算损失函数ℓ（$\hat{y}$，y），通常是期望输出 y 和预测输出$\hat{y}$ 之间的差值。如果我们使用 θ 来替换权重和偏差，损失函数可以重新表示为ℓ（θ）。然后，神经网络试图通过最小化损失ℓ（θ）来优化可训练参数 θ 。

(2) 反向传播阶段

反向传播常与最优化方法（如梯度下降法）联合运用,其全称为“误差反向传播”。

梯度下降法（GD）通常用于通过计算损失函数的偏导数来训练反向传播中的神经网络（θ）关于 θ 中每个元素的整个 N 个数据样本。然而，如果输入样本的总数非常大，这种方法需要很长时间来计算每次迭代中的梯度。而随机梯度下降（SGD）算法的缺陷在于训练不稳定。为了在计算效率和训练稳定性之间取得平衡，提出了一种小批量随机梯度下降（mini-batch-randomicgradient down，简称 mini-batch-SGD）算法，用于随机选择训练数据的小批量进行梯度计算。

2.1.6 具体应用

在人脸检测识别领域，卷积神经网络具有很强的判别能力，可以准确地区分人脸和背景，同时具有很高的效率和鲁棒性。卷积神经网络可以分为两个阶段：人脸检测和人脸识别。

人脸检测是指在图像中定位和提取人脸区域的过程，它是人脸识别的前提和基础。人脸检测的难点在于要处理不同的姿态、表情、光照、遮挡等因素，以及不同的背景和干扰物。为了解决这些问题，一些基于卷积神经网络的人脸检测算法被提出，例如 YOLOv5、MTCNN3、SSD4 等。这些算法通常采用多尺度、多阶段、多任务的策略，利用卷积神经网络的特征提取和分类能力，快速而准确地检测出图像中的人脸区域。

人脸识别是指在已知或未知的人脸库中识别出给定人脸图像的身份的过程，它是人脸检测的后续和目标。人脸识别的难点在于要处理不同的身份、年龄、性别、种族等因素，以及不同的姿态、表情、光照、遮挡等因素。为了解决这些问题，一些基于卷积神经网络的人脸识别算法被提出，例如 FaceNet、VGG-Face、ArcFace 等。这些算法通常采用端到端的方式，利用卷积神经网络的特征提取和度量学习能力，将人脸图像映射到一个低维度的特征空间，然后根据特征之间的距离或相似度来判断人脸的身份。

2.2 本章小结

本章通过介绍卷积神经网络的基本概念、组成部分、激活函数和训练方法，以及卷积神经网络在人脸检测识别领域的应用，为本文后续的 YOLOv5 算法的原理分

析和实现提供了理论支撑。

第 3 章 核心库和识别算法介绍

本章主要介绍了本文涉及到的核心库和识别算法，包括 OpenCV 库、Dlib 库和 FaceNet 算法。首先，介绍了 OpenCV 库和 Dlib 库的基本功能、特点以及它们在人脸检测识别领域的应用。最后，介绍了 FaceNet 算法的网络结构、损失函数和训练方法，以及 FaceNet 算法在人脸识别领域的应用。

3.1 OpenCV 库

OpenCV[25]是一个跨平台计算机视觉和机器学习软件库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上，轻量级而且高效，实现了图像处理和计算机视觉方面的很多通用算法。本系统开始设计时采用 OpenCV 进行人脸识别。OpenCV 有自己训练好的库，调用 OpenCV 的 haarcascade_frontalface_default.xml 文件，可以检测人脸。但是经过实际验证，识别效果较差，部分人脸无法识别。

在人脸是否佩戴口罩的检测中，采用 OpenCV 自训练软件来训练戴口罩的人脸检测模型来进行佩戴口罩的人脸检测。首先获取带口罩的图片以及未戴口罩的人脸图片共 3000 张，把戴口罩的照片看作正样本，未戴口罩的图片看作负样本，正负样本比例为 1:3。由于 OpenCV 不太准确，而且噪音样本影响非常大，因此将正负样本分别采用人脸识别器进行分类并获取人脸位置，将图片灰度化处理后进行剪裁。由于 OpenCV 的误识别，需要手工删除剪裁完后的图片中非人脸的图片。将修正完的正样本采用 OpenCV 自带的 opencv_createsamples.exe 进行训练，这是 OpenCV 自带的一个工具，封装了 haar [26]特征提取 LBP 和 HOG 特征分类器。但是新版本的 OpenCV 中找不到这个工具，需要在以往的版本中重新下载或者降低 OpenCV 版本。训练过程中出现了很多次中断，最终以 6 个小时左右的时长训练完毕。将训练完后的结果放入程序进行调试，发现结果比较差，很多非人脸区域也被识别了，因此放弃这种做法。

3.2 Dlib 库

Dlib[27]是一个机器学习的开源工具包，用于机器人、嵌入式设备、移动电话和

大性能计算环境。由于原生的 Python 不支持安装 Dlib 的 whl,因此需要先下载 Dlib 安装包，然后再安装。Dlib 基于 HOG，Histogram of Oriented Gradients 实现人脸检测，利用 Dlib 的 get_frontal_face_detector（正向人脸检测器）进行人脸检测，提取人脸外部矩形框，利用训练好的 shape_predictor 人脸 68 点特征检测器，进行人脸面部轮廓特征提取。Dlib 库识别人脸比 OpenCV 更为精确，但是在测试时发现不能检测到佩戴口罩的人脸。前期为了检测佩戴口罩的人脸区域，采用 OpenCV 来识别人脸区域，返回人脸坐标，并转化为 Dlib 类型的坐标。但是这样做就会导致无法识别人脸是否佩戴口罩，处理比较麻烦，而且精确度不高。所以后期改用 YOLO 算法进行目标检测。

3.3 FaceNet 算法

2015 年，谷歌公司提出了一种新的人脸检测算法，即 FaceNet。该算法在 YouTube Faces DB 数据集上，取得了 95.1%的计算精度,是目前该赛道上最好的检测记录。FaceNet 主要通过 CNN 的方法，将输入的人脸图像数据映射到欧氏空间中。映射后的空间距离直接代表图片之间的相似度，人脸图像的相似度越高，空间距离较小，人脸图像差异越大，空间距离则越大。那么就可以通过图像的空间映射所产生的欧氏距离信息情况，实现人脸识别。本文设计的人脸识别系统的人脸识别部分基于的就是 FaceNet 算法。

3.3.1 FaceNet 网络结构

FaceNet 的主干网络的作用是提取特征，其网络结构如 3.3 图所示。

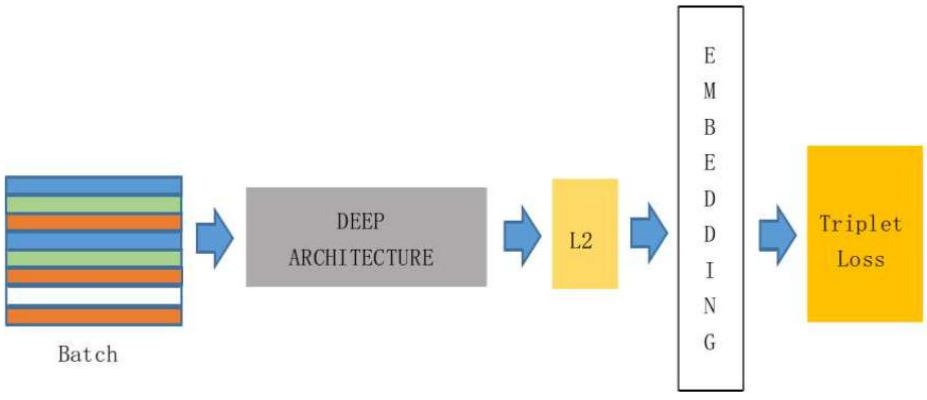

图 3.3 FaceNet 的网络结构

其中 Batch 表示批处理后的人脸图像训练数据，接着是卷积模块，然后采用 L2 正则化进行归一化操作。采用类似自然语言处理中的嵌入方法（Embedding learing），该算法能对原始网络输出层提取出的特征信息进行再进一步训练，从而使特征具有更好的语义信息。最后一部分结构为三元组损失函数，网络结构的末端使用 triplet loss 来进行分类任务，将单一个体的图像与其它人脸图像加以区分。

3.3.2 损失函数与训练

在 FaceNet 算法被提出之前，研究人员将基于深度学习的人脸识别算法抽象为分类任务，并使用交叉熵损失函数（Cross-entropy loss function）softmax 作为该任务的目标函数。交叉熵的主要原理在于计算样本标签与待分类样本预测标签之间的相似性，由此在人脸识别的过程中，可以看成一个待检测样本人脸与哪个已确定某个分类的人脸值更为接近，则这个待检测人脸即是该分类中的人脸。但是这个方式与人类大脑本身的识别方式有差异，例如人类大脑在区分双胞胎时，会对此类数据所产生的不同特征进行进一步区分。然而交叉熵损失却忽略了人脸特征的差异性，而着重人脸特征的相似性。

为了更好的弥补传统的交叉熵损失函数这一不足之处，研究者们在 FaceNet 模型中采用了三元损失函数（triplet loss function）的方式来提升性能。三元函数由 anchor(a),positive(p),negative(n)这三个部分组成，即（a,p,n）。为了保持绝对区分，该函数将 Xa 与 Xp 的距离比 Xa 与 Xn 的距离小一个间隔，类似 svm 中的几何间隔。triplet loss 函数的最终表达式如 3.1 所示：

$$L = \sum_i^n \left[\left\| f(x_i^a) - f\left(x_i^p\right) \right\|_2^2 - \left\| f\left(x_i^a\right) - f\left(x_i^n\right) \right\|_2^2 \right] + \alpha \qquad (3.1)$$

其中α为阈值，只有当类内距离大于类间距离时，才产生梯度和损失。该模型在每个 Mini-Batch 的训练时，需要选定合适的 triplet 三元组来计算 triplet loss 值。需要注意的是，如果采用暴力方法检测，可能出现查找时间过长、模型不收敛的情况。因此采用在线生成 triplet 的方式，该方法主要流程如下：

(1) 在初始化阶段，随机抽取图像。

(2) 将数据输入神经网络模型中并计算特征嵌入结果，通过计算得到三元组结

果。

(3) 根据三元组结果，通过目标函数计算损失，并更新嵌入的特征结果。

3.4 本章小结

本章介绍了本文涉及到的核心库和识别算法，包括 OpenCV 库、Dlib 库和 FaceNet 算法。通过介绍 OpenCV 库和 Dlib 库的基本功能、特点以及它们在人脸检测识别领域的应用，为本文后续的算法原理分析和实现提供了技术支持。通过介绍 FaceNet 算法的网络结构、损失函数和训练方法，以及 FaceNet 算法在人脸识别领域的应用，为本文后续的人脸识别工作流程设计和实现提供了算法支持。

第 4 章 YOLOv5 算法和质心追踪算法的介绍及应用

本章主要介绍了本文所采用的两种核心算法，即 YOLOv5 算法和质心追踪算法。首先，介绍了 YOLOv5 算法的原理及应用，包括输入端、Backbone、Neck 和 Head 四个部分，以及 YOLOv5 算法在戴口罩人脸检测领域的应用。其次，介绍了质心追踪算法的原理及应用，包括质心计算、距离度量、更新策略等，以及质心追踪算法本系统中的应用。最后，介绍了系统整体工作流程设计，包括人脸检测、人脸规范化、人脸特征提取和人脸匹配与识别四个步骤。

4.1 YOLOv5 算法

2020 年 6 月 25 日，Ultralytics 发布了 YOLOv5 的第一个正式版本。该框架采用了更轻量级[28]的网络结构，YOLOv5 的大小仅有 27MB，而使用 Darknet[29]架构的 YOLOv4 达到 244MB，对比之下该算法将模型大小压缩了近 90%，同时在准确度方面又与 YOLOv4 具有相近的竞争力。同时，该算法也是现今最先进的目标检测技术，并在推理速度上是目前最快的。YOLOv5 在 YOLOv4 的基础上进行了进一步的探索，在模块上和基础理论上都做了新的改进，使得其时间复杂度与模型精度上都取得了极大的提升。目前 YOLOv5 官方给出的目标检测模型一共有 3 个版本,本节以最基础的 YOLOv5s 为例进行说明，其他三种版本的结构原理与之类似，就是网络的深度和宽度有所不同。由于当前官方只发布了代码并没有相关的论文，所以只能通过官方 1.0 版本来了解 YOLOv5 算法。

4.1.1 输入端

(1) Mosaic 数据增强

YOLOv5 的输入端在模型训练阶段使用了 Mosaic 数据增强算法。该算法是在 YOLOv4 论文中第一次被提出来的，是一个很重要的数据增强技巧。Mosaic 在原理上参考的是 2019 年提出的 CutMix[30]数据增强方法，并在 CutMix 的基础之上上做了小小的改进。CutMix 采用的是两个样本，而 Mosaic 采用的是四个样本。一般而言，Mosaic 采用的方法是从数据集中随机选取四张图片样本，通过随机裁剪、

随机缩放、任意组合拼接等方式，形成一张新的图片样本。该方式的优点在于使检测集有了足够多的样本数据。更具意义的是，通过随机裁剪等方式，该方法将检测物体的背景和目标物体数据集进行扩容，提高了网络的稳定性。同时一次性计算四张图片样本的数据，使 Mini-batch 的大小无需增加的同时，通过单 GPU 便可达到预期的效果，但是不适用于数据集中本身就有很多小目标的情况，那样会导致小目标变得更小，降低了模型的泛化能力。Mosaic 与 Cutmix 数据增强操作的对比如图 4.1 所示。

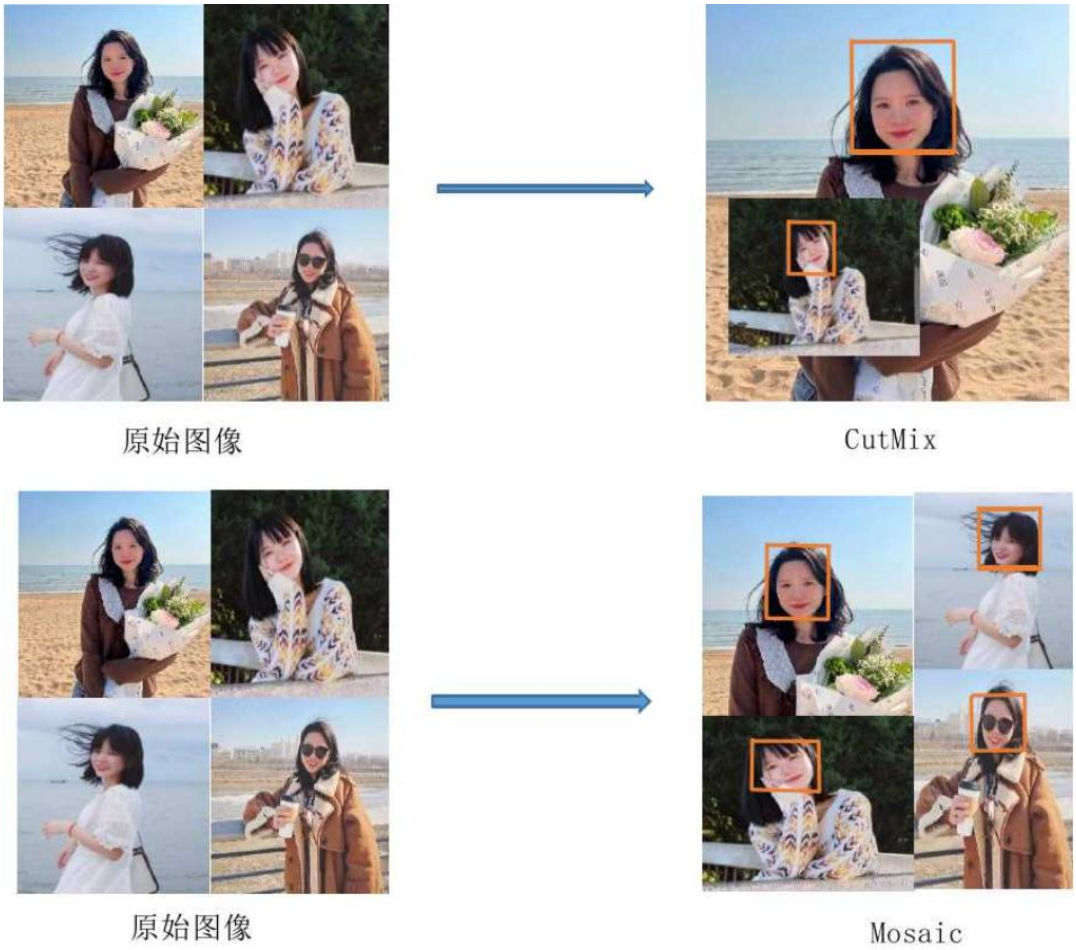

图 4.1 CutMix 与 Mosaic 数据增强对比

(2) 自适应锚框计算

YOLOv5 直接把锚框的计算嵌套到了训练的全过程之中，在模型训练过程中，输出预测框的方式不同于 YOLOv3 和 YOLOv4，它的新颖之处是在起初设定好的锚框之上进行，然后通过与 Ground-truth 比较，得出 loss 值，接着再进行更新，从而不断地更新锚点框的大小。通过这种自适应的形式，即使在不同的训练集中，

YOLOv5 也可以得出最佳的锚框值。如果在特定情况下，存在计算的锚框效果不理想的情况时，可以选择在代码中将计算锚框的功能关闭以提升最终精度。

(3) 自适应图片缩放

YOLOv5 之前的版本使用的图片长宽比通常是一比一的，例如 416×416, 608×608 等像素尺寸。而在实际检测项目中，存在图片长宽比例不同的情况。如果强行采用数据增强的方法将比例填充为 1：1，那么就会存在缩放填充后，两端的黑边大小不相同的情况。如果将图像高度上两端的黑边变少，那么冗余信息也会减少，目标检测速度也会在此基础上，进一步得到提升，具体操作步骤依次为计算收缩比，计算收缩后图片的长宽，计算需要填充的像素，最后 resize 图片并填充像素。

4.1.2 Backbone

Backbone 是模型的主干部分，它的作用是提取图片中的特征，供给后续网络用于定位目标位置和分类。YOLOv5 使用 CSPDarknet 结构作为 Backbone，其相对于 YOLOv4 算法来说，特点就是将 Focus 作为基准网络，加上 CSP 结构提高特征提取的能力。

(1) Focus 结构

Focus 模块在主体网络结构之前，是一种对特征图进行切片的方法。将宽度 w 和高度 h 的信息集中到通道 c 中，相对来说更适用于输入尺寸较大而通道数较少的图片。具体来说，就是在一张图像数据中，每隔一定的像素取一个值，类似于下采样操作。此种方法在一张图像数据中可获得四张图片，并且在保证信息无损的前提下，将通道扩充为原来的 4 倍，最后用新的图像数据进行卷积操作。例如，将原始的 640×640×3 的图像通过 Focus 结构变为 320×320×12 的特征图，再将变换过的数据经过一次卷积操作，输出的特征图尺寸变为 320×320×32。该方法有效地降低了 FLOPs（浮点运算数），提高了运行速度。切片操作如图 4.2 所示。

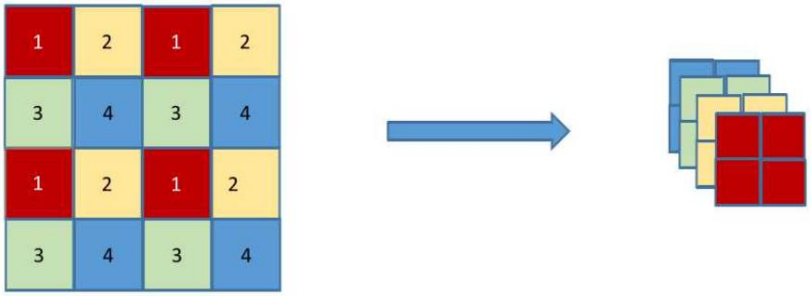

图 4.2 Focus 切片操作

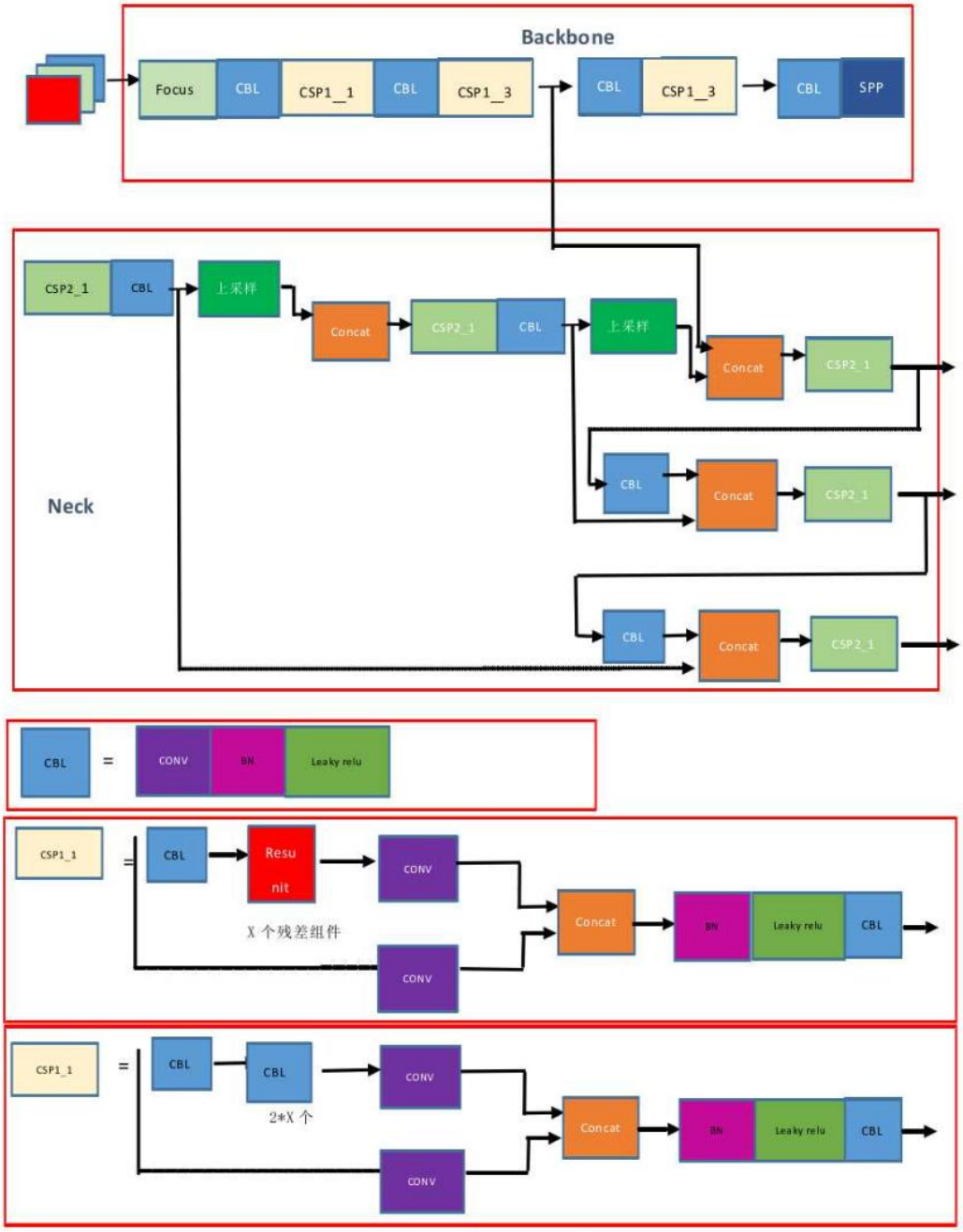

图 4.3 两种 CSP 结构

(2) CSP 结构

在 YOLOv5 网络中，参考了 CSPNet[31]的构想。该方法设计了两种 CSP 结构，其中 CSP1_X 应用在 Backbone 中，另一种 CSP2_X 应用在 Neck 结构中。针对两种不同的 CSP 结构，在 CSP1_X 前设置一个卷积层，使特征图的尺寸减半，该方法起到下采样的作用。两种 CSP 结构如图 4.3 所示。

将输入分为两个分支，一个分支先通过 CBS，再经过多个残差结构（Bottleneck * N），再进行一次卷积；另一个分支直接进行卷积；然后两个分支进行 concat，再经过 BN（正态分布），再来一次激活（之前的版本是 Leaky，后期是 SiLU），最后进行一个 CBS。

在深度学习网络框架的推理过程中，非常容易出现计算量过多的问题，大部分情况是因为网络中梯度信息大量重复，使用 CSP 结构可以解决这个问题，增强深层特征图的信息，较少计算量，提高了推理速度，保证了模型的准确性。

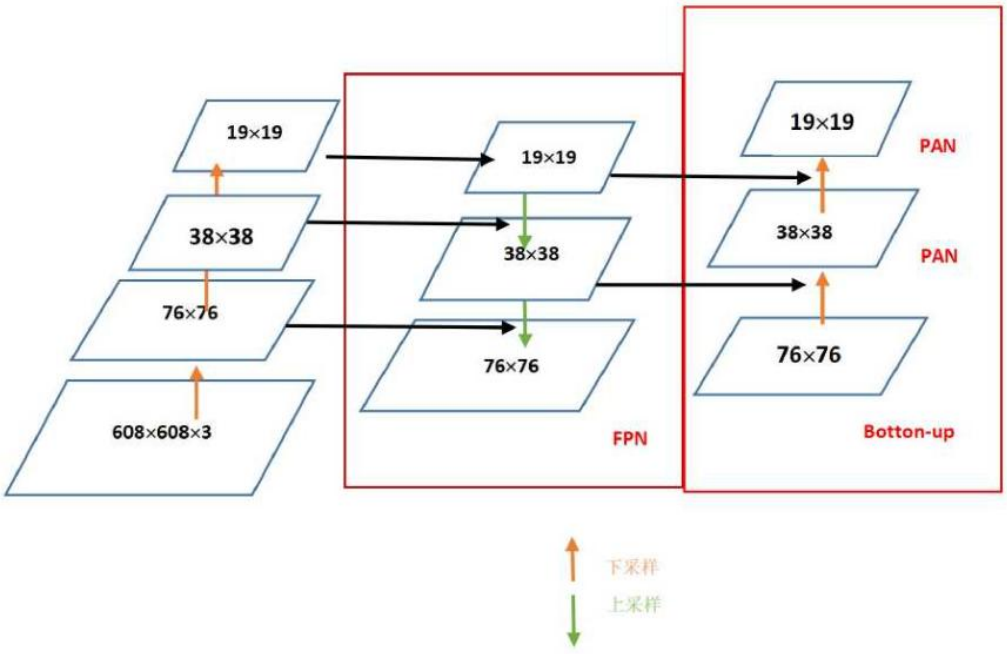

图 4.4 FPN+PAN 结构

4.1.3 Neck

在深度学习里，Neck 的位置是在 Backbone 和 Head 之间的，一般情况来说 Backbone 提取的特征图的长宽都较大，而定位输出结果和分类维度较小，由此就需要将特征的长宽进行压缩。Neck 可以将 Backbone 提取到的特征更好的去利用，

将特征进一步融合和处理后传给 Head 模块进行定位和分类。随着深度学习的不断发展，更多的 Neck 模块和更新的技术被创新出来，YOLOv5 的 Neck 模块采用的是 FPN+PAN 结构，结合下采样和上采样生成特征金字塔，产生网络具有不变形的效果。FPN+PAN 结构如图 4.4 所示。

FPN 所产生的特征金字塔将上采样结果与下一层特征进行融合，该方法针对小尺寸目标进行检测时，具有卓越的效果。而与 FPN 相反的是，PAN 产生的特征金字塔是是下而上的，它将下采样结果与上一次特征进行融合，PAN 可以对网络浅层的特征进行分割，因为目标检测任务所产生的输出为像素级，所以浅层信息具有很多边缘特征等位置信息，所以网络浅层信息在目标检测里的重要性非同一般。YOLOv5 中选择 FPN+PAN 结合的结构作为 Neck 模块，自上而下的方向上增加了自下而上方向上的加强，使顶层信息或得底层特征的位置信息，增强了大目标的检测效果。

4.1.4 Head

在 YOLOv5 算法里，Head 是最终检测和输出部分，将 Backbone 提取的特征，经过 Neck 的压缩和融合，进行分类做出预测。

(1) 损失函数

目标检测算法中，识别准确率主要与置信度、预测位置范围和预测类别有关联，而预测框的位置准确性要比框内目标的分类准确度更重要一些。YOLOv5 中预测框的损失函数 GIoU_Loss，如公式 4.1 所示。

$$GloU\_Loss = 1 - \frac{loU - |Ac - U|}{Ac} \tag{4.1}$$

GIoU_Loss 是在 2019 年基于 IoU_Loss 损失函数的基础上做了一些改进，解决了预测框与目标框都重叠且 IoU 相同，但是二者重叠位置不一样，IoU_Loss 对此情况无法进行区分的问题。其中 IoU 代表交并比，就是两个框相交的面积除以相并的面积。Ac 代表两个框最小闭包的区域面积值，U 代表两个框的并集面积值。

YOLOv5 算法中目标分类损失采用的是二分类交叉熵函数，具体公式如 4.2 所示：

$$Loss(o,t) = -\frac{1}{n}\sum_i^n (t[i] \times \log(o[i]) + (1 - t[i]) \times \log(1 - o[i])) \tag{4.2}$$

其中 t[i]为预测值，o[i]为真实值。

YOLOv5 算法中置信度缺失的目标函数采用的是 BECLogits Loss 损失函数。

(2) NMS 非极大值抑制

在目标检测的后处理过程中，由于检测模型会对输入图像输出较多的预测框，往往这些预测框里包含的内容很多都是相同的，只需要一个效果最佳的预测框即可，将检测结果简单化。同时，经过 NNS 抑制后，可以使原来未被检测出来的遮挡重叠目标被识别出来。

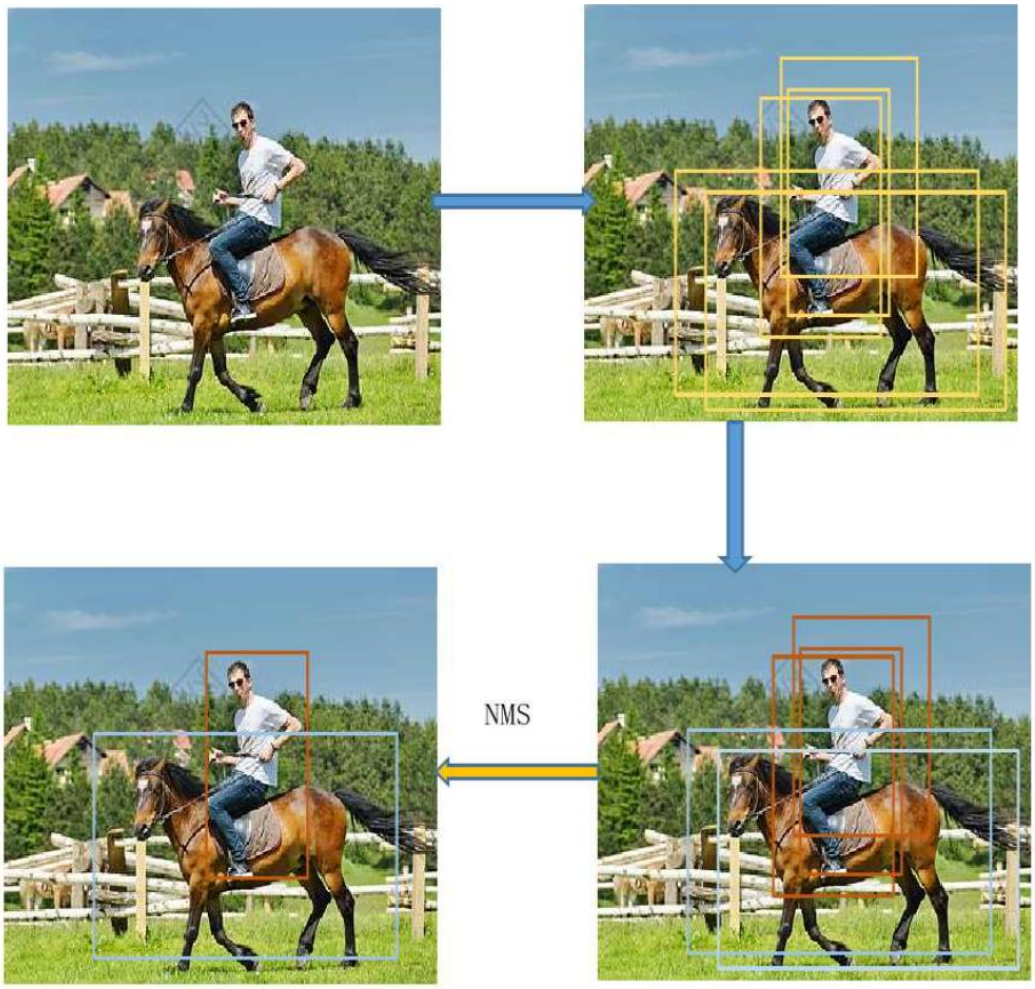

图 4.5 NMS 原理图

NMS 通过抑制非极大值元素来搜索局部极大值。具体步骤为：首先将所有得到的选择框进行排序，依据是迭代过程中的得分，将得分最高分的选择框提取；然后将其余的选择框一一和最高分选择框重叠，若重叠面积(IOU)大于初始设定阈值，则删掉该选择框；从未被处理的选择框中排序出另一个得分最高的选择框进行上述操作。NMS 原理图如 4.5 所示。

基于此，YOLOv5 算法比较显著的优点有：该方法采用 Pytorch 框架实现，该框架具有使用简单、用户基数大、易于上手扩展等优点，能够方便地针对特定任务进行训练与模型扩展；代码易读，该方法高度集成了先前计算机视觉领域的工作，有利于其他学者对该领域进行学习和研究；高效的模型训练过程。该方法预留大量可扩展接口，能够直接针对不同类型的输入数据进行有效运用和整合。同时，该框架支持将模型转化为其他平台，如安卓等所使用的 ONXX 格式。通过后续处理，也可以部署到移动设备端。YOLOv5s 高达 140FPS 的识别速度在一些实时场景中也存在极具意义的落地应用背景。

4.2 质心追踪算法

4.2.1 算法原理

由于对于当前帧进行人脸检测需要约 0.03s，对于检测出的人脸，提取特征描述子需要约 0.158s，对于当前帧中的所有人脸，都要和已知人脸数据库进行遍历对比，需要约 0.003s，整个过程需要约 0.19s，再加上设备性能的原因会造成很大的卡顿，基本看不到流畅性，帧率在 5 左右。为了解决这个问题采用质心追踪算法进行优化。

质心追踪算法的步骤:首先对于初始帧，通过检测算法，获得一系列目标的坐标位置；然后对目标创建 ID；在视频流中的后续帧，寻找帧之间目标对象的关系，将帧之间的目标关联起来。

目标追踪可以让我们对于每一个追踪的目标指定一个唯一的 ID,所以我们可以对视频中的跟踪物体进行计数，应用于计算人数的场景质心追踪算法,依赖于在视频流的连续帧中，比较已知目标与新出现目标之间质心的欧式距离。整体的处理逻辑流程如下，希望能够只在第一帧/初始帧进行检测识别，并试图将第 N+1 帧中的目标，与第 N 帧的目标关联起来，这样对于后续帧，不再需要进行识别，只需要进行检测就可以得到目标的 ID 了。

4.2.2 具体步骤

质心追踪算法的具体步骤如下：

(1) 对于某一帧找到特征框并计算质心，得到质心坐标（x,y）。如图 3.1 所示。

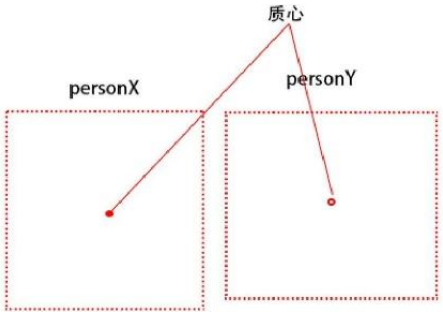

图 3.1 提取质心坐标

(2) 计算新旧目标特征框质心的欧式距离。对于视频流中的后续帧，我们利用检测算法来计算特征框，但是我们不会再去给每一个检测到的物体添加新的 ID 或者标记什么的（只做检测，不做识别），而是希望将新的目标能够和旧目标联系起来。如图 3.2 所示。

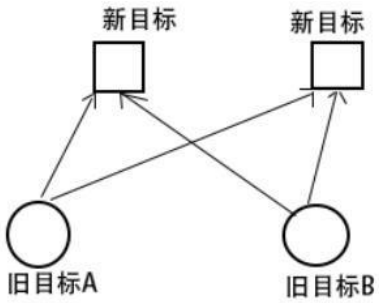

图 3.2 计算新旧目标之间的欧式距离

(3) 更新已知目标的质心坐标。质心追踪算法的前提是：对于一个给定目标，将会在连续几帧中都出现，而且在第 N 帧和 N+1 帧中的质心欧氏距离，要小于不同目标之间的欧式距离；因此我们在视频流的连续帧之间，根据欧氏距离最小原则，将这些帧中特征框的质心联系起来，可以得到一个目标 X 在这些连续帧中的变化联系，就达到了我们目标追踪的目的。

(4) 注册新目标。当新增一个目标时候，可以给这个新目标一个目标 ID,然后储存这个目标特征框的质心位置，从步骤 2 开始，对于视频流中的每一帧进行计算欧式距离，并更新坐标。

(5) 注销旧目标。一个目标在后续帧中可能会消失，对于消失的目标，注销旧目标。

4.3 系统整体工作流程设计

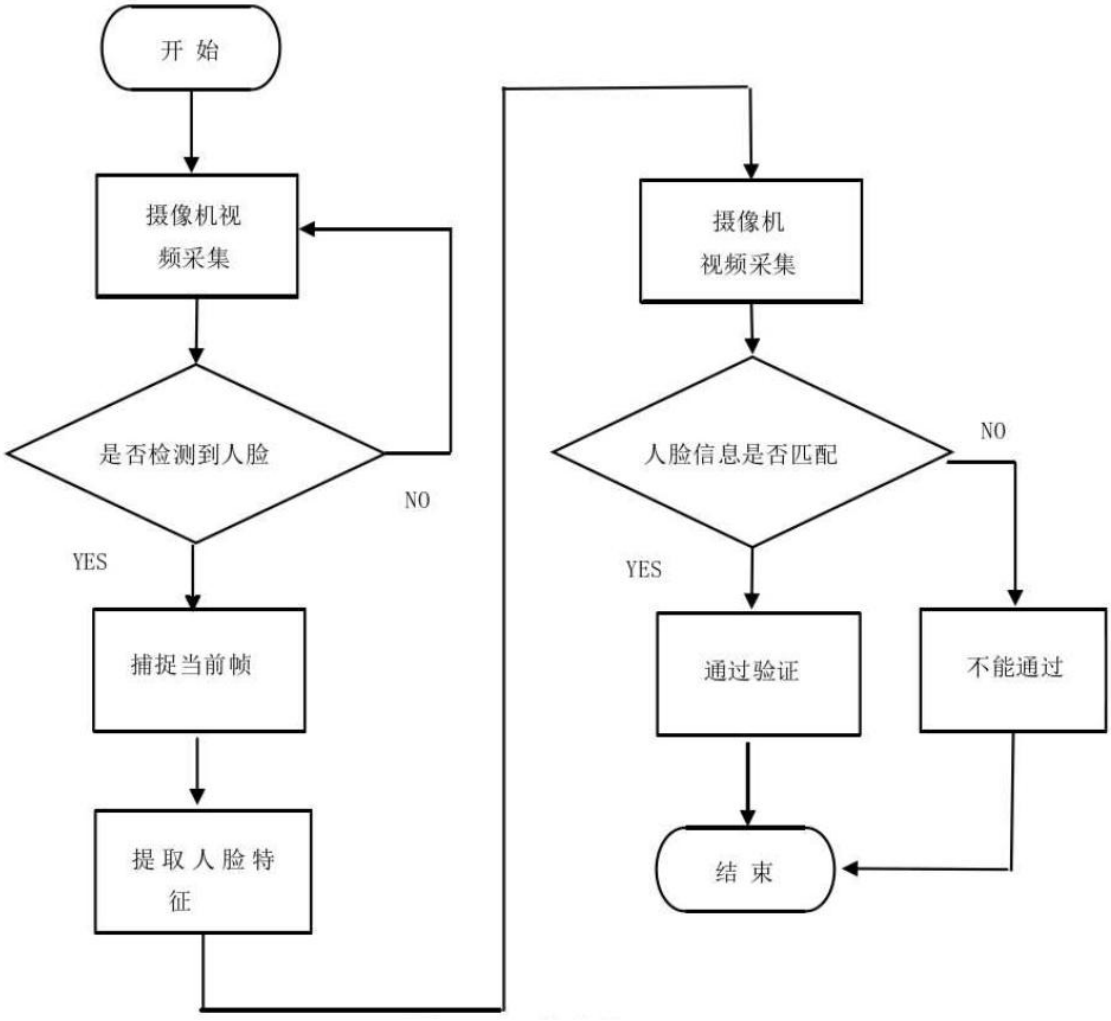

图 4.6 工作流程

一般情况下，人脸识别系统包含以下工作步骤：

(1) 人脸检测：对摄像头内的图片进行判断是否有人脸出现。

(2) 人脸规范化：对上一步骤中标定出的人脸图片进行灰度、噪声过滤、尺度、角度等图像预处理，确保规范化。

(3) 人脸图像特征提取：基于人脸的某些特征(比如像素特征、变换特征、视觉特征等)对已经进行检测和规范过的人脸图像，进行人脸特征提取,形成特征数据。

(4) 人脸图像匹配与识别:将上述步骤中提取到的人脸图像数据与数据库存储的已有数据进行搜索匹配，找到相似度最高的那个，判断是否为同一个人。本文设计的人脸识别系统包含人脸录入和人脸识别两个功能模块。在系统的主界面完成人脸

录入操作后，就可以进行人脸识别功能了，人脸识别模块中的人脸检测算法在上文中已经给出详尽的介绍，人脸识别算法我们用的是 FaceNet 算法，在本文的第 3 章里也有系统的介绍过。采用质心追踪算法来锁定视频中的图像，追踪首帧及后续帧中的人脸信息，然后提交给人脸识别模块。人部识别模块的算法流程如下：首先通过摄像头抽取一帧图像，利用人脸检测算法得到人脸特征信息与位置信息。这里需要注意的是，实际中出现的人脸可能是一张也可能是多张。接下来将该数据输入脸部识别模块，使用预测分类的方法与数据库中和其相似度最高的人脸图像进行匹配。具体工作流程如图 4.6 所示。

4.4 本章小结

本章介绍了本文所采用的两种核心算法，即 YOLOv5 算法和质心追踪算法。首先介绍了 YOLOv5 算法的原理及应用，包括输入端、Backbone、Neck 和 Head 四个部分。其次通过介绍质心追踪算法的原理及应用，包括质心计算、距离度量、更新策略等，以及质心追踪算法本系统中的应用，展示了本文所设计的系统在视频图像追踪上的稳定性。最后介绍了系统整体工作流程设计，将本章所提到的两种核心算法以及第三章所提到的核心库和识别算法在人脸检测识别系统中的具体应用进行了介绍，包括人脸检测、人脸规范化、人脸特征提取和人脸匹配与识别四个步骤。

第 5 章 系统设计

本章主要介绍了本文所设计的戴口罩人脸检测识别系统的系统设计，包括系统原型设计、功能模块设计和数据库设计，为后续实现系统提供架构基础。

5.1 系统原型设计

本系统原型设计采用 Axure 完成，后续根据系统实际情况进行更改。主界面计划设计六个按钮，点击按钮调用不同的子页面。最后一个按钮用于退出程序，点击按钮就可以退出当前程序。

人脸识别页面的原型设计思路是，当摄像头捕获到目标区域时，将目标区域展示在右侧方框中，并在下方显示识别的各种信息。按钮分别用来选择摄像头和打开摄像头。原型设计如图 5.1 所示。

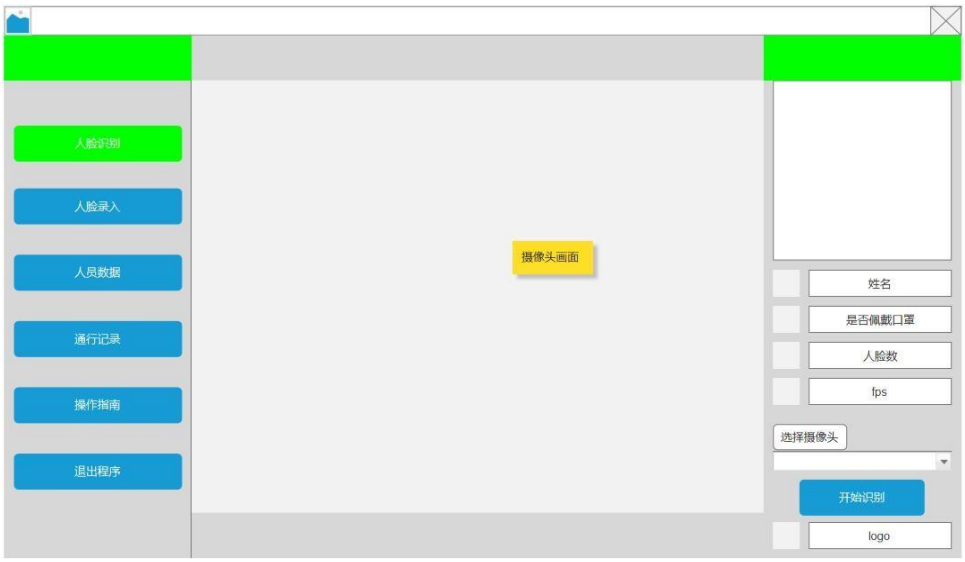

图 5.1 人脸识别界面原型设计

人脸录入的设计思路是，用户首先输入用户名和工号，然后选择摄像头，再打开摄像头，分别点击右下方的录入按钮，将捕获的目标区域显示在右上角的方框中，提取特征后存入数据库。原型设计如图 5.2 所示。

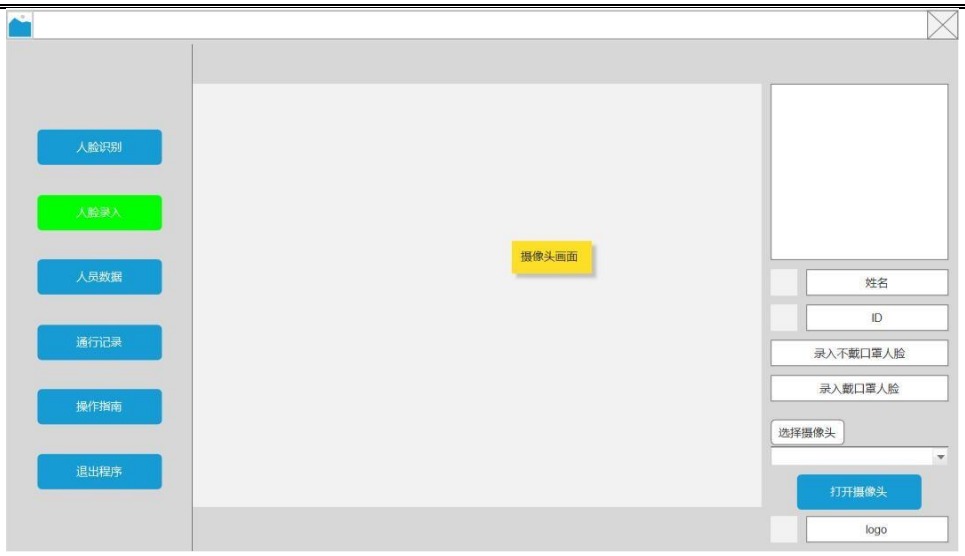

图 5.2 人脸录入界面原型设计

信息管理界面的设计思路是，点击搜索按钮，查询已录入的所有人脸图像信息并展示在列表区域。点击批量导入图片按钮导入按指定格式命名的图片，点击批量导入员工数据按钮，导入 excel 表格。点击新增按钮，新增单条记录。原型设计如图 5.3 所示。

图 5.3 信息管理界面原型设计

通信记录界面设计思路是，点击搜索按钮，展示所有已识别人脸信息的学号、姓名、通行时间和照片。支持条件查询，原型设计如图 5.4 所示。

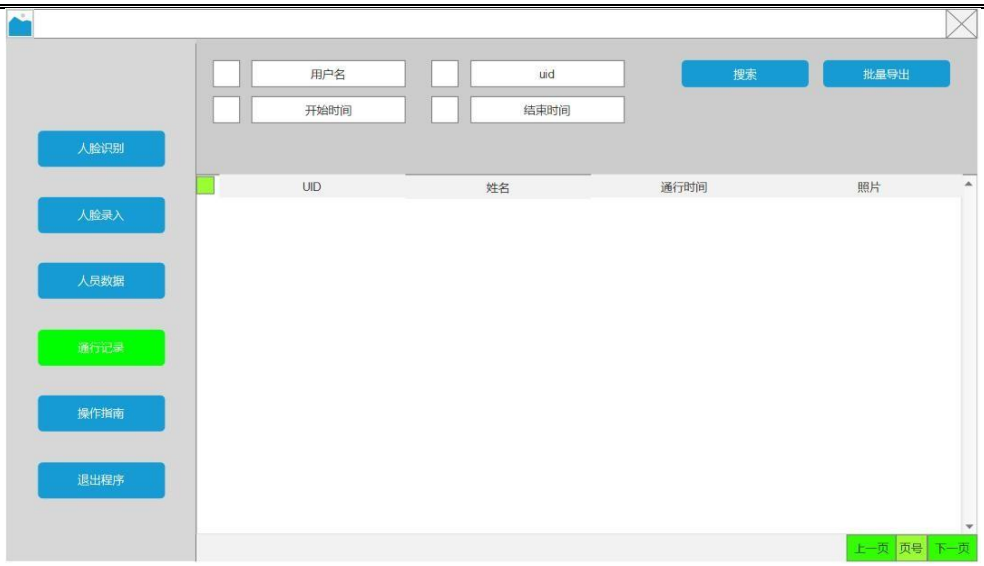

图 5.4 通行记录界面原型设计

5.2 功能模块设计

由需求分析可知，面向移动测温的人脸识别系统主要由以下模块构成：登录注册、人脸识别、人脸录入、信息管理和通行记录。其结构图如图 5.5 所示。

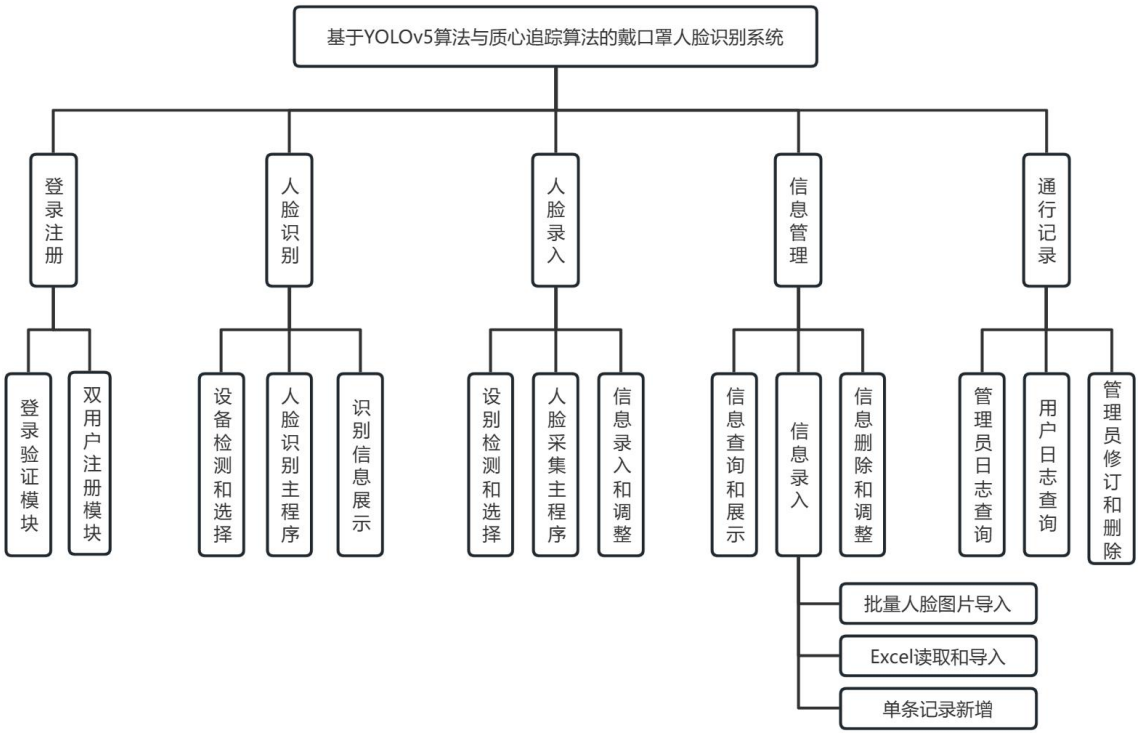

图 5.5 总体功能结构图

5.2.1 登录注册模块设计与流程

由需求分析可知，登录注册模块主要是为使用本系统的管理员服务。管理员可以使用已有的账号密码进行登录，登录成功后进入主页。输入用户名或密码错误，点击登录按钮后弹框提示“用户名/密码错误”。输入用户名或密码为空，点击登录按钮后弹框提示“用户名/密码为空，请重新输入”。输入用户名不合法，会提示“用户名输入不合法，请重新输入”。如果用户名/密码首尾存在空格，会去除空格后再进行判断。

如果用户没有账号，点击注册按钮进行注册。如果注册时输入的账号不合法，提示“请输入合法账号”。如果设置密码不符合规则，提示“密码必须包含字母、数字、特殊字符”。如果注册时输入的用户名或者密码或者确认密码为空，弹框提示“用户名/密码不能为空”。如果输入密码和确认密码不一致，弹框提示“两次输入密码不一致”。如果该账号已存在，弹框提示“该账号已存在，请重新注册”。注册成功弹框提示“恭喜您注册成功”，然后进入主界面。流程如图 5.6 所示

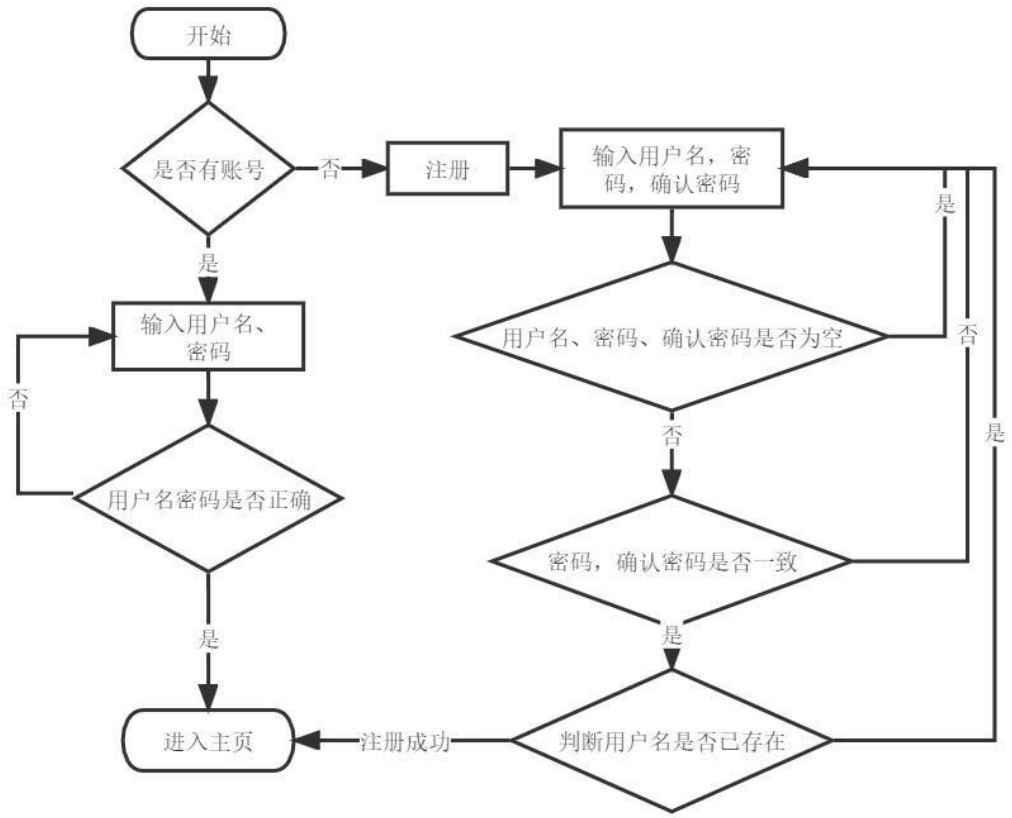

图 5.6 登录/注册流程图

5.2.2 人脸识别模块设计与流程

管理员登录成功后，首先选择摄像头编号，0 位内置摄像头，其他是外置摄像头。选择完后点击选择摄像头按钮，然后点击开始识别按钮，进行人脸识别。如果未选择摄像头，则开始识别按钮无法点击。该设置防止用户误操作。识别成功展示各种识别信息，如姓名、是否佩戴口罩、当前人脸数、帧率。姓名默认展示为 unknow，当识别的人脸在数据库中存在时，则更新姓名为识别结果。如果目标检测结果不佩戴口罩，展示为 xxx-nomask。如果目标检测结果为佩戴口罩，展示为 xxx-mask。识别结束后点击结束识别按钮停止人脸识别，此时开始识别按钮置为禁用。人脸识别流程如图 5.7 所示。

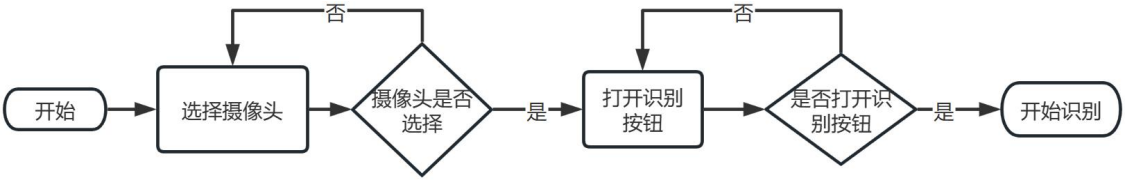

图 5.7 人脸识别流程图

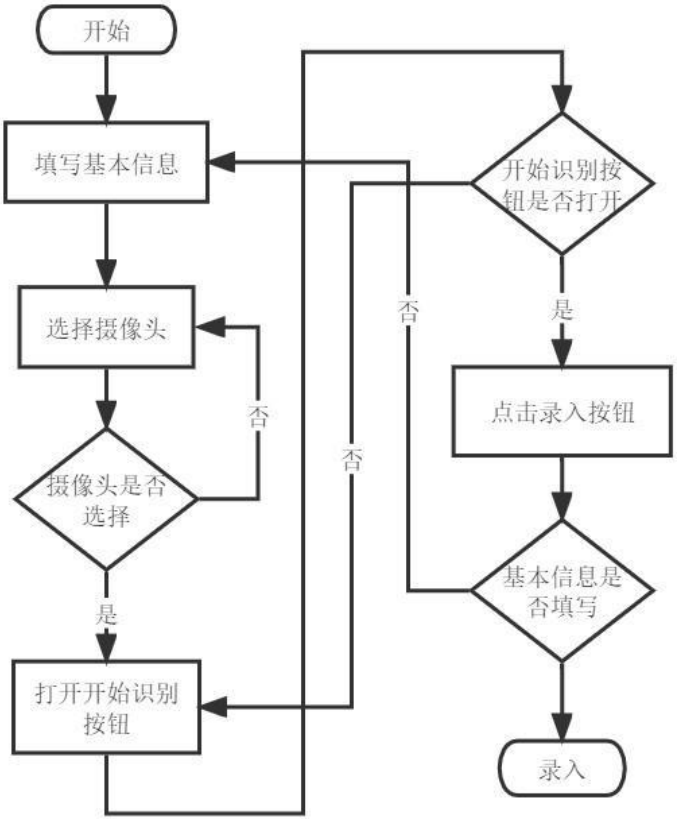

图 5.8 信息录入流程图

5.2.3 信息录入模块设计与流程

点击信息录入按钮进入信息录入页面，首先选择摄像头，根据 log 框提示信息点击开始录入按钮打开摄像头，分别点击录入佩戴口罩人脸图片按钮和录入不佩戴口罩人脸图片按钮，录入人脸信息，录入完毕关闭摄像头。操作按钮必须按顺序点击，否则按钮处于禁用状态。点击录入佩戴口罩人脸图片按钮或者录入不佩戴口罩人脸图片按钮时，如果未输入相关信息或者相关信息输入不合法，会弹窗提示重新输入。输入和操作都合法后，录入的人脸图像会显示在右上角的信息图片区域。后台处理人脸数据存入数据库。信息录入模块流程如图 5.8 所示。

5.2.4 人员管理模块设计与流程

该模块点击查询按钮展示所有已录入员工信息员工的员工号、姓名、佩戴口罩照片和不佩戴口罩照片。选择某一行，点击删除按钮，删除该条记录。点击批量导入图片按钮，选择存放按指定格式命名的文件夹，批量导入图片。点击批量导入 excel 表格按钮，选择存放员工号的姓名的 excel，导入数据。点击新增按钮，弹出新增窗口，按要求输入各种信息，点击确认，开始新增。

5.2.5 记录管理模块设计与流程

该模块点击查询按钮查询所有通行记录，展示识别结果、通行时间、截取的人脸图片。可以输入指定人员的姓名，然后点击查询按钮，根据姓名查询指定人员的通行记录。

5.3 数据库设计

面向移动测温的人脸识别系统只有两类用户，登录系统的用户和扫脸用户。因此不存在身份权限问题。管理员用户只需在使用该系统时进行登录，然后开始其他操作。扫脸用户需要提前将人脸图片和相关信息提交给管理员，管理员再将相关信息录入人脸数据库，扫脸用户在扫脸时候就可以识别人脸了，可以选择佩戴口罩或者不佩戴口罩识别。数据表主要包括管理员表、员工表、人脸信息表、图片路径表

和通行记录表，表名依次为：administrator，user，feature，pic，record。

(1) 管理员表，主要用于存储使用系统的管理员的登录账号和密码。如表5.1所示。

表 5.1 administrator 表

序号	字段名称	类型	长度	允许为空	描述
1	id	int	12	否	主键
2	username	varchar	20	否	用户名
3	password	varchar	20	否	用户密码

(2) 员工表，主要用于记录已经录入系统的相关人员的学号和姓名信息。如表5.2所示。

表5.2 user表

序号	字段名称	类型	长度	允许为空	描述
1	uid	varchar	12	否	主键
2	username	varchar	50	是	用户名

(3) 人脸信息表，主要用记录已经录入系统的人脸的 128D 人脸特征，包括佩戴口罩的人脸特征，不佩戴口罩的人脸特征，以及二者的镜像文件的特征。如表5.3所示。

表5.3 feature表

序号	字段名称	类型	长度	允许为空	描述
1	id	int	8	否	主键
2	uid	char	20	是	员工号
3	mirror	int	1	是	是否镜像文件
4	mask	int	1	是	是否佩戴口罩
5	featureX	mediumtext		是	人脸 128D 特征

(4) 图片路径表，用于存储已录入人脸图片的存储位置，用于后期展示图片或者删除。如表5.4所示。

表 5.4 pic 表

序号	字段名称	类型	长度	允许为空	描述
1	id	int	12	否	主键
2	uid	varchar	20	否	员工号
3	mask_path	varchar	50	否	配戴口罩图片路径
4	nomask_path	varchar	50	否	不配戴口罩图片路径

(5) 通行记录表，主要用于记录识别的人脸通行记录。如表5.5所示。

表 5.5 record 表

序号	字段名称	类型	长度	允许为空	描述
1	id	int	8	否	主键
2	username	varchar	12	是	用户名
3	time	datetime		是	通行时间
4	pic	varchar	50		通行照片路径

5.4 本章小结

本章介绍了本文所设计的戴口罩人脸检测识别系统的系统设计，包括系统原型设计、功能模块设计和数据库设计。通过介绍系统原型设计，包括系统的整体架构、用户界面和交互方式，展示了本文所设计的戴口罩人脸检测识别系统的系统概貌和用户体验。通过介绍功能模块设计，包括登录注册模块、人脸识别模块、信息录入模块、人员管理模块和记录管理模块，以及各个模块的设计目标、功能描述和流程图，展示了本文所设计的戴口罩人脸检测识别系统的系统功能和业务逻辑。通过介绍数据库设计，包括数据库的逻辑结构、物理结构和数据表的设计，展示了本文所设计的戴口罩人脸检测识别系统的数据存储和管理方式。

第 6 章 系统实现

本章主要对上一章节所涉及的系统设计进行实现，包括前端和数据库实现、功能模块实现。

6.1 前端和数据库实现

(1) 前端实现

本系统采用 PySide2 进行前端界面设计和美化。PySide2 是一个 Python 模块，提供对 Qt5 框架的访问。与 PyQt 相比，PySide2 是开源的，使用不受限制，与 Qt 的 C++相比，PySide2 可以大大减少代码量。基于以上有点，PySide2 适合团队规模不大，项目需要快速推进，需要开发精美界面的场合。同时采用 Qt Designer 工具进行界面设计，生成 ui 界面，后续只需加载 ui 文件，在需要的地方添加界面代码，真正做到界面与代码分离，降低耦合性。

(2) 数据库实现

本系统使用 MySQL 数据库。使用 pymysql 进行连接，开始尝试过用 QMYSQL 进行连接，但是存在很多问题，比如版本问题和不兼容问题，试了很多博主推荐的方法都没有解决，最终采用了最原始的方法。

6.2 功能模块实现

6.2.1 登录注册模块实现

登录和注册模块隐藏 PySide2 原有的边框，和退出、最大化、最小化按钮，设置窗口大小为 800*450，且打开后窗口固定在屏幕中央不可拖动。在主窗口自定义最小化和退出按钮。点击注册按钮跳转的注册页面，注册页面添加返回功能，点击返回按钮回到登录界面。

设置窗口样式关键代码实现如下：

```
self.ui.resize(800, 450)
# 设置窗口不可拖动
self.ui.setFixedSize(self.ui.width(), self.ui.height())
```

```
# 设置窗口只显示关闭按钮
self.ui.setWindowFlags(Qt.WindowCloseButtonHint)
self.ui.setWindowFlags(Qt.FramelessWindowHint)
```

设置按钮功能代码如下：

```
self.ui.close_btn.clicked.connect(QApplication.quit)
self.ui.min_btn.clicked.connect(self.min_window)
self.ui.back_btn.clicked.connect(self.back_login)

def min_window(self):
    self.ui.showMinimized()
# 注册页面返回功能
def back_login(self):
    self.ui.hide() SI.loginWin.ui.show()
```

登陆界面实现用户名校验，输入合法用户名、密码直接回车可以登录，点击登录按钮也可以登录。点击右上角搜索按钮，程序页面最小化，点击退出按钮退出程序。登陆界面程序运行效果如图 6.1 所示。

图 6.1 登录页面

注册页面实现账号、密码校验，输入合法用户名、合理密码、确认密码后点击注册按钮注册成功后进入主界面，不成功则重新注册。点击右上角收缩按钮最小化页面，点击退出按钮退出程序。点击左上角返回按钮，返回到登录界面。界面大小

固定且居中，不可拖动。注册界面程序运行效果如图 6.2 所示。

图 6.2 注册页面

4.2.2 人脸识别实现

主界面采用 QMidArea 多区域设计。人脸识别是其中一个页面，点击人脸识别按钮，进入人脸识别页面，点击选择摄像头按钮，选择可用摄像头按钮选择摄像头，然后点击开始识别按钮，如果未选择可用摄像头，则开始识别按钮禁用。在人脸识别这一板块，涉及人脸区域目标检测和算法优化。人脸区域目标检测采用 YOLOv5 算法，返回坐标区域和识别结果。关键代码如下：

```
for *xyxy, conf, cls in reversed(det):
xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4))).view(-1).tolist()
xywh = [round(x) for x in xywh]
xywh = [xywh[0] - xywh[2] // 2, xywh[1] - xywh[3] // 2, xywh[2], xywh[3]]
cls = names[int(cls)]
conf = float(conf)
detections.append({'class': cls, 'conf': conf, 'position': xywh})
```

在人脸识别优化上，采用目标检测算法进行优化，优化完后的帧率可以达到 15 以上，视频流畅度可以满足，主要代码如下：

```
# 如果当前帧和上一帧人脸数未发生变化
```

```
if (self.current_face_cnt == self.last_face_cnt) and (self.reclassify_cnt != self.reclassify_interval):
    ……
    else:
    if self.current_face_cnt == 0:
        self.current_frame_face_name_list = [] self.current_feature = []
else:
```

人脸识别的过程，根据检测出的人脸区域，标定人脸 68 个特征点，然后提取 128D 人来特征，遍历人脸特征库，返回二者之间的欧氏距离，如果欧式距离小于一定的阈值，就可以认为识别的人脸是同一张人脸，本系统采取的阈值是 0.5，在训练样本最够大的情况下，阈值可以继续调整。

```
# 检测人脸 可以检测多个人脸 返回检测结果和人脸坐标(x,y,w,h)
detections = fa.detect(img_rd)
print("人脸数：", len(detections)) faces = []
mask_or_not = []
for i in detections:
    pos = i['position']
    # 将坐标转化为 Dlib 类型（左上，右下）
face = Dlib.rectangle(pos[0], pos[1], pos[0] + pos[2], pos[1] + pos[3])
faces.append(face)
mask_or_not.append(i['class'])
```

人脸识别界面点击主程序界面的人脸识别按钮，进入该界面。界面中间区域用于展示摄像头画面，右上角小区域用于展示人脸捕获图像，下面展示人脸识别信息。程序运行效果如图 6.3 所示。

图 6.3 人脸识别主界面

当不佩戴口罩的人脸区域出现时，摄像头区域框出人脸范围，并在左上角标注识别姓名和 no-mask 的标志。右侧展示当前人脸和姓名、是否佩戴口罩、人脸数、帧率等信息。程序运行效果如图 6.4 所示。

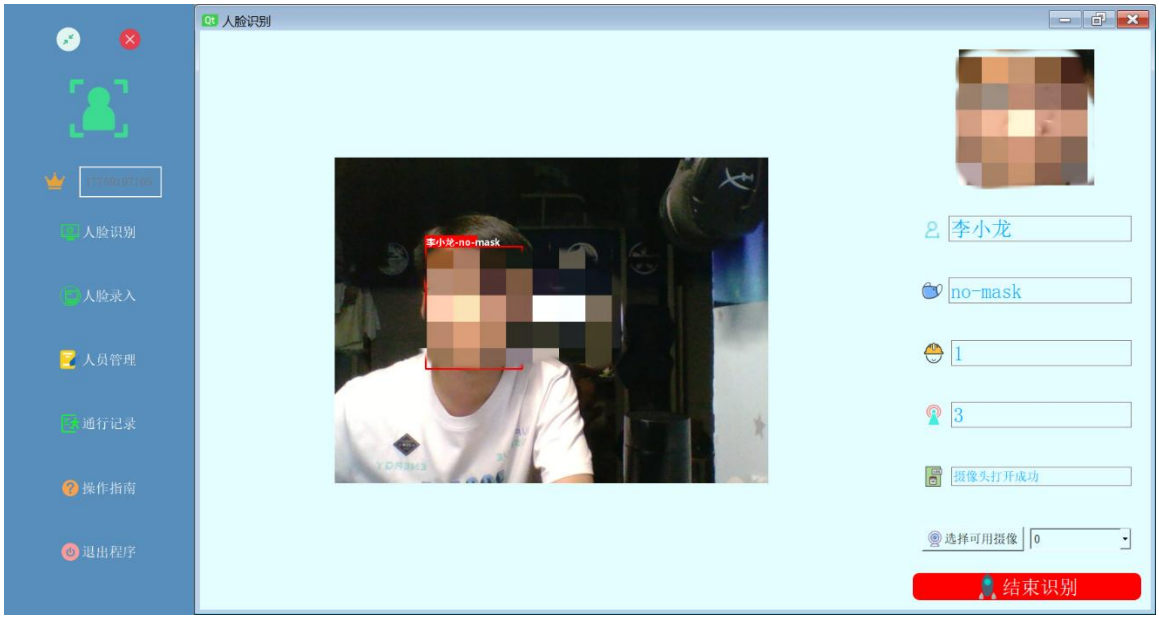

图 6.4 不佩戴口罩人脸识别结果

当佩戴口罩的人脸区域出现时，摄像头区域框出人脸范围，区域右上角展示姓名和 mask，右侧展示各种信息，包括姓名、mask、人脸数、帧率。程序运行效果如图 6.5 所示。

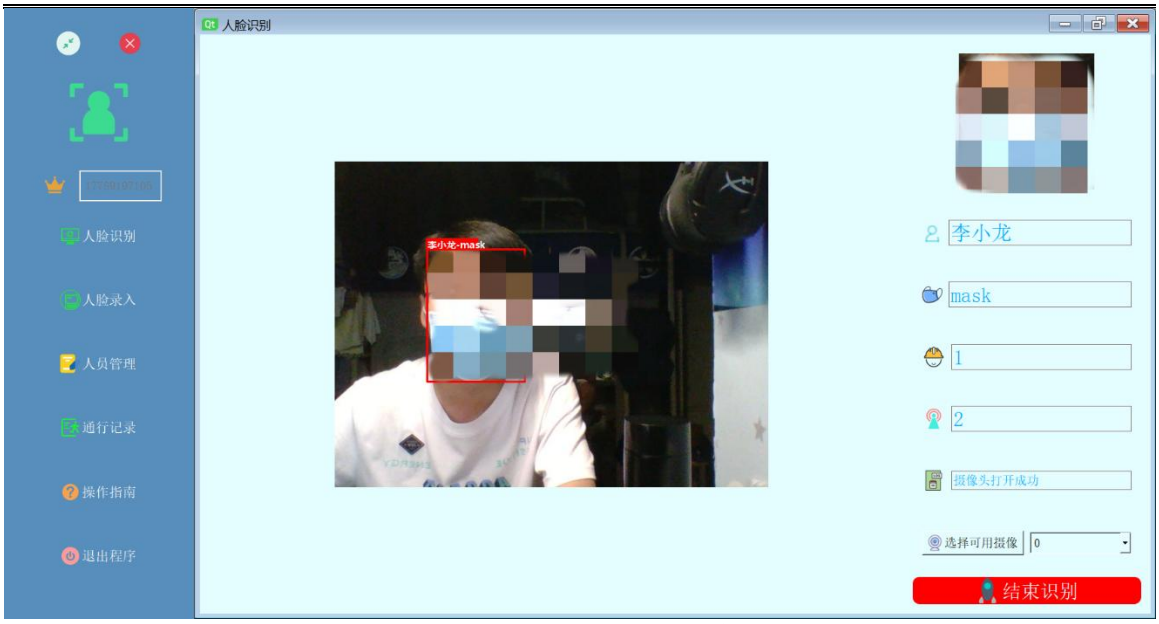

图 6.5 佩戴口罩人脸识别结果

4.2.3 信息录入实现

人脸信息录入模块与识别模块相似，都是利用 YOLOv5 目标检测算法进行人脸区域标定，据检测出的人脸区域，标定人脸 68 个特征点，然后提取 128 维人脸特征，同时将截取的图片镜像处理，一共得到同一个人佩戴口罩和不佩戴口罩以及镜像文件四种特征值，存入数据库。关键代码如下：

```
# 不管用户图片是否已经录入都可以重新插入，人脸识别要想准确就要多多益善
img = self.image
img_mirror = cv2.flip(img, 1) # 镜像文件
# 识别人脸区域，并 class,cfs,position 分类，概率，位置
detections = fa.detect(img)
# 将人脸位置转化为列表
face = [detections[0]['position'][0], detections[0]['position'][1], detections[0]['position'][0]
+ detections[0]['position'][2],
```

人脸录入界面中间区域用于展示当前摄像头捕获的画面，右侧输入用户 ID 和用户名，当点击采集佩戴口罩照片或者采集不配戴口罩照片的按钮时，采集到的人脸图片展示到右上角的小区域中。Log 区域用展示各种操作信息。程序运行效果如图 6.6 所示。

图 6.6 人脸录入主界面

4.2.4 人员管理实现

人员管理包括增、删、查三个二级模块，增加模块又包括批量导入图片，批量导入 excel 表格，新增一个人的信息。导入图片需要现将图片按照指定格式命名后，放在一个文件夹里面，然后选择这个文件夹导入图片。关键代码如下：

```
filePath = QFileDialog.getExistingDirectory(self.ui, "选择图片路径")
if not filePath:
    return
# 设置项目根目录
SI.projectPath = filePath fileList = []
for root, dirs, files in os.walk(filePath):
    fileList = files
```

新增单条记录是创建了一个新的 ui 界面，然后设置输入用户名，工号编辑框，点击按钮选择照片，点击保存开始导入，点击取消退出新增。

删除功能，选择所要删除的行，然后点击删除按钮就可以删除一行信息。关键代码如下：

```
# 删除某一行的数据
row_select = self.ui.all_user_table.selectedItems()
```

查询功能，可以根据用户名和工号进行查询，如果什么都不输入点击查询按钮，默认展示所有的录入数据。程序运行效果如图 6.7 所示。

图 6.7 信息展示

图 6.8 单条数据新增

单条数据新增的页面，输入用户名和学号，分别点击不佩戴口罩照片和佩戴口罩照片选择图片录入，点击保存按钮录入成功，点击取消按钮退出当前页面。学号做了长度和纯数字校验，防止用户将学号和姓名的位置输入错误。程序运行效果如图 6.8 所示。

6.2.5 通行记录管理实现

通行记录模块会展示所有的通行记录信息，通行记录不支持删除，因为需要根据记录找到对应的人。可以根据用户名查询指定姓名的人的通行记录。该模块也可以充当公司考勤功能，应为会记录识别时间。页面效果如图 6.9 所示。

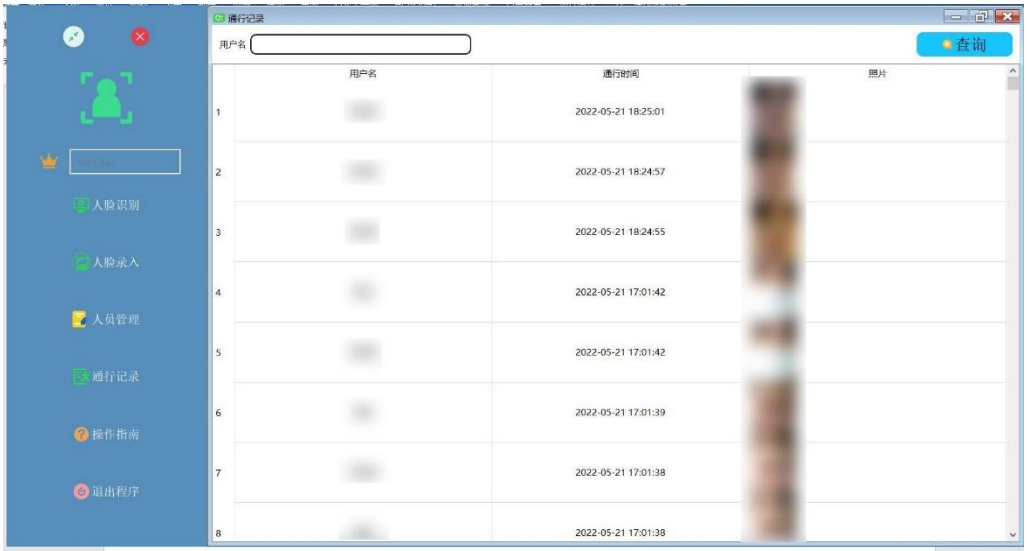

图 6.9 通行记录页面

6.3 本章小结

本章主要介绍了系统实现，包括前端和数据库实现以及功能模块实现。首先，介绍了前端和数据库实现，包括开发环境、开发工具和开发语言。其次，介绍了功能模块实现，包括登录注册模块、人脸识别模块、信息录入模块、人员管理模块和记录管理模块，以及各个模块的实现细节和代码片段。

第 7 章 实验结果与分析

本系统基于 Windows10 操作系统构建 Python3.7 编程环境实现，选用 Pytorch1.8.0 深度学习框架搭建网络模型。在 PyCharm 中导入 Dlib 库、OpenCV 库等库依赖，实现 YOLOv5 算法和质心追踪算法，使用 MySQL 数据库进行数据的存储。实验数据设置 2000 张人脸照片进行训练，用 labelimg 标注工具进行人脸佩戴口罩和不配戴口罩区域标记。标记完会生成 txt 文件，将 txt 文件转化为 YOLO 算法可以识别的 xml 文件。然后将生成的结果按 8:2 划分为训练集和测试集。修改 YOLOv5 源代码中关键部分进行训练，训练时长大约 12 小时。初始学习率设为 0.01；momentum 设为 0.937；权重衰减系数(weight decay)设为 0.0005。本章主要介绍实验数据集，实验环境和参数，实验评价指标以及实验结果。

7.1 实验数据集

首先选取佩戴口罩的人脸照片和不佩戴口罩的人脸照片共 2000 张，采用 labelimg 标注工具进行人脸佩戴口罩和不配戴口罩区域标记。标记完会生成 txt 文件，将 txt 文件转化为 YOLO 算法可以识别的 xml 文件。然后将生成的结果按 8:2 划分为训练集和测试集。修改 YOLOv5 源代码中关键部分进行训练，训练时长大约 12 小时。训练完后用 YOLOv5 原生代码进行测试，可以看到测试结果不错。最后将测试代码封装成一个 detect 函数，函数返回识别结果、识别概率和人脸区域坐标位置。在人脸识别算法中调用封装好的代码替换 Dlib 原有的人脸目标区域检测算法。detect 函数返回人脸坐标为(x,y,w,h)，与 OpenCV 返回结果类型相同，而 Dlib 识别人脸坐标为(x1,y1,x2,y2)，因此需要将 detect 返回结果进行处理。将处理后的结果传入 Dlib 人脸 landmark 特征点检测器 shape_predictor 进行人脸 68 个特征点标记，标记完后用 face_recognition_model_v1 提取人脸 128 维的特征矢量，进行人脸对比。

7.2 实验环境与参数

本实验使用 PyTorch 框架，在 CentOS 平台上完成，编程语言使用的是 python 3.7,

显卡为 NVIDIA GTX1660ti，内存为 16G,训练时使用 512G 固态硬盘缓存数据。训练网络用于检测人脸，在 YOLO 网络中类别名称表中只保留 face 一个标签，并改动配置文件 nc：1。其中网络的深度和宽度在代码中的设置如图 7.1 所示。

```
# parameters
nc: 1  # number of classes
depth_multiple: 0.67  # model depth multiple
width_multiple: 0.75  # layer channel multiple
```

图 7.1 网络的深度和宽度

其中 depth_multiple 代表网络的深度，用于控制模块的数量，模块的数量非 1 时，模块的数量为 number×depth；width_multiple 代表网络的宽度，用来控制卷积核的数量，卷积核数量为数量 width。本实验中锚定框在已预设图像大小为 640×640 下的尺寸，且锚定框在大特征图上检测小目标，在小特征图上检测大目标。每个特征图有三种尺寸的锚定框具体在代码中设置如图 7.2 所示。

```
# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

图 7.2 锚定框尺寸

初始化超参数如图 7.3 所示。

```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
```

```
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
```

图 7.3 初始化超参数

7.3 实验评价指标

基础解释：

混淆矩阵(Confusion Matrix)：混淆矩阵是表示精度评价的一种标准格式，用 n 行 n 列的矩阵形式来表示。每一列代表了预测类别，每一列的总数表示预测为该类别的数据的数目；每一行代表了数据的真实归属类别，每一行的数据总数表示该类别的数据实例的数目，如表 7.1 所示。

表 7.1 混淆矩阵

	预测类别			
实际类别		YES	NO	SUM
	YES	TP	FN	P(实际为 YES)
	NO	FP	TN	N(实际为 NO)
	SUM	P(被分为 YES)	N(被分为 NO)	P+N

True Positive(真正，TP)：将正类预测为正类数。

True Negative(真负，TN)：将负类预测为负类数。

False Positive(假正，FP)：将负类预测为正类数误报 (Type I error)。

False Negative(假负，FN)：将正类预测为负类数→漏报 (Type II error)。

精确率：正确预测为正占全部预测为正的比例，如式(7.1)所示。

$$Precision = \frac{TP}{TP+FP} \tag{7.1}$$

精确率代表对正样本结果中的预测准确程度，准确率则代表整体的预测准确程度，包括正样本和负样本。分母是预测到的正类，精确率的提出是让模型的现有预测结果尽可能不出错。

召回率：即正确预测为正的占全部实际为正的比例，如式(7.2)所示。

$$Recall = \frac{TP}{TP+FN} \tag{7.2}$$

召回率(Recall) 是针对原样本而言的，其含义是在实际为正的样本中被预测为正样本的概率。高召回率说明可能存在更多的误检，但仍然会努力找寻每一个应该被找到的目标。

为了能够说明不同算法间的优缺，在精确率和召回率的基础上提出了 F1 值的概念，来对精确率和召回率进行整体评价。F1 的定义如式(7.3)所示：

$$F1 = \frac{2Precision*Recall}{Precision+Recall} \tag{7.3}$$

PR 曲线：以召回率作为横坐标轴，精确率作为纵坐标轴。mAP 就是 PR 曲线与 X 轴围成的图形面积，mAP 值为 1 时模型性能最好。

7.4 实验结果

本系统混淆矩阵如图 7.4 所示。

选取精确率、召回率、平均精度均值(mAP)、F1 值作为评价指标来验证本系统的检测性能，验证本系统的可应用性，结果如表 7.2 所示。

表 7.2 各评估指标具体数据

精确率	召回率	平均精度均值	F1 值
0.929	0.900	0.796	0. 82

精确率、召回率、平均精度均值、F1 值相关评估指标曲线如图 7.5 所示。

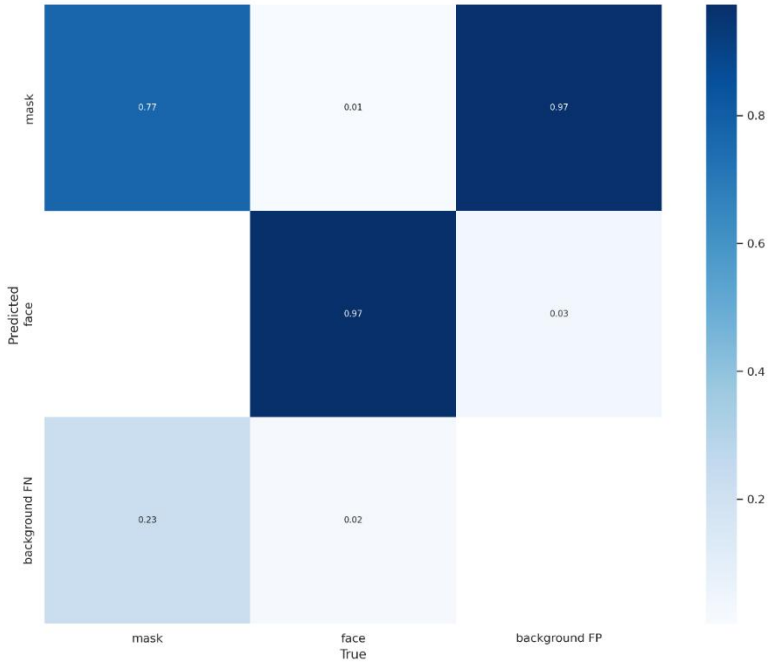

图 7.4 两千张图片训练后系统混淆矩阵

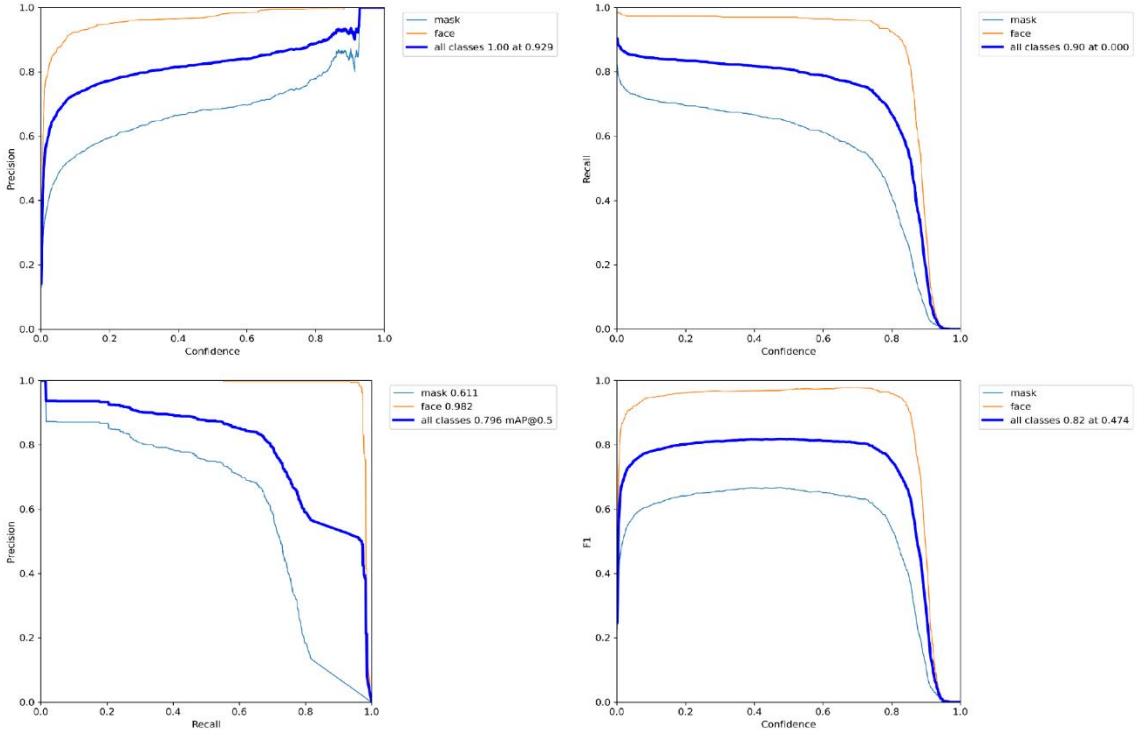

图 7.5 各个评估指标曲线图

训练结果曲线图如图 7.6 所示，可以看出损失或误差在下降，精度在提升。

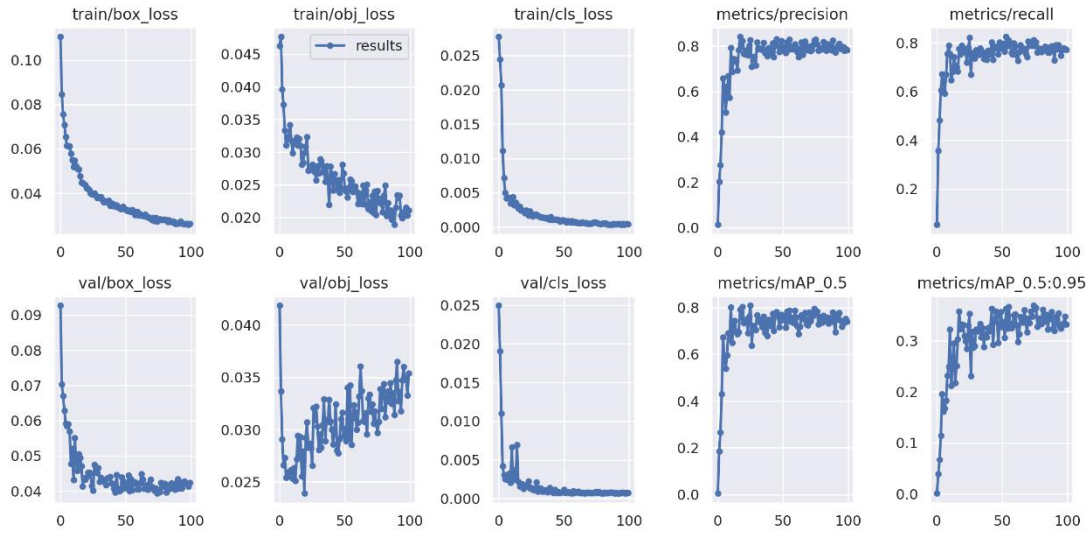

图 7.6 训练结果曲线图

经过训练后，测试一千张不同人脸，与数据库进行比对，得出如下数据见表 7.3（未识别或识别错误的人脸统一使用 unknown 进行标记，识别成功计数器开始计数，设置时间计数器并提交至列表中存储）训练结果曲线图。受限于训练样本和设备性能的影响，如果设备性能足够且训练样本足够大，准确度会大幅提升。

表 7.3 最终识别效果

人脸 图片数目	识别 准确度	平均每张 检索时间
1000	87.5%	0.194
2000	89.8%	0.172

7.5 本章小结

本章通过介绍实验结果，包括实验数据集、实验环境与参数、实验评价指标和实验结果分析，展示了本文所设计的戴口罩人脸检测识别系统的性能评估和良好效果。

第 8 章 总结与展望

8.1 总结

本文针对新冠肺炎疫情期间，佩戴口罩对人脸识别系统造成的困难和挑战，提出了一种基于 YOLOv5 目标检测算法与质心追踪算法的戴口罩人脸识别系统。本文的主要工作和创新点如下：

采用了 YOLOv5 目标检测算法替换了 Dlib 原有的基于 HOG 检测的人脸目标区域检测算法，提高了人脸识别的灵活性和精准性。YOLOv5 目标检测算法是一种基于深度神经网络的端到端的目标检测算法，它能够同时输出目标的类别和位置，具有速度快、准确率高、泛化能力强等优点。

利用了质心追踪算法处理目标特征，实现了对戴口罩人脸的追踪和识别。质心追踪算法是一种基于质心距离的简单而有效的目标追踪算法，它能够根据目标的质心位置和运动方向，来判断目标的身份和状态，具有计算量小、实时性好、鲁棒性强等优点。

在前端界面设计方面，使用了 PySide2 模块和 Qt Designer 工具，实现了界面与代码分离，降低了耦合性，提高了开发效率。PySide2 模块是一个基于 Qt 框架的 Python 绑定库，它能够方便地调用 Qt 框架提供的各种图形界面组件和功能；Qt Designer 工具是一个可视化的界面设计工具，它能够快速地创建和编辑图形界面，并生成相应的 UI 文件。

经过 12 小时的训练，本系统的识别准确率可达到 89.8%，并且随着训练时间的增加，识别准确率还有进一步提升的空间。本文的戴口罩人脸识别系统具有较高的实用价值和应用前景，可以为疫情防控、公共安全、智能交互等领域提供有效的技术支持。

8.2 展望

(1) 本文提出的人脸检测算法是基于理论性研究和简单的应用，虽然能够满足日常的使用，但是还有需要改进和加强的部分，可以后续增加活体检测模块，这样可以使系统拥有更高的安全性和稳定性，基于此可以推广到更多生活领域中。

(2) 本文提出的人脸检测算法在训练时数据集规模受限，更丰富的数据对于训练的提升才直观有效，怎样在小规模数据中保持算法的识别精度，仍然是一个需要研究的问题。

参考文献

[1] 周飞燕, 金林鹏, 董军. 卷积神经网络研究综述[J]. 计算机学报, 2017, 40(6): 1229-1251.

[2] 陈伟. 目标检测回归损失函数——IOU、GIOU、DIOU、CIOU、EIOU[Z]. 知乎专栏, 2021-10-13.

[3] 聂凡杰. 基于端到端的深度学习目标检测算法研究[D]. 华中科技大学, 2018.

[4] 宋克臣, 颜云辉, 陈文辉, 张旭. 局部二值模式方法研究与展望[J]. 自动化学报, 2013, 39(7): 730-745.

[5] 李娜, 王晓东. 基于方向梯度直方图的人脸识别方法[J]. 计算机应用研究, 2014, 31(1): 1-4.

[6] 王晓峰, 李晓光. 基于局部相位量化的人脸识别算法[J]. 计算机工程与应用, 2013, 49(9): 153-156.

[7] 王晓峰, 李晓光. 基于支持向量机的人脸识别算法综述[J]. 计算机工程与应用, 2012, 48(9): 1-5.

[8] 张宏伟, 郭建华. 基于 K 近邻的图像检索算法综述[J]. 计算机应用与软件, 2015, 32(11): 1-6.

[9] 李军平, 王春生. 随机森林算法综述[J]. 计算机应用与软件, 2014, 31(10): 65-69.

[10] 张志强, 赵建军. 循环神经网络及其在自然语言处理中的应用综述[J]. 中文信息学报, 2018, 32(2): 3-14.

[11] 王鹏飞, 赵建军. 注意力机制在自然语言处理中的应用综述[J]. 中文信息学报, 2019, 33(6): 1-12.

[12] 李娜娜. 基于交叉熵损失函数的深度学习模型优化方法研究[D]. 南京理工大学, 2018.

[13] 王鹏飞. 基于三元组损失函数的人脸识别方法研究[D]. 南京理工大学, 2019.

[14] 张宇. 基于余弦相似度损失函数的人脸识别方法研究[D]. 南京理工大学, 2020.

[15] S. Wang, Y. Yang, Z. Chen, et al. RGB-Infrared Cross-Modality Face Recognition via Joint Pixel and Feature Alignment[J]. IEEE Transactions on Image Processing, 2019, 28(5): 1993-2007.

[16] A. Majumder, T. K. Hazra and D. P. Dogra. Face Mask Detection and Face

Recognition for Covid-19 Pandemic Control[C]. 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), Bangalore, India, 2020: 1-6.

[17] M. H. Alkhatib and A. A. Alshennawy. Face Recognition with Masked Faces Using Sparse Representation[C]. 2019 International Conference on Innovative Trends in Computer Engineering (ITCE), Aswan, Egypt, 2019: 1-6.

[18] 王晓峰, 李晓宇, 张宇等. 基于 LBP 和 SVM 的人脸识别方法[J]. 计算机应用与软件, 2018, 35(11): 1-4.

[19] 孙毅, 王晓军, 李斌等. 基于 ResNet 和 Center Loss 的人脸识别方法[J]. 计算机工程与应用, 2017, 53(10): 89-93.

[20] 李晓宇, 张宇, 王晓峰等. 基于多任务学习的戴口罩人脸识别方法[J]. 计算机工程与设计, 2020, 41(12): 3558-3564.

[21] 张宇, 李晓宇, 王晓峰等. 基于知识蒸馏的戴口罩人脸识别方法[J]. 计算机工程与应用, 2020, 56(23): 1-7.

[22] Dong C,Loy C C,He K,et al.Image super-resolution using deep convolutionalnetworks[J].IEEE transactions on pattern analysis and machine intelligence,2015,38(2):295-307.

[23] 赵巍,黄晶晶,田斌.基于感受野模型的图像融合算法研究[J].电子学报,2008,36(9):1665-1669.

[24] 刘曙光,郑崇勋,刘明远.前馈神经网络中的反向传播算法及其改进:进展与展望[J].计算机科学,1996,23(1):76-79.

[25] Bochkovskiy A,Wang C Y,Liao H Y M.Yolov4:Optimal speed and accuracy ofobject detection[J].arX iv preprint arX iv:2004.10934,2020.

[26] Krizhevsky A,Sutskever I,Hinton G E.Imagenet classification with deepconvolutional neural networks[J].Advances in neural information processingsystems,2012,25.

[27] Everingham M,Eslami S M,Van Gool L,et al.The pascal visual object classeschallenge:A retrospective[J].International journal of computer vision,2015,111(1):98-136.

[28] 葛道辉,李洪升,张亮,等.轻量级神经网络架构综述[J].Journal of Software,2020,31(9).

[29] 刘瑶.改进 Darknet 框架的多目标检测与识别方法研究[D].西安:西安工程大

学,2019.

[30] Yun S, Han D, Oh S J, et al. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 6023-6032.

[31] Wang C Y,Liao H Y M,Wu Y H,et al.CSPNet:A new backbone that canenhance learning capability of CNN[C]//Proceedings of the IEEE/CVFconference on computer vision and pattern recognition workshops.2020:390-391.

附录

核心算法代码如下：

```
import datetime
import time
import cv2
import dlib
from PIL import Image, ImageDraw, ImageFont
from PySide2.QtGui import Qt
from PySide2.QtWidgets import QApplication, QMessageBox
from PySide2 import QtWidgets
from PySide2.QtUiTools import QUiLoader
from PySide2 import QtGui, QtCore
from lib.share import SI
import lib.share as sa
import lib.dataDB as db
import traceback
#from PyCameraList.camera_device import test_list_cameras, list_video_devices, list_audio_devices
import numpy as np
import face as fa

# Dlib  正向人脸检测器
detector = dlib.get_frontal_face_detector()
# Dlib  人脸  landmark  特征点检测器
predictor = dlib.shape_predictor('data/dlib/shape_predictor_68_face_landmarks.dat')
# Dlib Resnet  人脸识别模型，提取  128D  的特征矢量
face_reco_model                                                                    =
dlib.face_recognition_model_v1("data/dlib/dlib_face_recognition_resnet_model_v1.dat")


class Client:
    def __init__(self):
        self.ui = QUiLoader().load('ui/face_recognition.ui')
        # self.ui.setWindowFlags(Qt.WindowCloseButtonHint)

        """加载人脸识别各种信息"""
        self.current_face_position = []

        #  加载已录入人脸特征  和  人员姓名  下标对应
        self.features = SI.features
        self.features_names = SI.f_uid

        #  帧数
        self.frame_cnt = 0

        #  用来存储上一帧和当前帧的质心坐标
        self.last_centriod = []
        self.current_centroid = []
```

```
        # 当前帧和上一帧检测出的目标的名字
        self.current_face_name = []
        self.last_face_name = []

        # 当前帧和上一帧的人脸数
        self.current_face_cnt = 0
        self.last_face_cnt = 0

        # 存储当前摄像头中捕获到的所有人脸的坐标名称
        self.current_face_position = []
        # 存放当前帧捕获到的人脸坐标和人脸特征
        self.current_face_feature = []

        # 当前帧和上一帧的质心欧式距离
        self.last_current_distance = 0
        # 存放识别的距离
        self.current_face_distance = []

        # 控制再识别的后续帧，如果识别出 unknow 的人脸，将 reclassify_cnt 计数到
reclassify_interval 后，对人脸进行重新识别
        self.reclassify_cnt = 0
        self.reclassify_interval = 20

        self.current_face = None

        """视频相关操作"""
        # 定义定时器，用于控制显示视频的帧率
        self.timer_camera = QtCore.QTimer()
        # 视频流
        self.cap = cv2.VideoCapture()
        # 为 0 时表示视频流来自笔记本内置摄像头
        self.CAM_NUM = 0

        """操作"""
        # 若定时器结束，则调用 show_camera()
        self.timer_camera.timeout.connect(self.show_camera)
        # 选择打开的摄像头是内置摄像头还是外置摄像头
        self.ui.select_camera.clicked.connect(self.select_camera_num)
        # 开始识别
        self.ui.begin_btn.setEnabled(False)
        self.ui.begin_btn.clicked.connect(self.run_rec)
        # 获取可用摄像头列表
        self.get_camera()

    """获取可用摄像头列表，如果可以下载摄像头相关函数，就用注释的内容"""

    def get_camera(self):
        #cameras = list_video_devices()
        #camera_dict = dict(cameras)
        #print(camera_dict)
        camera_name = []
        # if len(camera_dict) <= 0:
        #         print("the serial port can't find！")
```

```
            #         return False
            # else:
            #         for items in range(len(camera_dict)):
            #                 camera_name.append(str(items))
            # print("camera:", camera_name)
            camera_name.append(str(0))
            camera_name.append(str(1))
            self.ui.cameraList.addItems(camera_name)

    """摄像头画面展示"""

    def show_camera(self):

        try:
            start_time = time.time()
            # 从视频流中读取画面
            flag, img_rd = self.cap.read()
            # 如果可以读取到画面
            if flag:
                # 将图片进行对称
                img_rd = cv2.flip(img_rd, 1)
                image = img_rd.copy()
                # 把读到的帧的大小重新设置为 640x480
                img_rd = cv2.resize(img_rd, (640, 480))
                # 检测人脸  可以检测多个人脸 返回检测结果和人脸坐标(x,y,w,h)
                detections = fa.detect(img_rd)
                print("人脸数：", len(detections))
                faces = []
                mask_or_not = []
                for i in detections:
                    pos = i['position']
                    # 将坐标转化为 dlib 类型（左上，右下）
                    face = dlib.rectangle(pos[0], pos[1], pos[0] + pos[2], pos[1] + pos[3])
                    faces.append(face)
                    mask_or_not.append(i['class'])
                print(">>>目标检测结果>>>:", mask_or_not)
                print(">>>人脸坐标>>>:", faces)

                # 展示人脸数
                self.ui.personNumber.setText(str(len(detections)))
                # 更新人脸计数器
                self.last_face_cnt = self.current_face_cnt
                self.current_face_cnt = len(detections)

                # 更新上一帧的人脸姓名列表
                self.last_face_name = self.current_face_name[:]
                # 更新上一帧和当前帧的质心列表
                self.last_centriod = self.current_centroid
                self.current_centroid = []

                # 如果当前帧和上一帧人脸数未发生变化
                if (self.current_face_cnt == self.last_face_cnt) and (self.reclassify_cnt !=
self.reclassify_interval):
```

```
                    # 当前人脸坐标
                    self.current_face_position = []
                    if "unknow" in self.current_face_name:
                        self.reclassify_cnt += 1
                    if self.current_face_cnt != 0:
                        # k 表示第 k 个人脸
                        for k, d in enumerate(faces):
                            # 将每一个人脸坐标添加进当前帧人脸坐标库
                            self.current_face_position.append(tuple(
                                [faces[k].left(),  int(faces[k].bottom()  +  (faces[k].bottom()  -
faces[k].top()) / 4)]))
                            # 添加当前帧质心坐标
                            self.current_centroid.append(
                                [int(faces[k].left() + faces[k].right()) / 2,
                                 int(faces[k].top() + faces[k].bottom()) / 2])
                            # 人脸矩形框
                            x1 = d.top()
                            y1 = d.left()
                            x2 = d.bottom()
                            y2 = d.right()
                            # 判断人脸区域是否超出画面范围
                            if y2 > img_rd.shape[1]:
                                y2 = img_rd.shape[1]
                            elif x2 > img_rd.shape[0]:
                                x2 = img_rd.shape[0]
                            elif y1 < 0:
                                y1 = 0
                            elif x1 < 0:
                                x1 = 0

                            # 剪裁人脸 并在右侧区域显示  此处右边人脸框在人脸数多
的时候会发生变化
                            crop = img_rd[x1:x2, y1:y2]
                            self.current_face = crop
                            self.display_face(crop)

                            # 人脸框的位置
                            rect = (d.left(), d.top(), d.right(), d.bottom())
                            # 人名字标签
                            name_lab = self.current_face_name[k] if self.current_face_name !=
[] else ""
                            name_lab = str(name_lab) + "-" + str(mask_or_not[k])
                            print("current_face_name:", name_lab)
                            # 画出人脸框
                            image = self.drawRectBox(image, rect, name_lab)
                            self.ui.name.setText(str(self.current_face_name[k]))
                            self.ui.maskMark.setText(str(mask_or_not[k]))

                    # 在画面中显示
                    self.display_img(image)

                    # 如果当前帧中不止一个人脸 使用质心追踪算法
                    if self.current_face_cnt != 1:
```

```
                        self.centroid_tracker()


                # 如果人脸数发生变化
                else:
                    self.current_face_position = []
                    self.current_face_distance = []
                    self.current_face_feature = []
                    self.reclassify_cnt = 0

                    # 如果人脸数为 0
                    if self.current_face_cnt == 0:
                        # 清空姓名和特征
                        self.current_face_name = []
                        self.current_face_feature = []
                    # 人脸数增加
                    else:
                        self.current_face_name = []
                        for i in range(len(faces)):

self.current_face_feature.append(sa.return_face_recognition_result(img_rd, faces[i]))
                            self.current_face_name.append("unknow")
                        print("self.current_face_name:", self.current_face_name)
                        for k in range(len(faces)):
                            x1 = faces[k].top()
                            x2 = faces[k].left()
                            y1 = faces[k].bottom()
                            y2 = faces[k].right()
                            self.current_face = img_rd[x1:y1, x2:y2]
                            self.current_centroid.append([int(faces[k].left() + faces[k].right()) /
2,
                                                          int(faces[k].top()                    +
faces[k].bottom()) / 2])
                            self.current_face_distance = []
                            # 每个捕获人脸的名字坐标
                            self.current_face_position.append(tuple(
                                [faces[k].left(),  int(faces[k].bottom()  +  (faces[k].bottom()  -
faces[k].top()) / 4)]))
                            # 遍历人脸库
                            print("人脸库：", self.features)
                            for i in range(len(self.features)):
                                if str(self.features[i][0]) != '0.0':
                                    e_distance_tmp                                          =
sa.return_euclidean_distance(self.current_face_feature[k],

self.features[i])
                                    self.current_face_distance.append(e_distance_tmp)
                                else:
                                    self.current_face_distance.append(99999999999)
                            print("self.current_face_distance:", self.current_face_distance)
                            # 寻找出最小的欧式距离匹配
                            min_dis = min(self.current_face_distance)
                            similar_person_num = self.current_face_distance.index(min_dis)
```

```
                                if min_dis < 0.4:
                                    self.current_face_name[k] = self.features_names[similar_person_num]
                                    # 获取系统当前时间
                                    curr_time = datetime.datetime.now()
                                    curTime = datetime.datetime.strftime(curr_time, '%Y-%m-%d %H:%M:%S')
                                    print("current_Time:", curTime)
                                    # 设置图片存储路径
                                    filePath = "record_img/" + str(curr_time.year) + "_" + str(curr_time.month) + "_" + str(
                                        curr_time.day) + "_" + str(curr_time.hour) + "_" + str(
                                        curr_time.minute) + "_" + str(
                                        curr_time.second) + ".jpg"
                                    print("filePath:", filePath)
                                    # 存储图片
                                    cv2.imencode('.jpg', self.current_face)[1].tofile(filePath)
                                    # 存储当日通行记录
                                    sql = "insert into record(username,time,pic) values ('%s','%s','%s')" % (
                                        self.current_face_name[k], curTime, filePath)
                                    print("通行记录:", sql)
                                    result = db.insertDB(sql)
                    end_time = time.time()
                    if end_time == start_time:
                        use_time = 1
                    else:
                        use_time = end_time - start_time
                    fps = int(1.0 / round(use_time, 3))
                    self.ui.fps.setText(str(fps))
            except Exception as e:
                traceback.print_exc()
                QMessageBox.critical(
                    self.ui,
                    "人脸识别错误",
                    str(e)
                )

    """摄像头选择事件"""

    def select_camera_num(self):
        try:
            if not self.timer_camera.isActive():  # 若定时器未启动
                self.CAM_NUM = int(self.ui.cameraList.currentText())
                print("self.CAM_NUM:", self.CAM_NUM)
                if self.CAM_NUM == 0:
                    self.ui.log.setText("内置摄像头准备就绪")
                else:
                    self.ui.log.setText("外置摄像头准备就绪")
                # 初始化数据
                self.ui.name.setText("unknow")
                self.ui.maskMark.setText("nomask")
                self.ui.personNumber.setText("0")
                self.ui.fps.setText("0")
```

```
                self.ui.cameraLabel.setPixmap(QtGui.QPixmap("img/camera_label.jpg"))
                self.ui.personX.setPixmap(QtGui.QPixmap("img/qql.jpg"))
            self.ui.begin_btn.setEnabled(True)

        except Exception as e:
            traceback.print_exc()
            QMessageBox.critical(
                self.ui,
                "摄像头选择函数出错了",
                str(e)
            )

    """画人脸框"""

    def drawRectBox(self, image, rect, addText):
        try:
            cv2.rectangle(image, (int(round(rect[0])), int(round(rect[1]))),
                          (int(round(rect[2])), int(round(rect[3]))),
                          (0, 0, 255), 2)
            cv2.rectangle(image, (int(rect[0] - 1), int(rect[1]) - 16), (int(rect[0] + 75), int(rect[1])),
(0, 0, 255),
                          -1, cv2.LINE_AA)
            img = Image.fromarray(image)
            draw = ImageDraw.Draw(img)
            font_path = "Font/platech.ttf"
            font = ImageFont.truetype(font_path, 14, 0)
            draw.text((int(rect[0] + 1), int(rect[1] - 16)), addText, (255, 255, 255), font=font)
            imagex = np.array(img)
            return imagex
        except Exception as e:
            traceback.print_exc()
            QMessageBox.critical(
                self.ui,
                "画人脸框错误",
                str(e)
            )

    """加载视频区域"""

    def display_img(self, image):
        try:
            # self.label_display.clear()
            image = cv2.resize(image, (640, 480))  # 设定图像尺寸为显示界面大小
            show = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            showImage = QtGui.QImage(show.data, show.shape[1], show.shape[0], show.shape[1]
* 3,
                                     QtGui.QImage.Format_RGB888)
            self.ui.cameraLabel.setPixmap(QtGui.QPixmap.fromImage(showImage))
            self.ui.cameraLabel.setScaledContents(True)
            QtWidgets.QApplication.processEvents()
        except Exception as e:
            traceback.print_exc()
            QMessageBox.critical(
                self.ui,
```

```
                    "加载视频区域错误",
                    str(e)
                )

    """在右边小框显示人脸"""

    def display_face(self, image):
        try:
            # 清空右侧显示区域
            self.ui.personX.clear()
            if image.any():
                # 图片颜色转化
                show = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                showImage = QtGui.QImage(show.data, show.shape[1], show.shape[0],
show.shape[1] * 3,
                                         QtGui.QImage.Format_RGB888)
                self.ui.personX.setFixedSize(200, 200)
                self.ui.personX.setPixmap(QtGui.QPixmap.fromImage(showImage))
                self.ui.personX.setScaledContents(True)
                QtWidgets.QApplication.processEvents()
        except Exception as e:
            traceback.print_exc()
            QMessageBox.critical(
                self.ui,
                "显示人脸框错误",
                str(e)
            )

    """ 使用质心追踪来识别人脸"""

    def centroid_tracker(self):
        for i in range(len(self.current_centroid)):
            distance_current_person = []
            # 计算不同对象间的距离
            for j in range(len(self.last_centriod)):
                self.last_current_distance = sa.return_euclidean_distance(
                    self.current_centroid[i], self.last_centriod[j])

                distance_current_person.append(self.last_current_distance)

            last_frame_num = distance_current_person.index(min(distance_current_person))
            self.current_face_name[i] = self.last_face_name[last_frame_num]

    """点击开始运行按钮执行函数"""

    def run_rec(self):
        self.features, self.features_names = sa.get_all_features()
        try:
            # 如果定时器未启动
            if not self.timer_camera.isActive():
                QtWidgets.QApplication.processEvents()
                flag = self.cap.open(self.CAM_NUM)
                if not flag:
                    QMessageBox.critical(
```

```
                        self.ui,
                        "错误",
                        "摄像头打开失败，请检查连接后重新打开"
                    )
                else:
                    self.ui.log.setText("正在打开摄像头，请稍等")
                    # 打开定时器
                    self.timer_camera.start(30)
                    # 将摄像头按钮的文字置为结束识别
                    self.ui.begin_btn.setText("结束识别")
                    self.ui.log.setText("摄像头打开成功")
                    self.ui.begin_btn.setStyleSheet("background-color: red;\n"
                                                    "color: rgb(255, 255, 255);\n"
                                                    "font: 87 14pt \"Arial Black\";\n"
                                                    "border-radius:10px;")
                    self.features = SI.features
                    self.features_names = SI.f_uid
            else:
                # 关闭定时器
                self.timer_camera.stop()
                # 释放视频流
                self.cap.release()
                self.ui.log.setText("摄像头已关闭")
                # 清空视频显示区域
                self.ui.cameraLabel.clear()
                self.ui.personX.clear()
                self.ui.cameraLabel.setPixmap(QtGui.QPixmap("img/camera_label.jpg"))
                self.ui.personX.setPixmap(QtGui.QPixmap("img/qql.jpg"))
                # 重置识别按钮
                self.ui.begin_btn.setText("开始识别")
                self.ui.begin_btn.setStyleSheet("background-color: rgb(104, 164, 253);\n"
                                                "color: rgb(255, 255, 255);\n"
                                                "font: 87 14pt \"Arial Black\";\n"
                                                "border-radius:10px;")
                # 重置区域文字
                self.ui.name.setText("unknow")
                self.ui.maskMark.setText("nomask")
                self.ui.personNumber.setText("0")
                self.ui.fps.setText("0")
                QtWidgets.QApplication.processEvents()
        except Exception as e:
            traceback.print_exc()
            QMessageBox.critical(
                self.ui,
                "打开摄像头错误",
                str(e)
            )


if __name__ == "__main__":
    app = QApplication([])
    SI.loginWin = Client()
    SI.loginWin.ui.show()
```

```
app.exec_()
```