基于龙芯派 2 代的无感人脸识别考勤系统

全芯全意；何烨；王佳豪；宋健

指导老师：黄迪

目录

摘要

随着人工智能（AI）技术的快速发展，特别是深度学习和计算机视觉技术的进步，人脸识别技术取得了显著的进展。本项目旨在设计一款基于深度学习的无感人脸识别考勤系统，以龙芯派二代作主控板，采用预训练的深度学习模型提取人脸特征，通过欧式距离进行人脸匹配，使用软硬件结合的方式实现高效、自动化的考勤管理。与传统考勤方式相比具有非接触，高精度，识别速度快，操作简便，防作弊等特点，可广泛运用于中小型企业，工厂，学校等场景，提供了一种高效可靠的考勤管理解决方案。

第一部分 作品概述

1.1 功能与特性

精确且快速的人脸识别能力：

- 系统采用基于 yunet 网络的深度学习模型作为人脸检测器，能够在边缘设备以毫秒级快速准确地识别出复杂环境下的人脸。
- 系统采用基于 MobileFaceNet-V2 网络训练的深度学习模型作为人脸特征提取器，在保证高准确的识别率下同时极大优化了原模型的推理速度，保证了人脸识别的精度和速度。

高度自动化的考勤管理：

- 自动统计与记录员工或学生的出勤情况，减少了人工干预和统计错误的可能性。
- 考勤数据实时同步更新后台，便于管理者随时查看和分析。

可视化界面与实时反馈：

- 提供直观、简洁的 UI 界面，用户可以实时查看考勤记录和结果。
- 支持语音播报功能，实时反馈考勤状态，提升使用的交互体验。

多场景适用性：

- 适用于各类场所的考勤管理，包括企业、学校、医院等，满足不同场景下的考勤需求。
- 也可用于门禁系统，控制重要区域的进出，确保设施和数据的安全性。

1.2 应用领域

本系统主要应用于各类需要精准考勤管理场所，如公司企业、学校、科研机构、工厂和医院等。在公司企业和工厂，系统可有效管理员工的出勤情况，

减少人工统计的误差和工作量，提高人力资源管理效率。在学校和科研机构，可以用于学生和教职工的出勤记录，确保校园安全和秩序。在医院等医疗场所，系统可以帮助管理医护人员的考勤，确保医疗服务的有序进行。此外基于人脸识别的门禁系统还可用于重要区域的安全控制，如实验室、机房和数据中心等，防止未经授权人员进入，保障内部设施和数据的安全。通过将智能考勤与门禁管理结合，系统在各领域都能实现高效、便捷的人员管理。

1.3 主要性能指标

测试环境和硬件配置	win11，pytorch3.8.19，NVIDIA RTX3060GPU，IntelCorei7-12700H CPU,python3.6,Vscode
训练数据集	CASIA-Webface 数据集,共有 10575 个人,494414 张人脸图像
测试数据集	LFW 数据集,共有 5749 个人,13232 张人脸图像
测试对象	mobileFaceNet-V2 网络
配置信息	损失函数为 SFace,优化器采用 SAM,迭代次数 epoch=125，batch_size=128,学习率 lr=0.1(在 epoch=50,70,80，学习率下降/10)
性能指标	我们的模型在验证集 lfw 上人脸识别准确度达到 99.1%!提取一张人脸图像的特征仅需 0.011s!。

1.4 主要创新点

MobileFaceNet-V2 神经网络：

该网络的核心设计思想是先对特征图升维,在更深的通道上提取丰富的空间信息,再对特征图进行降维，减少计算量,为了防止降维带来的空间信息丢失,同时采用两次深度卷积,并置于两个 1×1 逐点卷积之外,从而减少空间信息编码的丢失,保证更多的特征信息从输入端传递到输出端。除此之外损失函数采用了 SFace，SFace 可以增加在过拟合和欠拟合之间找到最佳折衷的可能性，从而获得更好的泛化能力,优化器采用 SAM，它能够适应不同的训练数据和模型复杂度，提供稳健的优化性能和更强的泛化能力。

1.5 设计流程

本项目主要涵盖三大核心模块，深度学习模型的训练与部署，考勤客户端的开发和后台应用程序的开发。主要设计流程如下：

（1） 进行需求分析，将项目的需求进行拆分，确定项目需要的技术框架，制定项目解决方案以及团队之间各成员的任务；

（2） 选择合适的深度学习框架，通过框架提供高级的抽象接口，便于我们设计强大，具有高鲁棒性的深度学习模型；

（3） 选择合适开源训练集，模型通过大量的数据，以便学习到数据中的模式和规律；

（4） 选择合适的开源验证集，通过验证数据集可以评估模型的性能，以便调整模型的超参数和架构，使其更好地适应数据集；

（5） 设计人脸特征提取网络，利用卷积神经网络的特性，设计适用于目标平台的特征提取网络，提取人脸特征；

（6） 选择合适的损失函数：作为优化算法的目标函数，通过最小化该函数来指导模型的训练过程，适当增加人脸类间损失和人脸类内损失；

（7） 选择合适的优化器：用于调整神经网络模型的参数，以最小化损失函数，使模型能够更好地拟合训练数据，并在未见过的数据上表现良好；

（8） 配置超参数：为了优化模型的训练过程和最终效果，我们需要不断的微调整模型的学习率，批处理大小等参数，得到一组最适合模型的参数；

（9） 训练模型并部署：通过多轮迭代训练模型，监控模型性能变化，将经过验证集验证并且表现优异的模型保存，在最终训练结束后将最优异的模型转换成机器学习通用格式，以便部署在龙芯派二代；

（10）客户端 UI 界面设计：设计直观友好的用户界面，使用户方便于系统交互。

（11）数据库设计：设计一个高效、可靠的数据库结构，用于存储用户的学号，班级，姓名，照片等信息；

（12）实时人脸识别算法的设计：使用预先训练的深度学习模型和通过多线程，互斥锁，缓存等技术实现用户的实时识别；

（13）通信协议的设计：设计合适的通信协议确保客户端和后台程序数据的可靠传输；

（14）后台登录界面设计：通过输入账户、密码登录，方便管理员登录管理用户信息和查看考勤记录；

（15）用户管理界面设计：设计一个直观且易于操作的用户管理界面，允许管理员添加、编辑、删除和查看用户信息。

（16）考勤管理界面设计：用于设置考勤时间和查看、管理用户的考勤情况；

设计流程图如下：

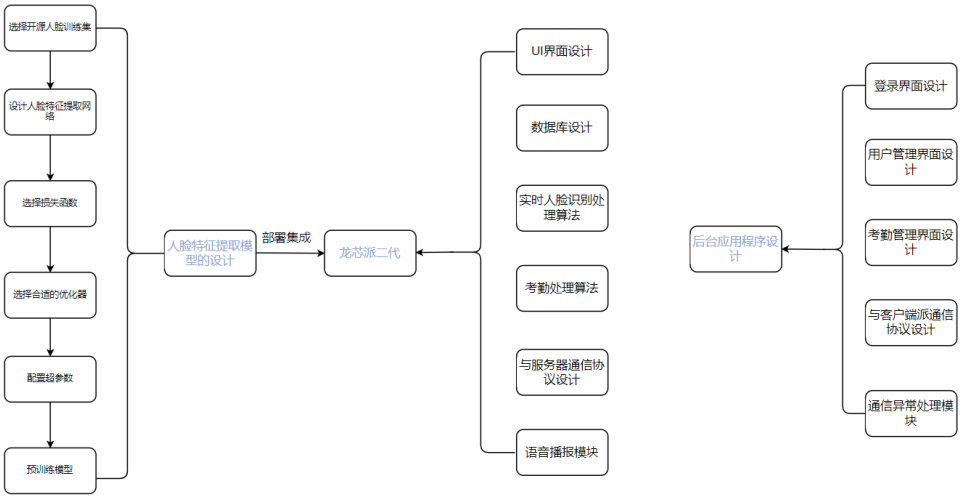


图 1.1 设计流程图

第二部分 系统组成及功能说明

2.1 整体介绍

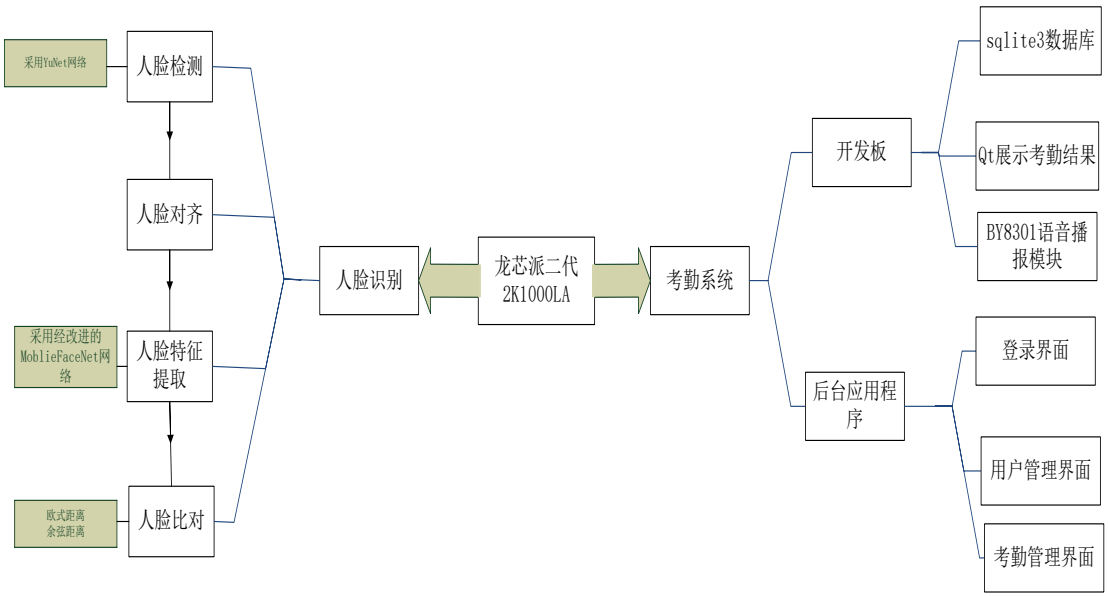


图 2.1 系统总体设计框图

本项目的系统框图展示了一套基于深度人脸识别的考勤系统，采用龙芯派

2 代开发板（2K1000LA）作为核心硬件平台，结合多种软硬件模块来实现高效、准确的考勤管理。系统包括人脸检测与识别模块、核心处理模块、外设及用户界面、以及后台应用程序，各模块之间密切协作，确保系统的整体功能。

系统的工作流程如下：管理员通过账户密码登录后台管理系统，进入考勤管理界面设置考勤时间并将设置发送到客户端。客户端利用摄像头实时采集图像信息，并通过 OpenCV 库进行图像预处理。预处理后的图像输入人脸检测模型进行推理，获取人脸关键特征坐标（眼睛、鼻子、嘴巴）。接着，对检测到的人脸进行对齐处理，并将对齐后的图像送入人脸特征提取模型，得到每张人脸的 512 维特征向量。

系统会将实时检测到的人脸特征向量与预先缓存的特征向量进行欧式距离或余弦距离的比对。若距离结果在设定的阈值范围内，则确认为同一张人脸。根据系统设置的考勤时间进行考勤结果判定，龙芯派将考勤结果显示在 UI 界面上，并通过驱动的语音播报模块进行实时语音提示。同时，客户端将考勤信息实时发送回服务器端，服务器端据此更新考勤记录。

2.2 硬件整体介绍

核心硬件平台：龙芯派二代开发板（2K1000LA）

龙芯派搭载龙芯最新一代的嵌入式处理器 2K1000LA,提供了包括 USB、GMAC、SATA、 PCIE 在内的主流接口,可以满足多场景的产品化应用,也是进行国产化开发的入门级硬件的首选。该开发板的特性如下:

功能	描述
CPU	龙芯 2K1000 处理器
内存	板载 2G DDR3，主频 400Mhz
Bios	8Mb SPI FLASH
GPIO	2.54 间距 27 个可配置 GPIO 插针排
网络	2 个千兆自协商网口（2 个标准接口）
PCIE	1 路 X1 夹板接口 PCIE
EJTAG	1 个 EJTAG 调试接口，可用于程序下载、单步调试
接口	3 路 USB2.0 标准接口（TYPE A USB*2, Micro USB*1） 2 路 CAN 接口，4 路串口(TTL*3, RS232*1)
显示和音频接口	1 路 TYPE A HDMI 接口 DVO 接口适配飞凌嵌入式触摸屏 1 路 3.5mm 标准音频输入/输出接口
存储	M2 接口 支持 SSD 硬盘
电源	12V 3A 圆柱电源
尺寸	120mm*120mm

图 2.2 龙芯派硬件参数图

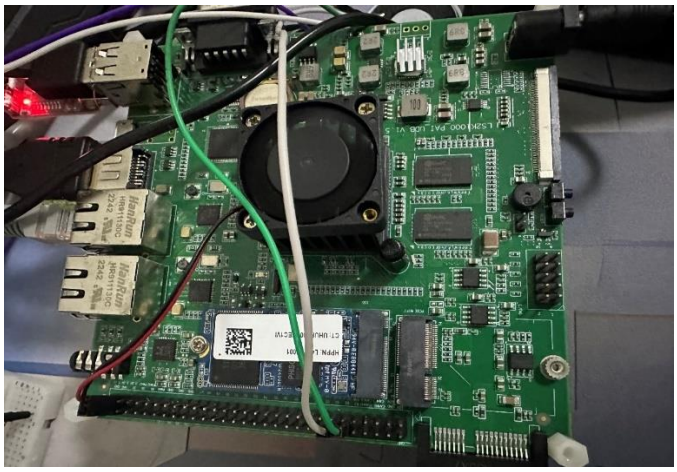


图 2.3 开发板高清特写

USB 摄像头：负责实时采集图像信息，摄像头分辨率为 1080p，可通过旋转镜头环调节摄像头焦距；具体参数信息如下：

室内强光 低照度拍摄

产品型号	SY011HD
分辨率/帧率	200万硬件 1920*1080P 30帧
可视角度	2.45mm(广角150度），不同角度联系客服
麦克风	数字麦克风（默认出货不带麦克风 特殊需要请联系客服）
供电方式	USB bus power
工作电压	5V
工作电流	200ma
默认格式	出厂设置MJPG/YUV/YUV2
接口类型	USB2.0
对焦方式	手动调焦
对焦方式	自动曝光/手动曝光
信躁比	62dB
动态范围	88dB
支持免驱协议	USB2.0支持uvc通信协议
支持OTG协议	USB2.0-OTG
线长	1.5米/2米/3米/5米（默认2米出货，其它联系客服）
工作温度	-20 to 70度
支持系统	WindowsXP Vista Win7 Win8 Win10 linux Ubuntu 树莓派等
支持系统	安卓设备、广告机、ATM柜员机、工业设备、网络视频会议、人脸识别监控录像、POS机、超市、医疗设备、安卓设备、网络机顶盒、视频教学网络电视盒一体机设备等

图 2.4 摄像头参数图

图 2.5 USB 摄像头实拍

显示屏：开发板通过 HDMI 接口和显示屏连接，用于显示客户端 UI 界面；

图 2.6 HDMI 屏幕

语音播报模块： 采用 BY8301-QSOP24 MP3 主控芯片，支持 MP3、WAV 格式双解码。模块内置 SPI-FLASH 作为存储介质，配有 Microc USB 接口；使用示意图如下：

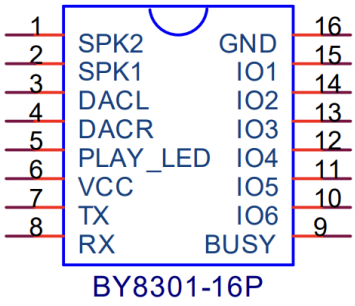


图 2.7 BY8301-16 原理图

管脚号	管脚名称	功能描述	备注
1	SPK2	外接单声道无源喇叭两端	接2W/4Ω或1W/8Ω以上喇叭
2	SPK1		
3	DACR	DAC 右声道输出	可外接功放、耳机
4	DACL	DAC 左声道输出	可外接功放、耳机
5	PLAY_LED	播放指示灯，停止常亮，播放闪烁	输出高电平，LED 负极接地
6	VCC	电源正极	3.6-5V
7	TX	UART 异步串口数据输出	3.3V 的 TTL 电平
8	RX	UART 异步串口数据输入	3.3V 的 TTL 电平
9	BUSY	播放时输出高电平，停止为低	忙信号
10	IO6	触发输入口6	接地触发
11	IO5	触发输入口5	接地触发
12	IO4	触发输入口4	接地触发
13	IO3	触发输入口3	接地触发
14	IO2	触发输入口2	接地触发
15	IO1	触发输入口1	接地触发
16	GND	电源负极	系统地，外接大功率功放时，大电流不要流过这个地脚

图 2.8 管脚功能图

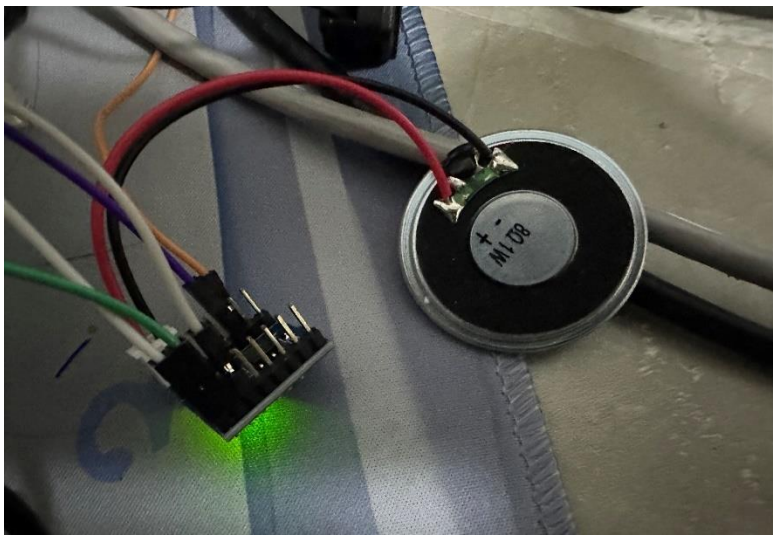


图 2.9 语音播报模块实拍

2.3 软件整体介绍

2.3.1 基于 MobileFaceNet-V2 的深度学习模型

项目采用了 PyTorch 深度学习框架，我们设计了MobileFaceNet-V2作为主干网络，该网络较于传统的卷积神经网络具有模型参数少，计算精度高的优点。为了提升模型的区分能力，我们使用了 SFace（sigmoid 限制超球面损失）作为损失函数，适当增加了类间损失以减少类内损失，从而提升模型的泛化能力和识别精度。在优化器的选择上采用了 SAM（锐度感知最小化），确保模型训练到损失最平坦的最小值，进一步增强模型的泛化能力。训练数据集使用了CASIA-Webface，涵盖了丰富的面部特征信息，有助于模型的充分训练。最终，将训练得到的模型权重和网络转换为 ONNX 格式，以便在龙芯派等平台上高效部署和运行。网络结构图如下：

输入	卷积操作	c输出通道	s步长	输出
112*112*3	conv3×3	64	2	56*56*64
56*56*64	conv3×3	64	1	56*56*64
56*56*64	sandglass	64	2	28*28*64
28*28*64	sandglass	64	1	28*28*64
28*28*64	sandglass	64	1	28*28*64
28*28*64	sandglass	64	1	28*28*64
28*28*64	sandglass	64	1	28*28*64
28*28*64	sandglass	128	2	14*14*128
14*14*128	sandglass	128	1	14*14*128
14*14*128	sandglass	128	1	14*14*128
14*14*128	sandglass	128	1	14*14*128
14*14*128	sandglass	128	1	14*14*128
14*14*128	sandglass	128	1	14*14*128
14*14*128	sandglass	128	1	14*14*128
14*14*128	sandglass	128	2	7*7*128
7*7*128	sandglass	128	1	7*7*128
7*7*128	sandglass	128	1	7*7*128
7*7*128	conv1×1	512	1	7*7*512
7*7*512	conv7×7	512	1	1*1*512

图 2.10 网络结构图

2.3.2 人脸考勤客户端程序

人脸识别客户端程序基于 C/C++ 开发，摄像头实时采集图像，使 OpenCV 库进行图像的预处理，避免图像因光照，尺寸等因素影响深度学习模型的推理。使用多线程、线程同步、缓存等技术实现实时人脸识别考勤的算法；客户端的 UI 界面基于 Qt 开发，主窗口继承 Widget 父类，界面布局基于 QLabel 进行摆放， 并且使用了多种控件，如 Qtimer 定时器等。

采用 sqlite3 数据库进行用户信息的存储，包括用户姓名，班级，照片等，通信协议基于 Tcp/Ip 开发，在该协议的基础上，通过自定义的应用层封装实现考勤数据的可靠传输。主要处理流程如下：

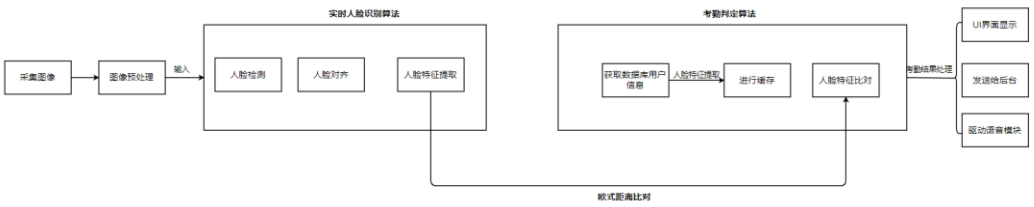


图 2.11 考勤人脸识别流程图

2.3.3 考勤后台应用程序

考勤后台应用程序基于 Qt5 开发，涵盖四大部分：登录界面，用户管理界面，考勤管理界面，异常处理机制。登录界面包括账户，密码字段，以表单的方式提交；用户管理界面使用 QTableView 以表格的形式显示用户信息，通过自定义委托类重载 paint 方法实现表格字段(主要是图像数据)的居中显示，使用 QMessageBox 对话框用于用户交互实现对用户的增删查改操作；考勤管理界面使用 QtableView 以表格的形式显示用户的考勤信息，通过实现自定义控件 Pagination 实现界面分页显示，使用 LCD_NUMBER 组件实现实时时间的显示，使用 QText 实现考勤时间的设置。主要工作流程如下：

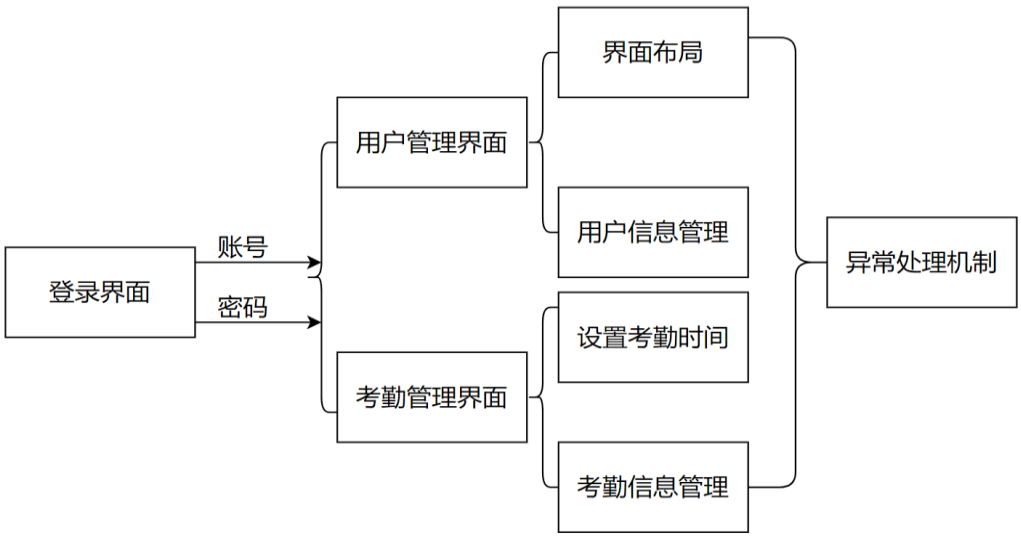


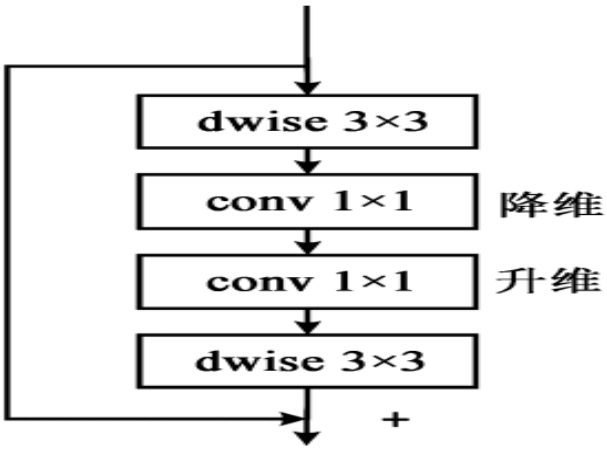
图 2.12 考勤后台界面工作流程

2.4 核心算法设计

2.4.1 MobileFaceNet-V2 网络的设计

（1）sandglass 模块的设计

传统人脸识别算法多采用全局平均池化层处理特征图， 但是经最新研究发现特征图不同位置的重要性也是不同的,平均池化层会弱化关键信息的提取。因此我们设计实现 sandglass 模块处理最后得到的特征图，sandglass 模块先对特征图升维,在更深的通道上提取丰富的空间信息,再对特征图进行降维，减少计算量,为了防止降维带来的空间信息丢失,模块采用两次深度卷积,并置于两个 1×1 逐点卷积之外,从而减少空间信息编码的丢失,保证更多的特征信息从输入端传递到输出端。表 2 表示一个尺寸为 DF×DF×M 的特征图经过 sandglass 模块处理得到 DF/s×DF/s×N 的特征图,s 为步距,t 为缩小系数。



输入	卷积操作	输出
$D_F \times D_F \times M$	3×3 dwise，ReLu6	$D_F \times D_F \times M$
$D_F \times D_F \times M$	1×1 conv，linear	$D_F \times D_F \times \frac{M}{t}$
$D_F \times D_F \times \frac{M}{t}$	1×1 conv，ReLu6	$D_F \times D_F \times N$
$D_F \times D_F \times N$	3×3 dwise s，linear	$\frac{D_F}{s} \times \frac{D_F}{s} \times N$

图 2.13 sandglass 模块处理过程

在不考虑捷径分支的情况下,一个尺寸为 DF×DF×M 的特征图经过卷积操作得到 DF×DF×N 的特征图,bottleneck 模块和 sandglass 模块的计算量分别为

Cbtn =

1×1×M×teM×DF×DF+DK×DK×teM×teM×DF×DF+1×1×teM×N×DF×DF

Csg = DK×DK×M×M×DF×DF+1×1×

M×Mtr×DF×DF+1×1×Mtr×N×DF×DF+DK×DK×N×N×DF×DF

二者计算量之比为

$$\eta = \frac{C_{sg}}{C_{btn}} = \frac{D_K^2(M^2+N^2) + \left(\frac{M^2}{t_r} + \frac{MN}{t_r}\right)}{t_e^2 D_K^2 M^2 + t_e(M^2+MN)}$$

式中,DK 为卷积核的宽度和高度,通常取 3;te,tr 分别为 bottleneck 的扩大系数和 sandglass 的缩小系数,通常取 2 至 6,而一般情况下 M≤N≤2M 因此,可以得到:

$$\eta \leqslant \frac{2D_K^2M^2+\frac{2M^2}{t_r}}{t_e^2D_K^2M^2+3t_eM^2} \leqslant \frac{2D_K^2+2}{t_e^2D_K^2+3t_e} = \frac{20}{9t_e^2+3t_e}$$

可以看到 te 的值越大,采用 sandglass 模块能够减少的计算量就越明显。大量实验表明,线性瓶颈结构有利于避免出现 特征零化现象,减少信息丢失。因此,在 sandglass 模块的第一个逐点卷积层和第二个深度卷积层采取线性输出。而 PReLU 函数已经证明在人脸识别算法中的性能优于 ReLU6 函数。因此 sandglass 模块另外两层改用 PReLU 激活函数。

（2）SFace 损失函数

SFace 算法对分别由两个乙状结构梯度重新缩放函数控制的超球面流形施加类内和类间约束。乙状结构曲线精确地重新调整了类内和类间梯度，以便在一定程度上优化训练样本。因此，SFace 能够在减小类内距离和防止过度拟合标签噪声之间取得较好的平衡，为深度人脸识别模型提供更强的鲁棒性。SFace 具有如下特点：超球面约束，SFace 将特征嵌入限制在嵌入空间的超球面上。通过将特征向量归一化为单位范数，实现此约束。这种约束有助于通过关注特征向量之间的相对距离而不是它们的绝对值，提高学习嵌入的区分能力。如图，乙状结构约束超球面损失的示意图，它对超球面流形施加类内和类间约束。样本和目标嵌入的优化方向总是沿着超球面的切线，而移动速度分别由两条乙状结构曲线控制。具体来说，深部特征 xi 及其目标中心 wyidecrease 的移动速度随着它们的接近而逐渐减小，而 xi 及其他目标中心 Wj 的移动速度则随着它们开始接近而迅速增加；过拟合和欠拟合间良好平衡：与严格优化训练样本相比，SFace 的优点是在过拟合和欠拟合之间提供了较好的平衡，因为 SFace 分别采用类内和类间梯度重尺度项的乙状结构函数来实现良好的控制。我们在图 2 中给出了一个简单易懂的例子。在标签噪声设置下,通过严格地将噪声样本拖拽到错误的标签标识上，模型会过度适应标签噪声。相比之下，SFace 可以在一定程度上缓解这个问题，因为它适度地优化了噪声样本。通过精确的控制，清洁的训练样本可以更早更容易地优化，而标签噪声可以留在后面。所以 SFace 可以增加在过拟合和欠拟合之间找到最佳折衷的可能性，从而获得更好的泛化能力。

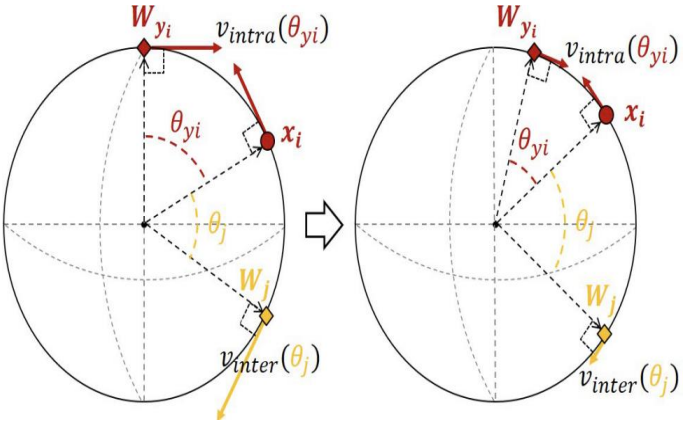


图 2.14 乙状结构约束超球面损失示意图

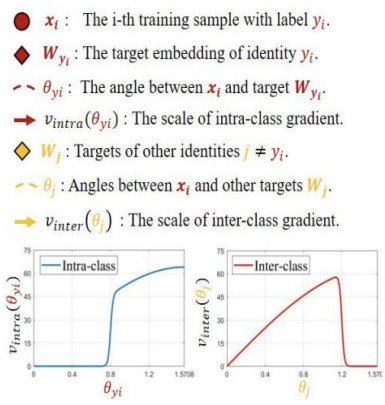


图 2.15 SFace 过滤标签噪声

Sigmoid 激活：在一对样本的特征嵌入之间的点积上应用 Sigmoid 函数。这种 Sigmoid 激活将点积转换为介于 0 和 1 之间的类似概率的分数，表示样本之间的相似度或不相似度。SFace 基于经过 Sigmoid 转换的点积构建损失函数。通常，它结合了鼓励类内紧凑性（相似的特征向量应具有较高的相似度分数）和类间可分性（不相似的特征向量应具有较低的相似度分数）的组成部分。SHL 的目标是学习特征嵌入，使其对光照、姿势和表情变化等常见的人脸识别挑战具有鲁棒性。通过在超球面上嵌入特征并使用 Sigmoid 约束，SHL 旨在比传统损失函数更好地推广和鲁棒性。这种方法有助于在具有挑战性的人脸识别场景中实现最先进的性能。该算法在 CASIA-WebFace、vggface2 和 MS-Celeb-1M 数据库上进行了大量的模型实验，并在 LFW、megface 和 ijb-c 数据库等人脸识别基准上进行了评估，证明了 SFace 的优越性。

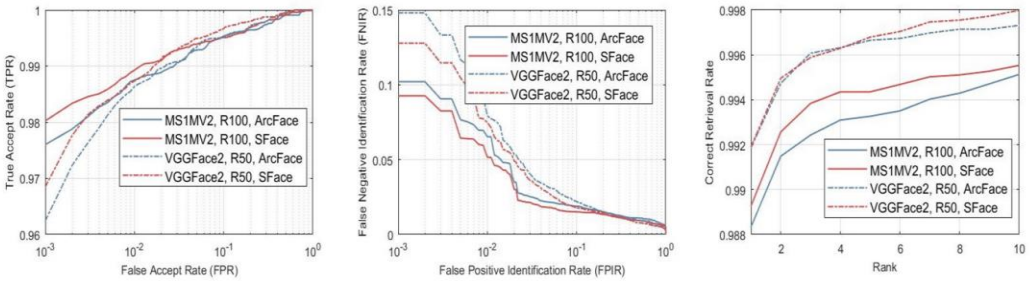


图 2.16 SFace 与 ArcFace 基于 IJA-B 数据集的性能对比

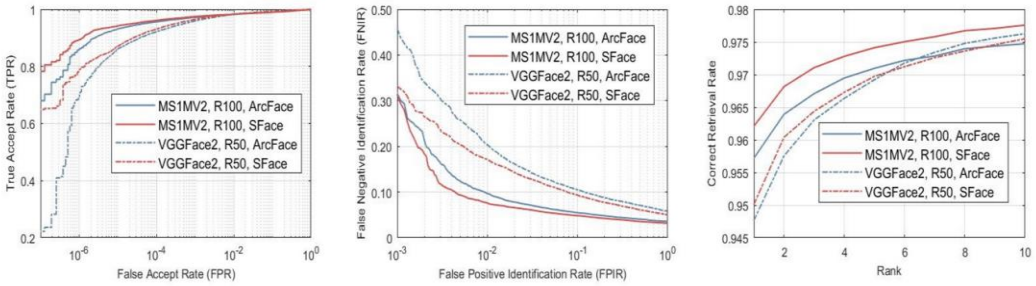


图 2.17 SFace 与 ArcFace 基于 IJB-C 数据集的性能对比

（3）SAM 优化器

优化器在神经网络训练中起着至关重要的作用，它直接影响着模型收敛速度、泛化能力以及最终的性能表现。。对比于传统的优化器，SAM 优化器泛化能力更强，SAM 优化器通过考虑模型参数的“锐度”（sharpness），即损失函数在参数空间中的平坦程度，来调整学习率。这种机制有助于避免模型过度拟合训练数据，从而提高模型在未见过的测试数据上的泛化能力；SAM 优化器也具有更快的收敛速度，SAM 优化器能够根据损失函数的梯度方向和锐度来动态调整学习率，使得模型在训练初期能够更快地收敛到合适的解。这有助于加速训练过程，尤其是对于大规模和复杂的卷积神经网络。

实现 SAM（Sharpness-Aware Minimization）优化器涉及到对梯度和参数的处理，以及尖锐度的计算和更新。SAM 的核心思想是通过额外的步骤来调整参数更新的方向，以减少损失函数关于参数的二阶导数（即尖锐度），从而提高模型的泛化能力和收敛速度。首先需要安装 SAM 库，可以通过以下命令使用 pip 安装 SAM；在代码中导入 SAM 相关的库；然后，定义模型和标准 PyTorch 优化器如 SGD；使用 SAM 替代标准的 PyTorch 优化器，并指定相关参数，例如基础优化器、rho 参数等； 这里的 rho 参数控制了尖锐度的大小，影响学习率的动态调整程度；在每个训练迭代中，首先调用 optimizer.zero_grad() 清除梯度，然后计算损失和反向传播。在反向传播后，通过 optimizer.step() 更新参数，并通过 optimizer.zero_grad(set_to_none=True) 保留梯度用于尖锐度的计算；

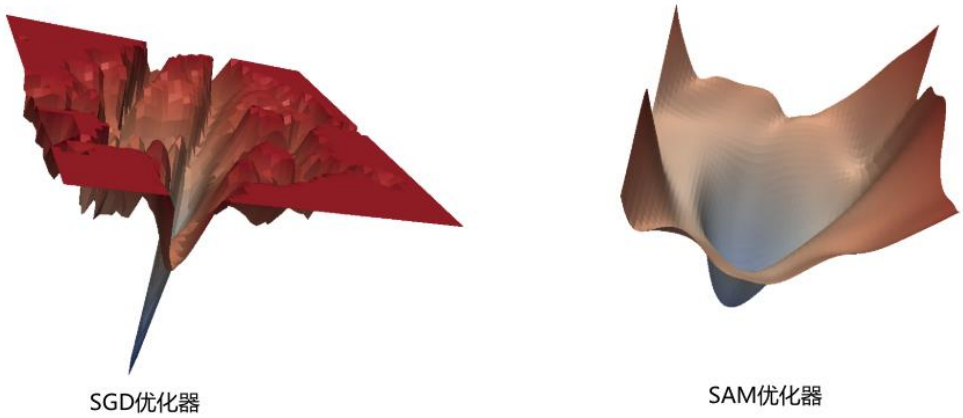


图 2.18 SAM 优化示意图

2.4.2 图像预处理算法

在将摄像头采集到的图像送入到深度学习模型进行预测之前，要对图像进行预处理操作,减少图像中噪声和干扰，确保输入到模型中的图像具有一致的质量和特征，从而提升模型预测的准确性和鲁棒性。处理过程如下：

（1）调整图像尺寸：使用 opencv 的 resize 接口将输入图像尺寸统一到 160*120；

（2）曝光预测算法：预测出图像中人脸区域是否存在过曝的现象。首先裁剪出人脸区域并转换为 HSV 色彩空间，然后通过滑动窗口计算图像中每个小块的亮度均值，判断是否过曝光，如果过曝光的区域比例超过阈值，就对图像进行动态范围压缩处理，以降低亮度，并将处理后的人脸区域复制回原图像中。

（3）白平衡算法：该算法是让图像变得更加均衡。首先，将输入图像转换为 LAB 颜色空间；然后，分离 LAB 三个通道，并计算 L 通道的均值；接着，计算 A 和 B 通道的均值和相应的增益系数；通过乘以增益系数来调整 A 和 B 通道的值；最后，将调整后的 LAB 通道合并并转换回 BGR 颜色空间，返回调整后的图像。

2.4.3 实时人脸识别算法

（1）人脸检测：使用 opencv 的 FaceDetectorYN 接口配置模型的参数，并且加载模型，将预处理的图像送入模型进行检测，输出一个二维矩阵 Mat，检测信息如下：

0-1：边界框（bbox）左上角的 x 和 y 坐标。
2-3：边界框的宽度和高度。
4-5：右眼的 x 和 y 坐标（图像中的蓝点）。
6-7：左眼的 x 和 y 坐标（图像中的红点）。
8-9：鼻尖的 x 和 y 坐标（图像中的绿点）。
10-11：右嘴角的 x 和 y 坐标（图像中的粉点）。
12-13：左嘴角的 x 和 y 坐标（图像中的黄点）。
14：人脸检测的得分。

（2）人脸对齐：基于人脸检测五个关键点进行仿射变换人脸对齐，使用 opencv 的 estimateAffine2D()估计最优的仿射变换矩阵，用 warpAffine()对图像进行仿射变换。计算过程如下：

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{bmatrix} = M_A \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad M_A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

（3）人脸特征提取：将经过人脸对齐后的人脸送入特征提取模型，得到人脸特征的 512 维特征向量。
（4）实时视频流人脸识别：经过上述的处理只能线性的处理一张图像，要做到实时的图像处理，使用 c++的多线程技术，让人脸检测和人脸识别并行执行，同时使用条件变量和 loonginx 系统的信号机制解决线程间同步的问题，这样不仅充分利用了 2K1000LA 处理器的多核优势，还极大减轻了处理器运行神经网络的负担。为了进一步加快 CPU 的执行速度，利用缓存技术，预先将数据库中的人脸特征信息提取到内存，这样避免了 CPU 在实时人脸识别的过程中要同时提取两张图像的人脸特征，只牺牲了少部分内存，但速度却能够提升百分之五十。
（5）人脸比对：将输入人脸通过人脸特征提取模型后得到的 512 维特征向量进行欧几里得距离的对比，通过计算两张图像的空间距离判断是否为同一个人。计算公式如下：

$$dist(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2.5 UI 设计

2.5.1 客户端 UI 设计

图 2.19 客户端 UI 布局

UI 主窗口继承了 Qt 中的 Widget 父类，opencv 通过将摄像头捕获的图像数据转化成二维矩阵 Mat,再转化成 QImage 类进行实时视频流的显示。显示的时间信息包括日期和时间，使用 QTime 工具类获取当前日期和时间，并使用 Qlabel 进行显示。滚动文本提示使用了 QPropertyAnimation 类，通过设置文本显示的起始坐标和结束坐标，实现滚动播放的功能；右侧的用户信息侧边栏显示考勤成功的用户信息（姓名，学院，头像等信息）。

2.5.2 服务器 UI 设计

图 2.20 服务器登录界面 UI 设计

表单包括账号和密码，点击连接提交表单信息。

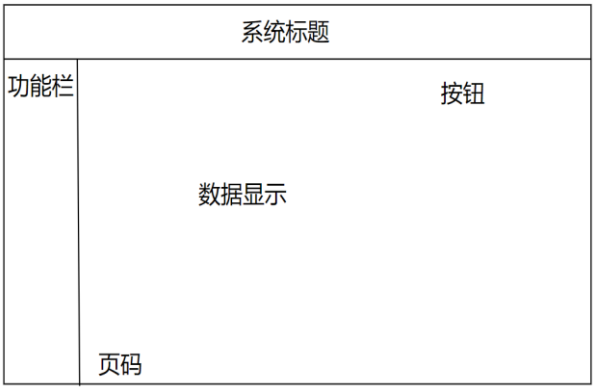


图 2.21 主界面 UI 设计

左侧功能栏使用ListView控件布局，按钮使用Qbutton，单击信号通过connect函数绑定到槽函数，使用 QTableView 以表格的形式直观的显示数据。

2.6 数据库设计

（1） 需求分析：将用户姓名，院系，照片等不易改变且重要的信息存入数据库

（2） 创建用户信息表：姓名和院系字段采用 TEXT 属性，照片信息比较特殊，采用 BLOB 字段以二进制的方式进行存储。

（3） 测试数据库：对数据库进行增删查改的测试，确保数据库能正常使用。

2.7 通信协议设计

（1）传输层协议采用 TCP 协议，TCP 协议是一种面向连接的、可靠的、基于字节流的通信协议。每发出一个数据包都要求确认，如果有一个数据包丢失，就收不到确认，发送方就必须重发这个数据包。为了保证传输的可靠性，TCP 协议在 UDP 基础之上建立了三次对话的确认机制，即在正式收发数据前，必须和对方建立可靠的连接。

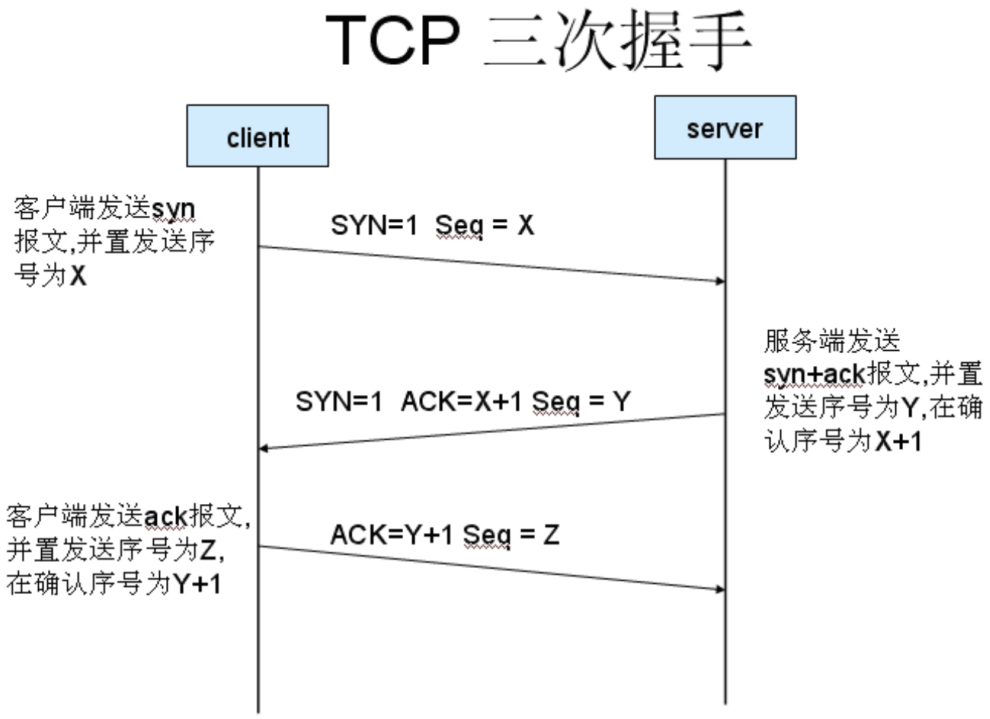


图 2.22 TCP 协议三次握手连接示意图

第一次握手：客户端发送一个 TCP 的 SYN 标志位置 1 的包指明客户打算连接的服务器的端口，以及初始序号 X,保存在包头的序列号(Sequence Number)字段里。

第二次握手： 服务器发回确认包(ACK)应答。即 SYN 标志位和 ACK 标志位均为 1 同时，将确认序号(Acknowledgement Number)设置为客户的 I S N 加 1 以.即 X+1。

第三次握手. 客户端再次发送确认包(ACK) SYN 标志位为 0,ACK 标志位为 1.并且把服务器发来 ACK 的序号字段+1,放在确定字段中发送给对方.并且在数据段放写 ISN 的+1。

（2）应用层协议:在客户端和服务器建立可靠的连接之后,发送的数据包括包头和数据包两部分，包头包括发送数据内容的大小和命令的种类，数据包则是包含发送方真正要发送的数据。通过这样的设计，在发送方发送数据包之后，接手方首先收到包头，解析包头，获取所要接收的数据的大小和要执行的命令种类，在将数据接收完毕之后，开始执行对应的指令。处理流程如下：

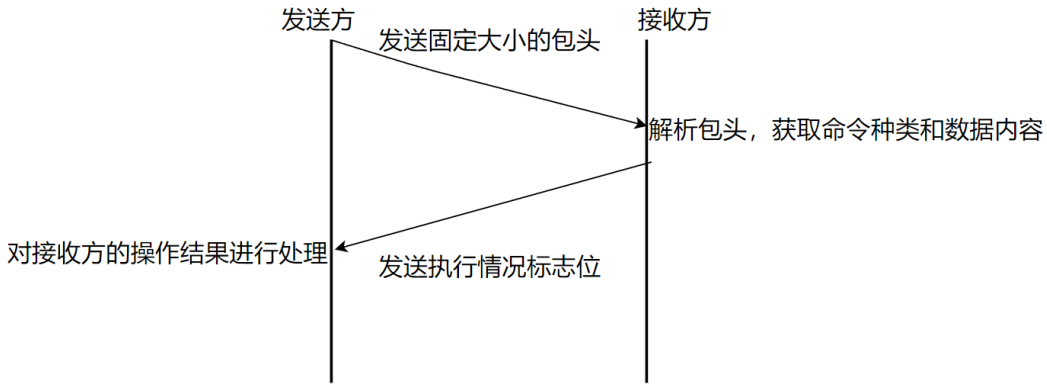


图 2.23 应用层传输数据示意图

（3）命令类型的设计：在通信过程中，为了确保数据传输的有效性和操作的准确性，通常会将命令类型和数据内容分隔开。这样设计的优势在于能够明确区分发送的数据和需要执行的操作，减少理解上的歧义。命令类型用于指示接收方需要执行的具体操作，例如后台程序向客户端发送考勤时间等。具体命令格式如下：

```
enum MessageType {
    SHOW_USERS = 0,             // 显示用户列表
    ADD_USER,                   // 添加用户
    MODIFY_USER_NAME_DEPT,      // 修改用户姓名和部门
    MODIFY_USER_IMGDATA,        // 修改用户图像数据
    DELETE_USER,                // 删除用户

    SET_TIME,                   // 设置时间

    SHOW_USERS_RES,             // 显示用户列表的响应
    ADD_USER_RES,               // 添加用户的响应
    MODIFY_USER_RES,            // 修改用户的响应
    DELETE_USER_RES,            // 删除用户的响应

    SHOW_ATTEND,                // 显示考勤信息
    SHOW_ATTEND_RES,            // 显示考勤信息的响应

    OVER,                       // 操作完成
    FAILURE                     // 操作失败
};
```

2.8 语音播报模块

先通过串口连接到上位机下载需要播放的音频文件，再将 BY8301-16P 芯片的 SPK1 引脚和 SPK2 引脚连接到喇叭两端，VCC 和 GND 连接到串口供电模块的 5V 和 GND，1 号引脚连接到开发板的第 61 个 GPIO 口，基本配置完成。面向对象开发过程如下：

（1） 打开 GPIO 芯片

（2） 请求 GPIO61 对象，设置为推挽输出模式

（3） 给 GPIO61 下降沿触发语音播报功能

第三部分　完成情况及性能参数

3.1 整体介绍

系统整体如下：主要硬件包括龙芯派二代，usb 摄像头，HDMI 显示屏，BY8301-16P 语音模块

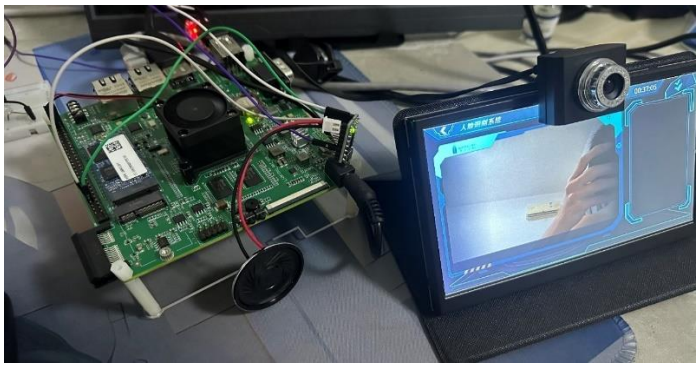


图 3.1 系统整体

3.2 工程成果

3.2.1 软件成果：

图 3.2 客户端 UI

图 3.3 后台管理程序登录界面

图 3.4 用户管理界面

图 3.5 考勤管理界面

3.3 特性成果

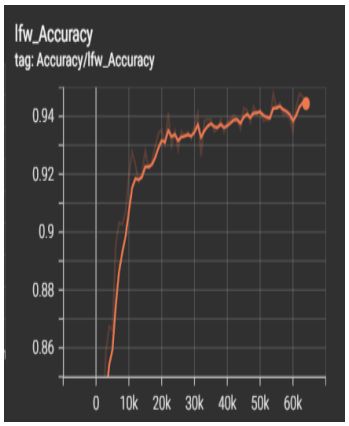


图 3.6 模型精确度变化曲线

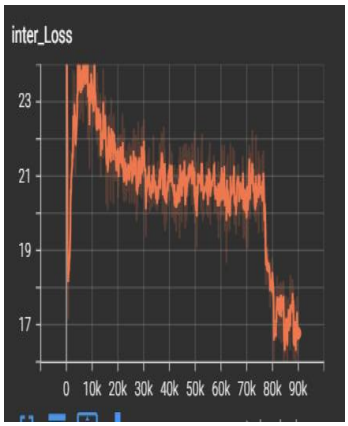


图 3.7 类间损失曲线

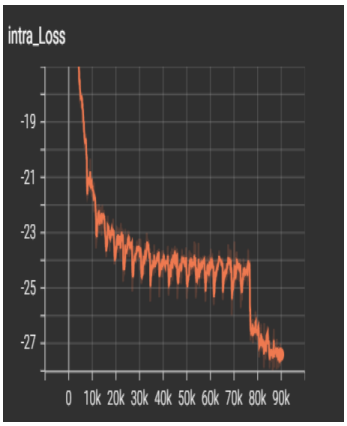


图 3.8 类内损失曲线

模型精确度对比

	将bottleneck模块换成sandglass模块的改进网	换成sam优化器后的再改进网络
准确度	0.755(0.755)	0.788(0.788)
intra_loss	-10.1233(-10.0200)	-9.7538(-9.7024)
inter_loss	17.5779(19.0992)	17.7100(18.6657)
intra_classimilarity	0.1582(0.1566)	0.1524(0.1516)
inter_classimilarity	0.0544(0.0537)	0.0468(0.0463)

最终实测，模型最后识别准确度在 99.1%左右

模型在部署在龙芯派上推理速度

处理器2K1000LA 输入图像160*120	mobilfacenet原网络	mobilfacenet原网络（sandglass模块）
推理速度	365ms	150ms

第四部分 总结

4.1 可扩展之处

（1）本次训练人脸识别神经网络时用到的训练集是欧美人脸，所以对亚洲人脸的识别精度会有所下降，所以可以用采集更广泛地区的人脸作为训练集来提高神经网络的泛化能力

（2）本次训练受制于设备性能，并未完全训练到底，但依旧获得了很好的效果，所以可以在有能力时将之完成训练以达到完整效果。

（3）本次的考勤签到系统可以加入活体检测模块，从而有效防止代签等人为规避行为。

4.2 心得体会

本人有幸参加第七届全国嵌入式大学生竞赛，想来也是自己大学最后一次参加学科竞赛，在做这个项目之前，首先谈谈我为什么会选择嵌入式人脸识别这个方向；在四月份中下旬报名比赛的时候，当时我掌握关于嵌入式方面的知识软件方面有 c 语言和 Linux 操作系统，以及一些开发常用的交叉编译工具(makefile 等)的使用,硬件方面会一些简单的 MCU 开发，如常见的 STM32 系列单片机的开发。由于当时的我已经做了关于关于 c 语言和 linux 的项目,并且时间临近秋招，我急需一个有分量的项目来为自己加分，所以这次比赛我选择了嵌入式+人工智能的赛道。

虽说人脸识别技术在人工智能领域一直都是比较热门的话题，但对于初学者且毫无相关背景知识的我来说，难度非常之大。光是相关背景知识的了解和学习就需要花费我大量时间和精力，落地实现和部署到目标平台运行更是天方夜谭。在冷静分析过后，我决定先做难度较小的软件部分，将整个考勤系统软件先大致实现，但后来实现的过程也不轻松，软件体量不小，包括人脸识别核心处理逻辑，考勤客户端和服务器的实现和通信，这一阶段，包括主要技术框架 c++,Qt 的学习，以及代码开发，调试和优化总共花了一个月时间左右。 到了六月份，此时的我已经不再像最开始那样对深度学习技术茫然，我先是花少量时间了解深度学习的概念和简单实现，再去深入了解深度学习下的人脸识别原理和实现，到选定合适的深度学习框架，主干网络，损失函数，最后甚至优化原有网络。最后能完成这个项目我是非常自豪的，两个月的时间能从 0 到 1 自主实现这个无感人脸识别考勤系统，并且效果还不错，真的是费了自己太多心血。

第五部分 参考文献

[1] Wu W , Peng H , Yu S .YuNet: A Tiny Millisecond-level Face Detector[J].机器智能研究:英文版, 2023(005):020.

[2]胡佳荣,孟文,赵晶晶.基于改进 MobileFaceNet 的人脸识别方法[J].半导体光电, 2022, 43(1):164.DOI:10.16818/j.issn1001-5868.2021110204.

[3] Zhong Y , Deng W , Hu J ,et al.SFace: Sigmoid-Constrained Hypersphere Loss for Robust Face Recognition[J].IEEE Transactions on Image Processing, 2021, PP(99):1-1.DOI:10.1109/TIP.2020.3048632.

[4] Guo L , Bai H , Zhao Y .A Lightweight and Robust Face Recognition Network on Noisy Condition[C]//2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC).2019.DOI:10.1109/APSIPAASC47483.2019.9023149.

[5] Chen S , Liu Y , Gao X ,et al.MobileFaceNets: Efficient CNNs for Accurate Real-time Face Verification on Mobile Devices[J]. 2018.DOI:10.48550/arXiv.1804.07573.

第六部分 附录

主干网络代码

```
class MobileFaceNet(Module):
    def __init__(self, embedding_size):
        super(MobileFaceNet, self).__init__()
        self.conv_1 = Conv_block(3, 64, kernel=(3, 3), stride=(2, 2), padding=(1, 1))
        self.conv_2_dw = Conv_block(64, 64, kernel=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
        self.dconv_23 = Glass_Wise(64, 64, kernel=(3, 3), stride=(2, 2), padding=(1, 1), groups= 16)
        self.res_3 = Residual(64, num_block=4, groups=16, kernel=(3, 3), stride=(1, 1), padding=(1, 1))
        self.dconv_34 = Glass_Wise(64, 128, kernel=(3, 3), stride=(2, 2), padding=(1, 1), groups=16)
        self.res_4 = Residual(128, num_block=6, groups=16, kernel=(3, 3), stride=(1, 1), padding=(1, 1))
        self.dconv_45 = Glass_Wise(128, 128, kernel=(3, 3), stride=(2, 2), padding=(1, 1), groups=16)
        self.res_5 = Residual(128, num_block=2, groups=16, kernel=(3, 3), stride=(1, 1), padding=(1, 1))
        self.conv_6sep = Conv_block(128, 512, kernel=(1, 1), stride=(1, 1), padding=(0, 0))
        self.conv_6dw7_7 = Linear_block(512, 512, groups=512, kernel=(7,7), stride=(1, 1), padding=(0, 0))
        self.conv_6_flatten = Flatten()  # 定义展平层
        self.pre_fc1 = Linear(512, embedding_size)  # 定义全连接层
        self.fc1 = BatchNorm1d(embedding_size, eps=2e-5)  # 定义批归一化层
    def forward(self, x):
        x = x - 127.5  # 图像归一化，减去均值
        x = x * 0.078125  # 图像归一化，乘以系数
        out = self.conv_1(x)  # 通过第一个卷积块
        out = self.conv_2_dw(out)  # 通过第二个卷积块（深度卷积）
        out = self.dconv_23(out)  # 通过第一个深度可分离卷积块
        out = self.res_3(out)  # 通过第一个残差块
        out = self.dconv_34(out)  # 通过第二个深度可分离卷积块
```

```
        out = self.res_4(out)  # 通过第二个残差块
        out = self.dconv_45(out)  # 通过第三个深度可分离卷积块
        out = self.res_5(out)  # 通过第三个残差块
        out = self.conv_6sep(out)  # 通过第四个卷积块
        out = self.conv_6dw7_7(out)  # 通过第五个线性卷积块
        out = self.conv_6_flatten(out)  # 展平成二维张量
        out = self.pre_fc1(out)  # 通过全连接层
```

SFace 损失代码

```
class SFaceLoss(nn.Module):  # 定义 SFaceLoss 类，继承自 nn.Module
    #输入特征是网络的
    def __init__(self, in_features, out_features, device_id, s=64.0, k=80.0,
a=0.80, b=1.23):
        super(SFaceLoss, self).__init__()  # 调用父类的构造函数
        self.in_features = in_features  # 输入特征数
        self.out_features = out_features  # 输出特征数
        self.device_id = device_id  # 设备 ID
        self.s = s  # 缩放因子
        self.k = k  # 控制因子
        self.a = a  # 角度调整因子
        self.b = b  # 角度调整因子
        self.weight = Parameter(torch.FloatTensor(out_features, in_features))  #
定义权重参数
        xavier_normal_(self.weight, gain=2, mode='out')  # 使用 Xavier 正态
初始化权重

    def forward(self, input, label):  # 定义前向传播函数
        if self.device_id is None:  # 如果设备 ID 为空
            cosine = F.linear(F.normalize(input), F.normalize(self.weight))  # 计
算余弦相似度
        else:
            x = input  # 输入特征
            sub_weights = torch.chunk(self.weight, len(self.device_id),
dim=0)  # 将权重按设备数量拆分
            temp_x = x.cuda(self.device_id[0])  # 将输入特征转移到第一个设
备
            weight = sub_weights[0].cuda(self.device_id[0])  # 将权重转移到第
```

```
一个设备
            cosine = F.linear(F.normalize(temp_x), F.normalize(weight))  # 计算余弦相似度

            for i in range(1, len(self.device_id)):  # 遍历剩余设备
                temp_x = x.cuda(self.device_id[i])  # 将输入特征转移到当前设备
                weight = sub_weights[i].cuda(self.device_id[i])  # 将权重转移到当前设备
                cosine = torch.cat((cosine, F.linear(F.normalize(temp_x), F.normalize(weight)).cuda(self.device_id[0])), dim=1)  # 计算余弦相似度并拼接

        output = cosine * self.s  # 余弦相似度乘以缩放因子

        one_hot = torch.zeros(cosine.size())  # 创建全零张量
        if self.device_id is not None:
            one_hot = one_hot.cuda(self.device_id[0])  # 将全零张量转移到第一个设备
        one_hot.scatter_(1, label.view(-1, 1), 1)  # 将 one-hot 编码标签填入张量

        zero_hot = torch.ones(cosine.size())  # 创建全一张量
        if self.device_id is not None:
            zero_hot = zero_hot.cuda(self.device_id[0])  # 将全一张量转移到第一个设备
        zero_hot.scatter_(1, label.view(-1, 1), 0)  # 将 one-hot 编码标签位置填零

        WyiX = torch.sum(one_hot * output, 1)  # 计算每个样本对应类别的加权和
        with torch.no_grad():  # 在不计算梯度的上下文中
            theta_yi = torch.acos(WyiX / self.s)  # 计算角度
            weight_yi = 1.0 / (1.0 + torch.exp(-self.k * (theta_yi - self.a)))  # 计算权重
        intra_loss = - weight_yi * WyiX  # 计算类内损失

        Wj = zero_hot * output  # 计算其他类别的加权和
        with torch.no_grad():  # 在不计算梯度的上下文中
            theta_j = torch.acos(Wj / self.s)  # 计算角度
```

```
            weight_j = 1.0 / (1.0 + torch.exp(self.k * (theta_j - self.b)))  # 计算权重
        inter_loss = torch.sum(weight_j * Wj, 1)  # 计算类间损失

        loss = intra_loss.mean() + inter_loss.mean()  # 总损失为类内损失和类间损失的平均值
        Wyi_s = WyiX / self.s  # 归一化类内相似度
        Wj_s = Wj / self.s  # 归一化类间相似度

        return output, loss, intra_loss.mean(), inter_loss.mean(), Wyi_s.mean(), Wj_s.mean()  # 返回输出、损失、类内损失、类间损失、归一化类内相似度、归一化类间相似度
```

SAM 优化器代码

```
class SAM(torch.optim.Optimizer):  # SAM 类继承自 torch.optim.Optimizer
    def __init__(self, params, base_optimizer, rho=0.05, **kwargs):
        assert rho >= 0.0, f"Invalid rho, should be non-negative: {rho}"
        defaults = dict(rho=rho, **kwargs)
        super(SAM, self).__init__(params, defaults)  # 调用父类的初始化方法
        self.base_optimizer = base_optimizer(self.param_groups, **kwargs)  # 实例化基础优化器
        self.param_groups = self.base_optimizer.param_groups  # 获取基础优化器的参数组

    @torch.no_grad()  # 不计算梯度
    def first_step(self, zero_grad=False):  # 第一阶段方法，接受参数 zero_grad 决定是否清零梯度
```

```
        grad_norm = self._grad_norm()  # 计算梯度范数
        for group in self.param_groups:  # 遍历每个参数组
            scale = group["rho"] / (grad_norm + 1e-12)  # 按比例缩放梯度
            for p in group["params"]:  # 遍历每个参数
                if p.grad is None:  # 如果梯度为 None，跳过
                    continue
                e_w = p.grad * scale.to(p)  # 计算 e_w
                p.add_(e_w)  # 将 e_w 加到参数上，朝局部最大值方向前进
                self.state[p]["e_w"] = e_w  # 将 e_w 保存到状态字典中
        if zero_grad:  # 如果 zero_grad 为 True
            self.zero_grad()  # 清零梯度

    @torch.no_grad()  # 不计算梯度
    def second_step(self, zero_grad=False):  # 第二阶段方法，接受参数 zero_grad 决定是否清零梯度
        for group in self.param_groups:  # 遍历每个参数组
            for p in group["params"]:  # 遍历每个参数
                if p.grad is None:  # 如果梯度为 None，跳过
                    continue
                p.sub_(self.state[p]["e_w"])  # 从参数中减去在第一步中加上的 e_w，回到初始位置
        self.base_optimizer.step()  # 执行基础优化器的更新
        if zero_grad:  # 如果 zero_grad 为 True
            self.zero_grad()  # 清零梯度

    def _grad_norm(self):  # 计算梯度范数的方法
        shared_device = self.param_groups[0]["params"][0].device  # 获取设备信息，确保所有操作在同一设备上进行，适用于模型并行化
        norm = torch.norm(  # 计算所有参数梯度的二范数
            torch.stack([
                p.grad.norm(p=2).to(shared_device)  # 计算每个参数梯度的二范数，并将其放到共享设备上
                for group in self.param_groups for p in group["params"]
                if p.grad is not None  # 仅计算非空梯度的参数
            ]),
            p=2
        )
        return norm  # 返回梯度范数
```

图像预处理算法

```
//对图像进行白平衡
cv::Mat applyWhiteBalance(cv::Mat& image){
    // 将图像转换为 LAB 颜色空间
    cv::Mat lab_image;
    cvtColor(image, lab_image, cv::COLOR_BGR2Lab);

    // 分离 LAB 通道
    std::vector<Mat> lab_planes(3);
    split(lab_image, lab_planes);

    // 计算 L 通道的均值
    Scalar l_mean = mean(lab_planes[0]);

    // 计算 A 和 B 通道的均值
    Scalar a_mean = mean(lab_planes[1]);
    Scalar b_mean = mean(lab_planes[2]);

    // 计算增益系数
    float a_gain = l_mean[0] / a_mean[0];
    float b_gain = l_mean[0] / b_mean[0];

    // 调整 A 和 B 通道
    lab_planes[1] = lab_planes[1] * a_gain;
    lab_planes[2] = lab_planes[2] * b_gain;

    // 合并 LAB 通道
    merge(lab_planes, lab_image);

    // 转换回 BGR 颜色空间
    Mat result;
    cvtColor(lab_image, result, COLOR_Lab2BGR);

    return result;
}
```

```
// 亮度调整函数
cv::Mat adjustBrightness(cv::Mat& image, double alpha, int beta)
{
    Mat new_image = Mat::zeros(image.size(), image.type());

    for (int y = 0; y < image.rows; y++) {
        for (int x = 0; x < image.cols; x++) {
            for (int c = 0; c < 3; c++) {
                new_image.at<Vec3b>(y,x)[c] =
                    saturate_cast<uchar>(alpha * image.at<Vec3b>(y,x)[c] +
beta);
            }
        }
    }

    return new_image;
}

// 光照强度计算并调整函数
cv::Mat adjustLighting(cv::Mat& image)
{
    Scalar mean_val = mean(image);
    double alpha;
    int beta;

    if (mean_val[0] > 150) { // 光照过强
        alpha = 0.5; // 降低亮度
        beta = -30;
    } else if (mean_val[0] < 100) { // 光照过弱
        alpha = 1.5; // 增强亮度
        beta = 30;
    } else {
        alpha = 1.0; // 保持原亮度
        beta = 0;
    }
    return adjustBrightness(image, alpha, beta);
}
//对需要检测的图像进行预处理
cv::Mat prepro_img(cv::Mat& image)
{
```

```
    cv::Mat temp;
    cv::Mat res;
    temp = adjustLighting(image);//先进行亮度调整
    res = applyWhiteBalance(temp);//在进行白平衡处理

    return res;
}
```

人脸识别核心逻辑代码

```
//实时人脸识别代码核心逻辑块
void FaceDetect(QLabel *label,Widget &widget)
{
    cv::VideoCapture cap = GetCapture();
    cv::Mat frame;
    cv::Mat temp;
    auto tick_meter = cv::TickMeter();

    int flag = 30;
    // 开启人脸识别线程
    thread compare_thread(FeatureCompare,ref(widget));
    while (true)
    {
        bool has_frame = cap.read(frame);
        if (!has_frame)
        {
            std::cout << "No frames grabbed! Exiting ...\n"<<std::endl;
            break;
        }

        //处理视频帧
        tick_meter.start();
        ProcessFrame(frame,img_size);
        tick_meter.stop();
        //适当条件唤醒人脸识别线程
        WakeFeatureCompare(flag);
```

```
        //显示图片
        DisplayFrame(label, frame,tick_meter.getFPS());

        tick_meter.reset();
    }

    compare_thread.join();
}
```

客户端 UI 代码

```
void Widget::startUi()
{
    this->setFixedSize(width, height);
    //初始化考勤时间，默认无效
    startTime.setHMS(0,0,0);
    endTime.setHMS(0,0,0);

    label = new QLabel(this); // 创建主画布标签
    img_label = new QLabel(label); // 创建图像标签，并设置为主画布标签的子控件
    video_label = new QLabel(label); // 创建视频标签，并设置为主画布标签的子控件
    sideBar_label = new QLabel(label);//侧边栏标签
    timer = new QTimer;//定时时钟
    timer->setInterval(1000);
    timer->start();//开启时钟

    // 创建标签用于显示时间和日期
    timeLabel = new QLabel(this);
    timeLabel->setStyleSheet("color: white"); // 设置字体颜色为白色
    timeLabel->setFont(QFont("Arial", 18));
    timeLabel->setGeometry(600, 5, 250, 30); // 设置标签的位置和大小
```

```
    // 创建滚动文本标签
    scrollingLabel = new QLabel(this);
    scrollingLabel->setGeometry(50, 490, 400, 30);
    scrollingLabel->setStyleSheet("color: white");
    scrollingLabel->setFont(QFont("Arial", 16));
    scrollingLabel->setText("请提醒管理员设置考勤时间");
    scrollingLabel->show();

    // 创建滚动动画
    scrollingAnimation = new QPropertyAnimation(scrollingLabel, "pos", this);
    scrollingAnimation->setDuration(7000); // 设置滚动时间为 10 秒
    scrollingAnimation->setStartValue(QPoint(50, 490));
    scrollingAnimation->setEndValue(QPoint(480, 490));
    scrollingAnimation->setLoopCount(3); // 设置循环次数为无限循环

   // 开始滚动动画
   scrollingAnimation->start();

    img_label->setGeometry(20, 20, 680, 560); // 设置图像标签的位置和大小
    video_label->setGeometry(40, 20, 680, 560); // 设置视频标签的位置和大
小
    sideBar_label->setGeometry(710,55,300,480);//侧边栏标签大小

    background.load("/home/hy/code/static/background.png"); // 加载背景图
片
    img_bgr.load("/home/hy/code/static/img_bgp.png"); // 加载图像背景图片
    side_bar.load("/home/hy/code/static/sidebar.png");//侧边栏图像

    background = background.scaled(width, height, Qt::IgnoreAspectRatio,
Qt::SmoothTransformation); // 调整背景图片大小
    img_bgr = img_bgr.scaled(680, 480, Qt::IgnoreAspectRatio,
Qt::SmoothTransformation); // 调整图像背景图片大小
    side_bar = side_bar.scaled(300,480,Qt::IgnoreAspectRatio,
Qt::SmoothTransformation);//调整侧边栏大小

    img_label->setPixmap(QPixmap::fromImage(img_bgr)); // 设置图像标签
的图片
    label->setPixmap(QPixmap::fromImage(background)); // 设置主画布标签
的背景图片
    sideBar_label->setPixmap(QPixmap::fromImage(side_bar));//侧边栏图片
```

```
    video_label->lower(); //  确保视频标签显示在图像标签下方
    img_label->raise(); //  确保图像标签显示在视频标签上方
    sideBar_label->raise();
    scrollingLabel->raise();
}
```

服务器端代码

```
backstorg::backstorg(QTcpSocket *s,Widget* widget,QWidget *parent) :
    QWidget(parent),
    ui(new Ui::backstorg),
    server(new QTcpServer),
    socket(s),//通信套接字
    widget(widget),
    model(new QStandardItemModel),
    pagination(new Pagination),
    pageLineEdit(new QLineEdit),
    totalPagesDisplay(new QLabel),
    pageLayout(new QHBoxLayout),
    prevButton(new QPushButton("上一页")),
    nextButton(new QPushButton("下一页")),
    totalPagesLabel(new QLabel("页码总数")),
    timer(new QTimer)
{
    ui->setupUi(this);
    //设置标题
    setTile();
    //设置左侧列表
    setList();
    //设置中央表格样式
    setUpTableView();
    //设置按钮样式
    setButton();
    //开启服务器
```

```
    startServer();

    initUserManageMent();//登录后默认进入用户管理界面

    connect(ui->listView, &QListView::clicked, this,
&backstorg::onListItemClicked);//点击左侧列表发送信号
    connect(this, &backstorg::itemClicked, this,
&backstorg::UsersManagement);//根据对应点击的 List 信号做出响应

connect(ui->addUserButton,&QPushButton::clicked,this,&backstorg::onAddUs
erClicked);//增加用户按钮

connect(ui->deleteUserButton,&QPushButton::clicked,this,&backstorg::onDele
teUserClicked);//删除用户按钮

connect(socket,&QTcpSocket::disconnected,this,&backstorg::backLogin);//断
开连接

connect(ui->refreshButton,&QPushButton::clicked,this,&backstorg::onUserRefr
eshButton);//用户刷新按钮

connect(ui->exitButton,&QPushButton::clicked,this,&backstorg::onExitButton);

connect(ui->modifyUserButton,&QPushButton::clicked,this,&backstorg::onMo
difyUserCilcked);//删除用户按钮
    connect(pagination, &Pagination::pageChanged, this,
&backstorg::updateTableView);// 连接 Pagination 的信号
    connect(timer, &QTimer::timeout, this, &backstorg::updateTimeLabel);//
定时器显示时间

connect(ui->confirmButton,&QPushButton::clicked,this,&backstorg::pushAtten
denceTime);//提交考勤时间

    // 设置定时器的超时时间为 1 秒
    timer->setInterval(1000);
    timer->start();
```