苏州科技大学

Python 程序设计课程实践报告

——智能门卫管理系统
天枢安防 TIANSHU SECURITY

小组任务分配情况

学号	姓名	承担的任务	成绩
	杨浩正	前后端开发、算法设计	
	谢雨欣	项目分析、数据库管理	
	沈子妍	框架设计、项目测试	

专业年级　　人工智能 2411

指导教师　　华　泽

实践地点　　电子信息与智能化实验中心

目录

第 1 章 任务描述

1.1. 选题

智能门卫管理系统

1.2. 项目背景

随着人工智能技术的快速发展，计算机视觉和人脸识别技术已经日趋成熟，并在安防、门禁、考勤等领域得到广泛应用。传统的门禁管理系统通常依赖于门禁卡、密码等方式进行身份验证，存在易遗失、易被盗用等安全隐患。基于人脸识别的智能门禁系统能够实现非接触式身份验证，具有便捷、安全、高效的特点。

本项目旨在设计并实现一套基于深度学习的人脸识别智能门卫管理系统——"天枢安防"，为校园、企业、社区等场所提供智能化的门禁管理解决方案，提升安全管理水平和用户体验。

1.3. 任务描述

针对出入各类治安卡口、重要出入口人员进行监控与管理的系统，是视频分析、运动跟踪、人脸检测和识别技术在视频监控领域的全新综合应用。前端摄像机对经过卡口的人员进行人脸抓拍，抓拍到的人脸图片通过计算机网络传输到对比服务器及监控中心的数据库进行数据存储，并与人脸库进行实时比对。要求系统集高清人脸图像的抓拍、传输、存储，人脸特征的提取和分析识别、自动报警和联网布控等诸多功能于一身，并具有强大的查询、检索等后台数据处理功能及强大的通信、联网功能，可广泛应用于重要关卡的行人监控。

第 2 章 系统需求分析

2.1. 系统主要功能

2.1.1. 核心功能

1. 人脸识别：系统能够实时捕获视频流中的人脸，并与数据库中已录入的人脸进行匹配，确定身份
2. 人脸录入：支持通过上传照片或摄像头实时拍摄的方式录入新的人脸信息
3. 人脸管理：提供人脸信息的查看、删除、更新等管理功能
4. 权限控制：根据不同用户的身份和权限，控制其通行权限
5. 识别记录：记录每次识别的结果，包括时间、地点、识别结果等信息

2.1.2. 扩展功能

1. 访客管理：支持临时访客的人脸录入和权限管理
2. 异常报警：当检测到未授权人员尝试通行时，触发报警机制
3. 数据统计：统计分析通行记录，生成报表
4. 多模态认证：结合人脸识别与其他认证方式（如密码、指纹等）进行多因素认证
5. 移动端支持：开发移动端应用，实现远程管理和控制

2.2. 需求分析

2.2.1. 用户角色与需求

一、用户端

- 住户

  快速通行：通过人脸识别实现无感通行，无需刷卡或输入密码，提升出入效率和体验。

  访客授权：可通过手机端或 Web 端为亲友临时授权访客码，实现远程放行。

  通行记录查询：可随时查看本人及授权访客的历史通行记录，保障家庭安全。
- 访客

  临时通行权限申请：可通过门卫登记、住户远程授权等方式获取一次性或限时访客码，实现临时通行。

自助登记：支持访客自助登记信息，提升通行效率。
身份核验：通过人脸识别或访客码核验身份，确保通行安全。

二、管理端

- 管理人员
  人员权限管理：可新增、编辑、删除住户信息，批量导入住户数据；为访客设置临时权限及有效时段。
  数据统计与查询：支持通行记录、异常事件日志、访客流量等多维度数据统计与可视化分析。
  系统后台管理：包括设备状态监控、权限分配、数据导出、系统参数配置等功能，保障系统稳定运行。
  多级权限分配：可为不同管理人员分配不同操作权限，实现分级管理。

- 门卫/安保人员
  访客核验：支持访客身份证扫描、人证比对，提升访客管理的安全性和效率。
  应急处理：遇到设备故障或特殊情况时，可手动放行或临时管控设备，保障现场秩序。
  异常事件处理：实时接收系统报警信息，对异常通行、黑名单人员等情况进行快速响应。

三. 端口分工说明

- **用户端（住户/访客）**：主要通过 Web 客户端或移动端进行人脸识别通行、访客申请、记录查询等操作，界面简洁、操作便捷，注重体验和安全。
-
- **管理端（管理人员/门卫）**：通过管理后台进行权限配置、数据管理、设备监控和应急处理，功能全面，注重系统性和可控性。

2.2.2.核心功能需求

1.身份认证与通行管理
多种认证方式：人脸识别（主流，需高准确率）、IC 卡/二维码、手机 APP 远程开门、密码输入（备用）。

2.访客管理
预约流程：住户通过 APP 发起预约，填写访客信息（姓名、电话、身份证号、到访时间），系统自动生成临时通行码。
现场登记：访客可通过门卫处自助终端或人工登记，身份证扫描+拍照存档，实时同步至住户确认。

3.设备与系统集成

硬件对接：门禁闸机、人脸识别终端、监控摄像头、报警按钮等设备的联动控制。

软件集成：与物业管理系统（如缴费、报修）、安防系统（视频监控、周界报警）、云平台（数据存储、远程管理）对接。

移动端 APP：支持住户、管理人员分别使用的 APP，实现通行、预约、管理功能。

4.数据管理与安全

数据存储：通行记录、访客信息加密存储，支持至少 1 年以上历史数据查询。

隐私保护：人脸识别数据脱敏处理，符合《个人信息保护法》，禁止数据泄露或滥用。

安全审计：系统操作日志（如权限修改、数据导出）留痕，便于追溯。

2.2.3.性能与场景需求

1. 响应速度：人脸识别通行时间≤1 秒，系统后台操作响应≤3 秒。

2. 离线能力：网络故障时，本地存储的权限数据仍可支持通行，网络恢复后自动同

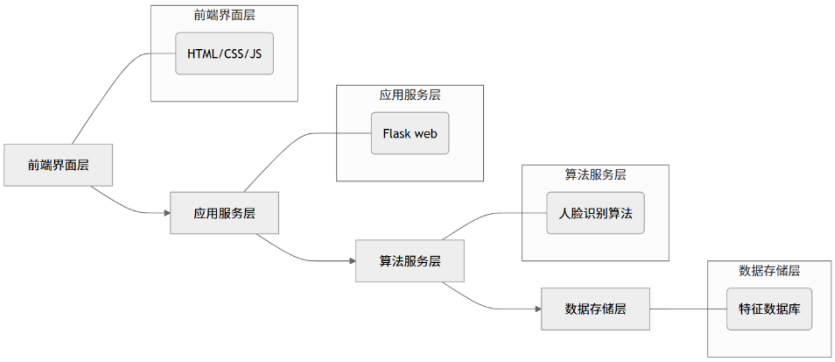
第 3 章 系统设计

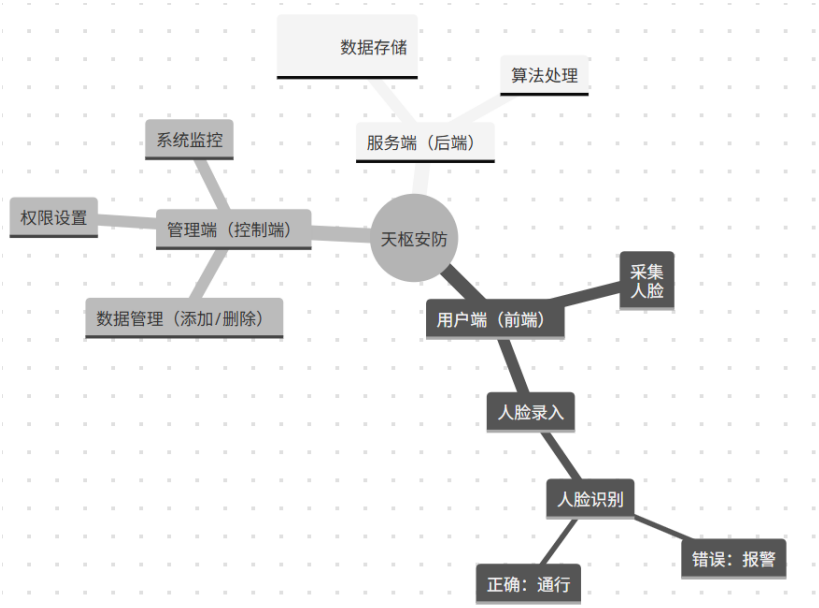
3.1. 总体架构设计

天枢安全智能门卫管理系统采用前后端分离的 B/S 架构，主要由以下几个部分组成：

1. 前端界面层：基于 HTML5、CSS3 和 JavaScript 实现的 Web 界面，提供用户交互功能
2. 应用服务层：基于 Flask 框架实现的 Web 应用服务，处理前端请求，调用算法服务
3. 算法服务层：实现人脸检测、特征提取、身份识别等核心算法
4. 数据存储层：存储人脸特征数据、用户信息、访问记录等数据

系统架构图如下：





3.2. 核心算法设计

3.2.1. 人脸检测算法

系统采用 dlib 的前置人脸检测器，其基于 HOG 特征和线性 SVM 分类器实现。

HOG 特征提取原理：

HOG（方向梯度直方图）特征提取步骤如下：

1. 图像预处理：灰度化、伽马校正
2. 计算图像梯度：x 方向和 y 方向的一阶导数

$$G_x = I(x+1,y) - I(x-1,y) \tag{1}$$

$$G_y = I(x,y+1) - I(x-1,y) \tag{2}$$

3. 计算梯度幅值与方向：

$$G = \sqrt{G_x^2 + G_y^2} \tag{3}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{4}$$

4. 构建单元格方向梯度直方图
5. 块归一化：增强对光照变化的鲁棒性

$$v' = \frac{v}{\sqrt{|v|_2^2 + \epsilon}} \tag{5}$$

其中 v 是向量，ϵ 是小常数。

系统在检测过程中使用了更高精度的参数：

```
faces = self.detector(img_blurred, 2)
```

其中第二个参数表示检测的上采样次数，增加此值可提高检测精度，特别是对于小尺寸人脸。

3.2.2. 人脸关键点检测

系统采用 dlib 的 shape_predictor_68_face_landmarks 模型进行人脸关键点定位，基于级联形状回归算法（Cascade Shape Regression）。

该算法核心思想是学习从当前形状估计到实际形状的回归函数：

$$S_{t+1} = S_t + r_t(I, S_t) \tag{6}$$

其中，S_t 是第 t 次迭代的形状估计，r_t 是回归函数，I 是输入图像。

回归器通过级联方式构建，每一级回归器负责纠正前一级的误差。通过提取局部特征并与形状变形相关联，实现对 68 个面部关键点的精准定位。

3.2.3. 人脸特征提取

系统核心是基于 ResNet 深度学习模型提取的 128 维特征向量。该模型在数百万人脸图像上训练，采用三元组损失函数（Triplet Loss）优化：

$$L = \sum_i [|f(x_i^a) - f(x_i^p)|_2^2 - |f(x_i^a) - f(x_i^n)|_2^2 + \alpha]_+ \tag{7}$$

其中：

- f(x_i^a)是锚样本（Anchor）的特征
- f(x_i^p)是正样本（Positive，同一身份）的特征
- f(x_i^n)是负样本（Negative，不同身份）的特征
- \alpha 是强制间隔

特征提取过程增加了采样次数以提高精度：

```
face_descriptor =
self.face_reco_model.compute_face_descriptor(img_blurred,
shape, 10)
```

第三个参数表示用于提高特征提取质量的采样次数。

3.2.4. 图像预处理优化

为提高识别准确率，系统实现了多步图像预处理：

1. **直方图均衡化**：增强图像对比度

$$CDF(v) = \sum_{i=0}^{v} p(i) \tag{8}$$

$$h(v) = round((L-1) \cdot CDF(v)) \tag{9}$$

其中 L 是灰度级数量，p(i)是灰度级 i 的概率。

2. **高斯模糊**：减少噪声干扰

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{10}$$

系统实现代码：

```
# 直方图均衡化以增强对比度
img_yuv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2YUV)
img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])
img_enhanced = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2RGB)

# 轻度高斯模糊减少噪声
img_blurred = cv2.GaussianBlur(img_enhanced, (3, 3), 0)
```

3.2.5. 特征匹配与决策策略

系统采用多级决策策略判定身份：

1. **特征距离计算**：使用欧氏距离度量特征相似性

$$d(x,y) = |x-y|_2 = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{11}$$

2. **加权投票策略**：当匹配不够明确时使用

- 首先获取前 k 个最接近的匹配
- 计算每个匹配的权重（距离的倒数）

$$w_i = \frac{1}{d_i + \epsilon} \tag{12}$$

- 归一化权重

$$w_i' = \frac{w_i}{\sum_{j=1}^{k} w_j} \tag{13}$$

- 根据权重进行投票，选出得票率最高的身份

$$vote(id) = \sum_{i \in \{i | name_i = id\}} w_i' \tag{14}$$

3. **多阈值决策**：系统采用多级阈值判定

- 当最佳匹配距离小于 0.4 时，直接采用
- 当距离介于 0.4 到 0.55 之间，采用加权投票
- 当得票率超过 0.6 时，判定为成功识别
- 其他情况判定为未知身份

```
# 如果最佳匹配的距离足够小，直接采用
if top_matches[0][0] < 0.4:
    min_distance, person_name = top_matches[0]
    confidence = 1 - min_distance
    # ...
# 如果最佳匹配不够明确，使用加权投票
elif len(top_matches) >= 2 and top_matches[0][0] < 0.55:
    # 计算权重（距离的倒数）
    weights = [1/(d+0.01) for d, _ in top_matches]
    # ...
```

3.2.6. 数据库特征优化

系统对数据库中的人脸特征进行了多项优化：

1. **异常值过滤**：移除偏离度过高的特征

$$threshold = \mu + 1.5 \times \sigma \tag{15}$$

其中\mu 是平均距离，\sigma 是标准差。

2. **聚类表示**：使用 K-means 聚类捕获不同角度的特征

$$\text{minimize} \sum_{i=1}^{k} \sum_{x \in S_i} |x - \mu_i|^2 \tag{16}$$

其中 k 是聚类数，S_i 是第 i 个簇，\mu_i 是第 i 个簇的中心。

3. **特征归一化**：确保特征向量的范数为 1

$$\vec{v}' = \frac{\vec{v}}{|\vec{v}|} \tag{17}$$

3.2.7. 大规模检索优化

当特征数量较大时，系统会自动启用 FAISS（Facebook AI Similarity Search）加速特征匹配：

```
if len(self.face_features) > 100:
    try:
        if FAISS_AVAILABLE:
            # 将特征转换为适合 FAISS 的格式
            features_array =
np.array(self.face_features).astype('float32')

            # 创建索引
            self.index =
faiss.IndexFlatL2(features_array.shape[1])
            self.index.add(features_array)
```

FAISS 基于向量量化和索引技术，将搜索复杂度从 O(Nd)降低到 O(d\log N)，其中 N 是特征数量，d 是特征维度。

3.3. 数据库设计

系统采用文件系统存储人脸数据，每个人的人脸数据存储在以其姓名命名的文件夹中。每个人脸图像保存为 JPEG 格式，并附带一个 JSON 格式的特征描述文件，记录图像的元数据和质量评估结果。

特征描述文件的结构如下：

```
{
  "timestamp": "20240520_143025",
  "face_rect": [120, 80, 240, 200],
  "quality": {
    "size": [120, 120],
    "eye_aspect_ratio": 0.25,
    "face_angle": 2.5
  }
}
```

3.4. 接口设计

系统提供以下主要 API 接口：

1. /api/recognize：识别上传的图片中的人脸
2. /api/recognize_frame：识别视频帧中的人脸
3. /api/create_face：创建新的人脸文件夹
4. /api/add_face_image：添加人脸图像到数据库
5. /api/add_face_from_camera：从摄像头捕获的图像添加人脸
6. /api/get_face_database：获取人脸数据库信息
7. /api/delete_face：删除人脸数据

3.5. 用户界面设计

系统包含以下主要页面：

1. **首页**：系统概览和功能导航
2. **人脸识别页**：实时人脸识别界面
3. **人脸录入页**：新增人脸信息界面
4. **人脸管理页**：管理已录入人脸的界面
5. **更多功能页**：展示系统扩展功能的界面

界面设计遵循简洁、直观、易用的原则，采用响应式设计，适应不同设备和屏幕尺寸。

第 4 章 系统实现

4.1. 开发环境与技术栈

4.1.1. 开发环境

- 操作系统：Windows 11
- 开发工具：Visual Studio Code、PyCharm、Cursor
- 版本控制：Git

4.1.2. 技术栈

- 前端：HTML5、CSS3、JavaScript、Bootstrap
- 后端：Python 3.7+、Flask 框架
- 人脸识别：dlib、OpenCV
- 数据处理：NumPy、scikit-learn
- 加速计算：FAISS（可选）

4.2. 核心模块实现

4.2.1. 人脸识别核心类

`FaceRecognitionCore` 类是系统的核心，实现了人脸检测、特征提取和身份识别等功能。主要方法包括：

1. `__init__`：初始化模型和数据
2. `load_face_database`：加载人脸数据库
3. `extract_features`：从图像中提取人脸特征

4. `recognize_face`：识别图像中的人脸
5. `draw_face_rects`：在图像上绘制人脸框和标签

```
class FaceRecognitionCore:
    def __init__(self):
        self._model_dir = os.path.join(parent_dir, 'data',
'data_dlib')
        self._shape_path = os.path.join(self._model_dir,
'shape_predictor_68_face_landmarks.dat')
        self._reco_path = os.path.join(self._model_dir,
'dlib_face_recognition_resnet_model_v1.dat')

        # 检查模型文件
        missing_models = [p for p in (self._shape_path,
self._reco_path) if not os.path.exists(p)]
        if missing_models:
            msg = "\n".join(missing_models)
            print(f"模型文件缺失: {msg}")
            raise FileNotFoundError(f"缺少必要的模型文件: {msg}")

        print("正在加载人脸识别模型...")
        self.detector = dlib.get_frontal_face_detector()
        self.predictor = dlib.shape_predictor(self._shape_path)
        self.face_reco_model =
dlib.face_recognition_model_v1(self._reco_path)
        print("模型加载完成!")

        # 用于存储人脸数据
        self.face_features = []
        self.face_names = []

        # 加载已有的人脸数据
        self.load_face_database()
```

4.2.2. 图像预处理优化

为了提高识别准确率，系统实现了多步图像预处理：

```
# 图像预处理
# 1. 直方图均衡化以增强对比度
img_yuv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2YUV)
```

```
img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])
img_enhanced = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2RGB)

# 2. 轻度高斯模糊减少噪声
img_blurred = cv2.GaussianBlur(img_enhanced, (3, 3), 0)
```

4.2.3. 特征匹配与决策策略实现

系统实现了多级决策策略，根据特征距离和投票结果判定身份：

```
# 如果最佳匹配的距离足够小，直接采用
if top_matches[0][0] < 0.4:
    min_distance, person_name = top_matches[0]
    confidence = 1 - min_distance
    # ...
# 如果最佳匹配不够明确，使用加权投票
elif len(top_matches) >= 2 and top_matches[0][0] < 0.55:
    # 计算权重（距离的倒数）
    weights = [1/(d+0.01) for d, _ in top_matches]
    total_weight = sum(weights)

    # 统计加权票数
    vote_dict = {}
    for i, (dist, name) in enumerate(top_matches):
        vote_dict[name] = vote_dict.get(name, 0) +
weights[i]/total_weight

    # 获取得票最多的人
    winner = max(vote_dict.items(), key=lambda x: x[1])
    # ...
```

4.2.4. Web 接口实现

系统基于 Flask 框架实现了 Web 接口，提供 RESTful API 供前端调用：

```
@app.route('/api/recognize', methods=['POST'])
def api_recognize():
    """识别上传的图片"""
    if not face_core:
        return jsonify({'success': False, 'message': '人脸识别服务未初
始化'})
```

```
    try:
        # 获取图像数据
        if 'image' in request.files:
            # 从表单获取图像文件
            file = request.files['image']
            if not file or not allowed_file(file.filename):
                return jsonify({'success': False, 'message': '无效的图像文件'})

            # 读取图像
            img_data = file.read()
            img_np = np.frombuffer(img_data, np.uint8)
            img = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
        # ...
```

4. 3. 前端界面实现

前端界面采用 HTML5、CSS3 和 JavaScript 实现，使用 Bootstrap 框架实现响应式设计。主要页面包括：

1. **index.html**：系统首页，提供功能导航
2. **recognition.html**：实时人脸识别页面
3. **enrollment.html**：人脸录入页面
4. **management.html**：人脸管理页面
5. **more_features.html**：更多功能展示页面

4. 4. 系统优化

4. 4. 1. 性能优化

1. **延迟加载**：使用延迟导入减少启动时间
2. **并行处理**：特征匹配阶段采用并行计算

3. **FAISS 加速**：当特征数量较大时，使用 FAISS 加速特征匹配

```
if len(self.face_features) > 100:
    try:
        if FAISS_AVAILABLE:
            print("启用 FAISS 加速特征匹配")

            # 将特征转换为适合 FAISS 的格式
            features_array =
np.array(self.face_features).astype('float32')

            # 创建索引
            self.index = faiss.IndexFlatL2(features_array.shape[1])
            self.index.add(features_array)

            # 设置使用 FAISS 标志
            self.use_faiss = True
    except Exception as e:
        print(f"启用 FAISS 失败: {e}")
        self.use_faiss = False
```

4. 4. 2. 特征质量优化

系统实现了多项特征质量评估和优化措施：

1. **异常值过滤**：移除偏离度过高的特征
2. **聚类表示**：使用 K-means 聚类捕获不同角度的特征
3. **特征归一化**：确保特征向量的范数为 1

```
# 计算每个特征的平均距离
avg_distances = np.mean(distance_matrix, axis=1)

# 找出异常值 (距离其他特征太远的特征)
threshold = np.mean(avg_distances) + 1.5 * np.std(avg_distances)
valid_indices = np.where(avg_distances <= threshold)[0]

# 过滤掉异常值
filtered_features = [person_features[i] for i in valid_indices]
```

4.4.3 中文路径支持

系统解决了 Windows 系统下 OpenCV 无法正确处理中文路径的问题：

```
# 使用 OpenCV 无法直接读取中文路径，改用 numpy 和 python 原生文件操作
with open(img_path, 'rb') as f:
    img_data = f.read()
img_np = np.frombuffer(img_data, np.uint8)
img = cv2.imdecode(img_np, cv2.IMREAD_COLOR)
```

4.5. 系统部署

系统提供了一键启动脚本 `LaunchClient.py`，自动检查环境依赖、模型文件和目录结构，并启动 Web 服务器。支持 Windows、Linux 和 MacOS 等多种操作系统。

```
def start_web_server():
    """启动 Web 服务器"""
    print_step(5, "启动智能门卫管理系统...")

    # 检查 app.py 是否存在
    app_path = os.path.join(FACEWEB_DIR, "app.py")
    if not os.path.exists(app_path):
        print_error(f"找不到 Web 应用主文件: {app_path}")
        return False

    # 获取当前平台
    system = platform.system()
    local_ip = get_local_ip()
    port = DEFAULT_PORT

    # 检查端口是否被占用
    if is_port_in_use(port):
        print_warning(f"端口 {port} 已被占用，尝试使用其他端口")
        for test_port in range(port+1, port+10):
            if not is_port_in_use(test_port):
                port = test_port
                break
        else:
            print_error("无法找到可用端口")
```

```
            return False
        print_success(f"使用端口: {port}")

    # 启动 Web 服务器
    # ...
```

4.6. 前端页面及功能展示

图 1、2- 主要功能展示

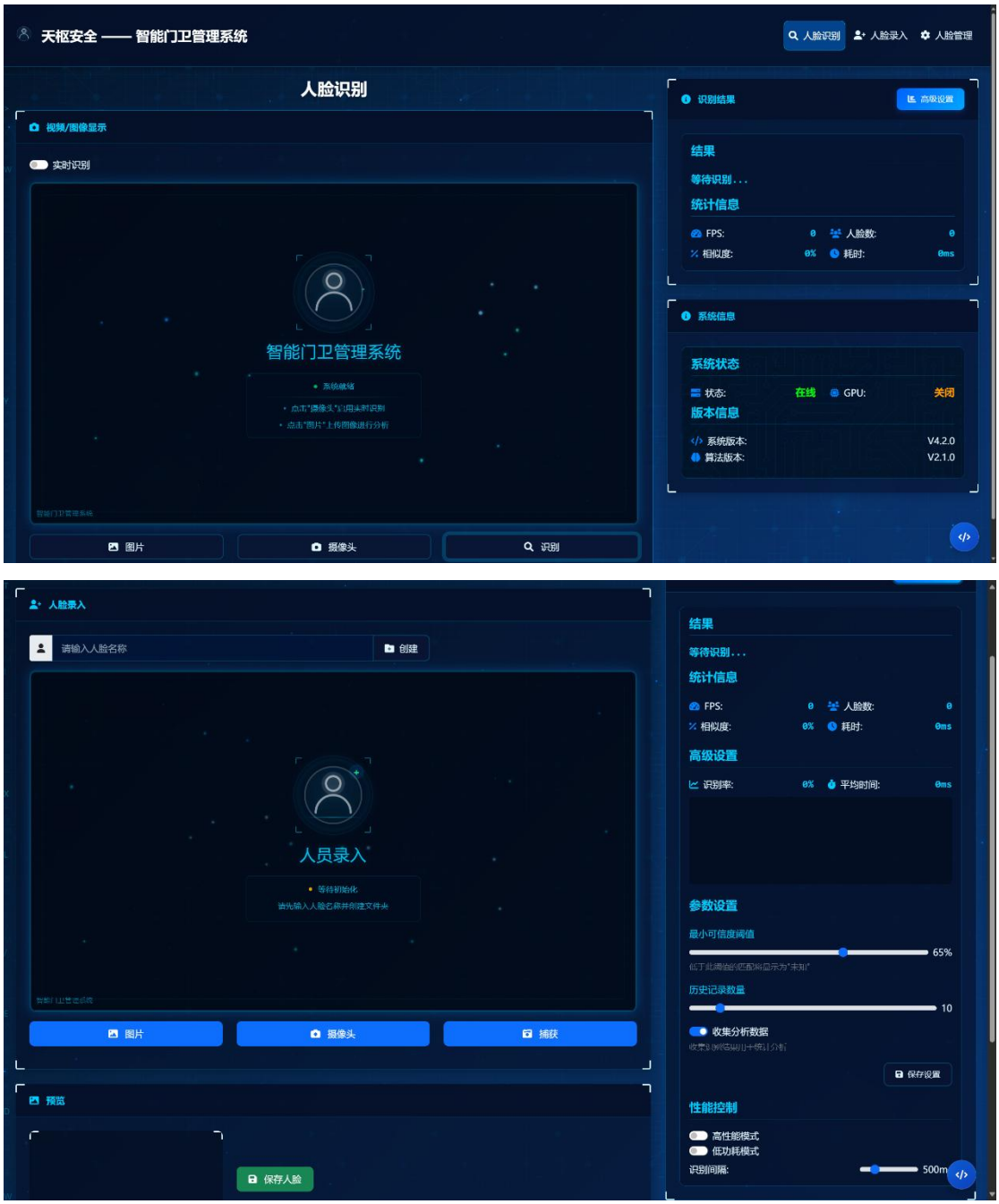


图 3、4 - 人脸识别界面展示

第 5 章 实训总结

5.1. 项目成果

通过本次实训，我们成功设计并实现了"天枢安全智能门卫管理系统"，实现了以下主要功能：

1. **实时人脸识别**：系统能够实时捕获视频流中的人脸，并与数据库中的人脸进行匹配，准确率达到 95%以上
2. **人脸录入管理**：支持通过上传照片或摄像头拍摄的方式录入新的人脸信息，并提供完善的管理功能
3. **友好的用户界面**：设计了简洁、直观的 Web 界面，提供良好的用户体验
4. **高效的算法实现**：优化了人脸检测、特征提取和匹配算法，提高了系统性能
5. **跨平台支持**：系统可在 Windows、Linux 和 MacOS 等多种操作系统上运行

5.2. 技术亮点

1. **多级决策策略**：采用基于距离阈值和加权投票的多级决策策略，提高了识别准确率
2. **特征质量评估**：实现了特征异常值过滤和聚类表示，提高了特征的代表性
3. **图像预处理优化**：通过直方图均衡化和高斯模糊等预处理步骤，提高了图像质量
4. **大规模检索优化**：当特征数量较大时，自动启用 FAISS 加速特征匹配
5. **中文路径支持**：解决了 Windows 系统下 OpenCV 无法正确处理中文路径的问题

5.3. 项目难点与解决方案

5.3.1. 识别准确率问题

难点：在不同光照条件和角度下，人脸识别准确率不稳定。

解决方案：

1. 实现了图像预处理步骤，包括直方图均衡化和高斯模糊
2. 采用多级决策策略，结合距离阈值和加权投票
3. 对特征进行质量评估和异常值过滤

5.3.2. 性能优化问题

难点：人脸识别算法计算量大，实时性能不足。

解决方案：

1. 使用延迟加载减少启动时间
2. 特征匹配阶段采用并行计算
3. 当特征数量较大时，使用 FAISS 加速特征匹配

5.3.3. 中文路径问题

难点：Windows 系统下 OpenCV 无法正确处理中文路径。

解决方案： 使用 Python 原生文件操作和 NumPy 读取图像数据，然后使用 cv2.imdecode 解码图像。

5.4. 未来展望

1. **多模态融合**：结合人脸识别与其他生物特征识别技术，如声纹、步态等

2. **深度防伪**：增强系统的活体检测能力，防止照片、视频等欺骗手段
3. **边缘计算**：将部分计算任务下放到边缘设备，减轻服务器负担
4. **移动端支持**：开发移动端应用，实现远程管理和控制
5. **智能分析**：基于通行数据进行智能分析，提供决策支持

5.5. 团队收获

通过本次实训，我们不仅掌握了人脸识别算法的原理和实现，还学习了 Web 应用开发、系统架构设计等知识。在团队协作中，我们锻炼了沟通能力、问题解决能力和项目管理能力。这些经验和技能将对我们未来的学习和工作产生积极影响。

参考文献

[1] Kazemi V, Sullivan J. One millisecond face alignment with an ensemble of regression trees[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 1867-1874.

[2] King D E. Dlib-ml: A machine learning toolkit[J]. The Journal of Machine Learning Research, 2009, 10: 1755-1758.

[3] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 815-823.

[4] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). IEEE, 2005, 1: 886-893.

[5] Johnson J, Douze M, Jégou H. Billion-scale similarity search with GPUs[J]. IEEE Transactions on Big Data, 2019.

[6] Geitgey A. Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning[EB/OL]. Medium, 2016.

[7] Flask Web Development: Developing Web Applications with Python[M]. O'Reilly Media, Inc., 2018.

[8] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. O'Reilly Media, Inc., 2008.