北京石油化工学院

人工智能研究院

计算机视觉课程设计指导书

——驾驶员疲劳检测系统

课程名称：计算机视觉课程设计

开设院系：人工智能研究院

计算机视觉课程组

2025 年 12 月 26 日

目录

1 课程设计概述

1.1 课程信息

课程基本信息

- **课程名称**：计算机视觉课程设计
- **开设院系**：人工智能研究院
- **设计题目**：驾驶员疲劳检测系统
- **设计周数**：2 周
- **学分数**：2 学分

1.2 设计背景

疲劳驾驶是道路交通安全的重大隐患。据统计，约有 20%-30% 的交通事故与驾驶员疲劳有关。疲劳驾驶会导致反应迟钝、注意力分散、判断失误，严重时甚至出现微睡眠现象（数秒的无意识状态）。

开发基于计算机视觉的疲劳检测系统具有重要的社会价值：

- **预防事故**：实时预警可大幅降低事故发生率
- **保护生命**：挽救驾驶员及他人的生命财产安全
- **技术融合**：综合运用图像处理、模式识别、机器学习等技术
- **实际应用**：可应用于车载系统、监控设备等场景

1.3 设计目标

通过本课程设计，学生应能够：

1. 理解计算机视觉在真实场景中的应用
2. 掌握人脸检测、特征提取、模式识别的基本方法
3. 熟练使用 OpenCV、Dlib 等计算机视觉库进行开发
4. 培养系统设计能力和工程实践能力

1.4 技术要求

基本要求

- 编程语言：Python 3.8+
- 核心库：OpenCV、Dlib、NumPy
- 开发环境：推荐使用 PyCharm、VS Code 或 Jupyter Notebook
- 版本控制：建议使用 Git 进行代码管理

1.5 系统功能要求

1. **人脸检测：**从视频流中准确定位人脸区域

2. **关键点提取：**检测 68 个面部关键点

3. **特征计算：**计算眼睛纵横比 (EAR)、嘴部纵横比 (MAR)

4. **疲劳判定：**根据 PERCLOS 准则判定疲劳状态

5. **结果展示：**可视化展示检测状态和疲劳告警

1.6 评估标准

表 1: 评分标准

等级	分数段	要求	权重
基础	60–75 分	• 能完成基本的人脸检测和关键点提取 • 能实现简单的 EAR 计算 • 代码能运行，有基本输出	30%
良好	76–85 分	• 人脸检测准确率达到 90% 以上 • 实现 PERCLOS 疲劳判定 • 代码结构清晰，有注释	25%
优秀	86–95 分	• 整体检测准确率 90% 以上 • 处理复杂场景（光照变化、角度偏转） • 有可视化界面或实时视频处理	30%
卓越	96–100 分	• 算法有创新改进 • 完整的测试报告和性能分析 • 支持多人脸检测或深度学习方法	15%

1.7 时间安排建议

表 2: 时间安排（共 2 周）

阶段	时间	主要任务
第一阶段	第 19 周	环境配置、文献调研、人脸检测与关键点提取实现
第二阶段	第 20 周	疲劳判定实现、系统集成测试、文档撰写

2 理论基础

2.1 计算机视觉基础概念

2.1.1 数字图像表示

数字图像可以表示为一个二维矩阵：

$$I(x,y) = \begin{cases} \text{灰度图像} & f : \mathbb{R}^2 \to [0,255] \\ \text{彩色图像} & f : \mathbb{R}^2 \to [0,255]^3 \end{cases} \tag{1}$$

其中，彩色图像通常使用 RGB 颜色空间，每个像素点由红 (R)、绿 (G)、蓝 (B) 三个通道组成。

2.1.2 图像处理基本操作

常用图像处理操作

1. **灰度化：**将彩色图像转换为灰度图像

$$\text{Gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

2. **滤波：**去除噪声，平滑图像
3. **边缘检测：**检测图像中的边缘信息
4. **直方图均衡化：**增强图像对比度

2.2 疲劳检测技术路线

2.2.1 传统图像处理方法

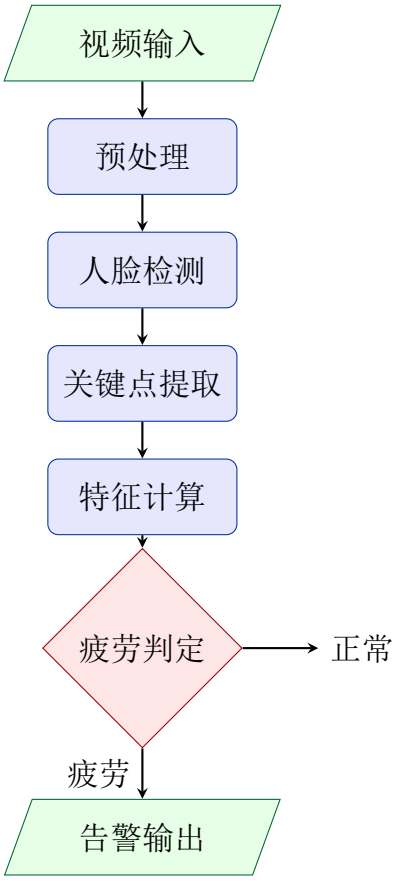

图 1: 传统疲劳检测流程

优点：

- 算法透明，易于理解和调试
- 计算量相对较小
- 不需要大量训练数据

缺点：

- 对复杂场景鲁棒性较差
- 需要手动设计特征
- 准确率有限

2.2.2 深度学习方法

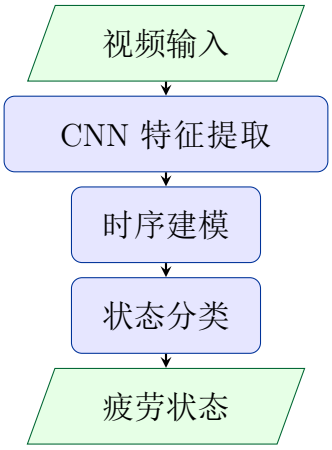

图 2: 深度学习疲劳检测流程

优点：

- 准确率高，鲁棒性强
- 端到端训练
- 可处理复杂场景

缺点：

- 需要大量标注数据
- 计算资源要求高
- 模型训练耗时长

2.2.3 混合方法（推荐）

结合传统方法和深度学习的优点，本课程设计推荐采用混合方案：

- **人脸检测：**使用 Dlib HOG+SVM 方法
- **关键点提取：**使用 68 点回归树模型
- **疲劳判定：**使用 PERCLOS 准则进行规则判定

这种方案既能让学生理解计算机视觉的基本原理，又能保证系统的检测准确率。

2.3 人脸检测技术

2.3.1 基于 Haar 特征的方法

Viola-Jones 算法使用 Haar-like 特征和 Adaboost 分类器：

- **优点**：速度快，可实时检测
- **缺点**：对侧脸、遮挡敏感
- **实现**：OpenCV 的`CascadeClassifier`

2.3.2 基于 HOG 特征的方法

方向梯度直方图（HOG）+ SVM 分类器：

- **优点**：对光照变化鲁棒
- **缺点**：计算量较大
- **实现**：Dlib 的`get_frontal_face_detector()`

2.4 面部关键点检测

面部关键点是疲劳特征计算的基础。常用的 68 点模型由 Kazemi 和 Sullivan 提出，通过 EERT（Ensemble of Regression Trees）算法实现。

2.4.1 68 点模型结构

关键点分布：

- **0-16**：下巴轮廓（17 点）
- **17-21**：左眉毛（5 点）
- **22-26**：右眉毛（5 点）
- **27-35**：鼻子（9 点）
- **36-41**：左眼（6 点）← 疲劳检测关键
- **42-47**：右眼（6 点）← 疲劳检测关键
- **48-67**：嘴巴（20 点）← 打哈欠检测

2.5 疲劳检测指标

2.5.1 PERCLOS 准则

PERCLOS（Percentage of Eyelid Closure Over Time）是最有效的疲劳检测指标，定义为单位时间内眼睛闭合程度超过阈值的时间比例。

$$\text{PERCLOS} = \frac{t_{\text{closed}}}{t_{\text{total}}} \times 100\% \tag{2}$$

其中：

- t_{closed}：眼睛闭合（EAR $<$ 阈值）的时间
- t_{total}：统计总时间

判定标准：

- PERCLOS $< 20\%$：清醒状态
- $20\% \leq$ PERCLOS $< 40\%$：轻度疲劳
- PERCLOS $\geq 40\%$：重度疲劳，需告警

2.5.2 眼睛纵横比（EAR）

EAR（Eye Aspect Ratio）是描述眼睛开合程度的几何特征，由 Soukupová 和 Čech 提出。

$$\text{EAR} = \frac{|p_2 - p_6| + |p_3 - p_5|}{2|p_1 - p_4|} \tag{3}$$

其中，p_1 到 p_6 是眼睛的 6 个关键点坐标。

EAR 数值范围：

- 睁眼状态：0.25 – 0.35
- 半闭状态：0.15 – 0.25
- 闭眼状态：0.01 – 0.05

疲劳判定阈值通常设置为 **0.20**（可调整）。

2.5.3 嘴部纵横比（MAR）

MAR（Mouth Aspect Ratio）用于检测打哈欠行为。

$$\text{MAR} = \frac{|p_{66} - p_{62}|}{|p_{63} - p_{61}|} \tag{4}$$

判定标准：

- 正常状态：$\text{MAR} < 0.5$
- 张嘴/打哈欠：$\text{MAR} \geq 0.5$

3 系统设计

3.1 整体架构设计

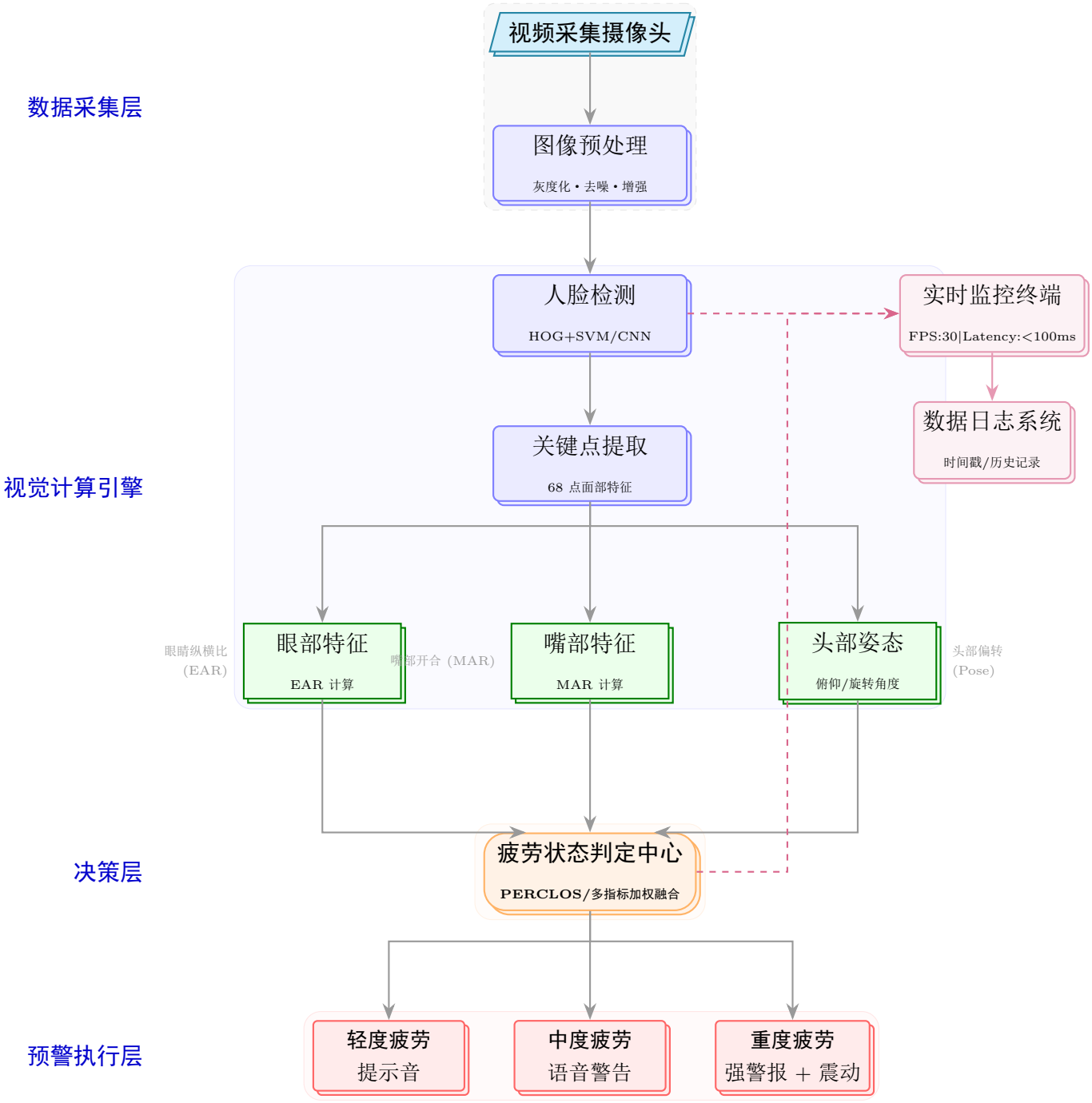

图 3: 系统整体架构

3.2 模块详细设计

3.2.1 预处理模块

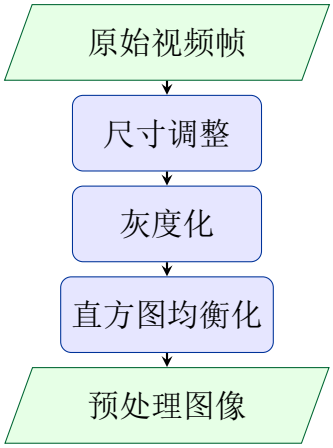

图 4: 预处理流程

功能说明：

- `resize`：统一图像尺寸，便于后续处理
- `cvtColor`：转换为灰度图像，减少计算量
- `equalizeHist`：增强对比度，改善光照影响

3.2.2 人脸检测模块

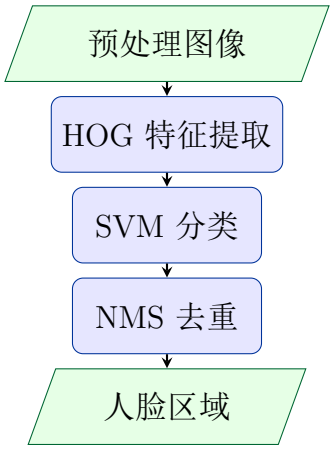

图 5: 人脸检测流程

关键技术：

1. **HOG 特征提取**：计算图像的梯度方向直方图

2. **SVM 分类**：使用支持向量机判断是否为人脸
3. **NMS 去重**：非极大值抑制，去除重叠检测框

3.2.3 关键点提取模块

功能说明：

- 使用 68 点回归树模型检测面部关键点
- 输入：人脸区域图像
- 输出：68 个 (x, y) 坐标点
- 模型文件：`shape_predictor_68_face_landmarks.dat`

3.2.4 特征计算模块

计算内容：

1. 左右眼 EAR 值
2. 嘴部 MAR 值
3. 头部姿态角（可选）

输出：特征向量 $[ear_L, ear_R, mar]$

3.2.5 疲劳判定模块

疲劳判定规则

状态机定义：

- **清醒**：EAR ≥ 0.2, PERCLOS $< 20\%$
- **轻度疲劳**：$0.15 \leq$ EAR < 0.2 或连续闭眼 2-3 秒
- **重度疲劳**：EAR < 0.15 或 PERCLOS $\geq 40\%$ 或连续打哈欠 3 次

3.3 项目目录结构

```
fatigue_detection/
    main.py                  # 主程序入口
    config.py                # 配置参数
    requirements.txt         # 依赖包列表
```

```
    README.md               # 项目说明文档
    detectors/              # 检测模块
        __init__.py
        face_detector.py        # 人脸检测
        landmark_detector.py    # 关键点检测
    features/               # 特征提取模块
        __init__.py
        ear_calculator.py       # EAR计算
        mar_calculator.py       # MAR计算
        pose_estimator.py       # 姿态估计(可选)
    fatigue/                # 疲劳判定模块
        __init__.py
        state_tracker.py        # 状态跟踪
        fatigue_detector.py     # 疲劳判定
    utils/                  # 工具函数
        __init__.py
        visualizer.py           # 可视化工具
        logger.py               # 日志记录
    models/                 # 模型文件目录
        shape_predictor_68_face_landmarks.dat
    data/                   # 测试数据
    tests/                  # 测试代码
        test.py
```

Listing 1: 推荐的项目目录结构

3.4 接口设计

```python
# 人脸检测模块接口
def detect_faces(image: np.ndarray) -> list:
    """
    人脸检测
    Args:
        image: 输入图像 (灰度图)
    Returns:
        人脸区域列表 [(left, top, right, bottom), ...]
    """

# 关键点检测模块接口
```

```python
def detect_landmarks(image: np.ndarray, face_rect: tuple) -> np.ndarray:
    """
    关键点检测
    Args:
        image: 输入图像
        face_rect: 人脸区域 (left, top, right, bottom)
    Returns:
        68个关键点坐标 (68, 2)
    """

# 疲劳判定模块接口
def detect_fatigue(ear_history: list, mar: float) -> dict:
    """
    疲劳判定
    Args:
        ear_history: EAR历史值列表
        mar: 当前MAR值
    Returns:
        判定结果 {'state': str, 'perclos': float, 'confidence': float}
    """
```

Listing 2: 核心接口定义

4 实现指南

4.1 开发环境配置

4.1.1 安装 Python 依赖

```
# 创建虚拟环境（推荐）
python -m venv venv
source venv/bin/activate  # Linux/Mac
# 或
venv\Scripts\activate     # Windows

# 安装核心依赖
pip install opencv-python numpy scipy

# 安装Dlib
# 方法1: 直接安装（需要编译环境）
pip install dlib

# 方法2: 使用预编译wheel文件（Windows推荐）
# 从 https://pypi.org/project/dlib/#files 下载对应版本
pip install dlib-19.x.x-cpxx-cpxx-win_amd64.whl

# 安装其他工具
pip install imutils matplotlib tqdm
```

Listing 3: 安装命令

4.1.2 验证安装

```
import cv2
import dlib
import numpy as np
print(f"OpenCV版本: {cv2.__version__}")
print(f"Dlib版本: {dlib.__version__}")
print(f"NumPy版本: {np.__version__}")
print("环境配置成功！")
```

Listing 4: 验证环境配置

4.2 核心算法实现

4.2.1 EAR 计算实现

```python
import numpy as np

class EARCalculator:
    """眼睛纵横比计算器"""

    # 左眼关键点索引 (Dlib 68点模型)
    LEFT_EYE_INDICES = [36, 37, 38, 39, 40, 41]
    # 右眼关键点索引
    RIGHT_EYE_INDICES = [42, 43, 44, 45, 46, 47]

    @staticmethod
    def calculate_ear(eye_points):
        """
        计算眼睛纵横比

        参数:
            eye_points: 眼睛的6个关键点坐标 (N, 2)

        返回:
            ear: 眼睛纵横比
        """
        # 计算垂直距离
        vertical_1 = np.linalg.norm(eye_points[1] - eye_points[5])
        vertical_2 = np.linalg.norm(eye_points[2] - eye_points[4])

        # 计算水平距离
        horizontal = np.linalg.norm(eye_points[0] - eye_points[3])

        # EAR公式
        ear = (vertical_1 + vertical_2) / (2.0 * horizontal)

        return ear

    @staticmethod
    def extract_eyes(shape):
        """
```

```
        从68点模型中提取左右眼关键点

        参数:
            shape: 68个关键点坐标 (68, 2)

        返回:
            left_eye: 左眼6个点
            right_eye: 右眼6个点
        """
        left_eye = shape[EARCalculator.LEFT_EYE_INDICES]
        right_eye = shape[EARCalculator.RIGHT_EYE_INDICES]
        return left_eye, right_eye

    @staticmethod
    def compute_avg_ear(shape):
        """
        计算双眼平均EAR

        参数:
            shape: 68个关键点坐标

        返回:
            avg_ear: 平均EAR
            left_ear: 左眼EAR
            right_ear: 右眼EAR
        """
        left_eye, right_eye = EARCalculator.extract_eyes(shape)
        left_ear = EARCalculator.calculate_ear(left_eye)
        right_ear = EARCalculator.calculate_ear(right_eye)
        avg_ear = (left_ear + right_ear) / 2.0
        return avg_ear, left_ear, right_ear
```

Listing 5: EAR 计算模块

4.2.2 疲劳状态跟踪实现

```
from collections import deque
from enum import Enum

class FatigueState(Enum):
```

```
    """疲劳状态枚举"""
    AWAKE = "清醒"
    MILD = "轻度疲劳"
    SEVERE = "重度疲劳"


class FatigueTracker:
    """疲劳状态跟踪器"""

    def __init__(self, window_size=60):
        self.window_size = window_size
        self.ear_buffer = deque(maxlen=window_size)
        self.mar_buffer = deque(maxlen=window_size)

        # 统计计数
        self.blink_count = 0
        self.yawn_count = 0

        # 配置阈值
        self.EAR_CLOSED = 0.15       # 闭眼阈值
        self.EAR_BLINK = 0.2         # 眨眼阈值
        self.MAR_YAWN = 0.5          # 打哈欠阈值
        self.PERCLOS_MILD = 0.2      # 轻度疲劳阈值
        self.PERCLOS_SEVERE = 0.4    # 重度疲劳阈值

        # 状态标记
        self.was_blinking = False
        self.is_yawning = False

    def update(self, ear, mar):
        """
        更新状态

        参数:
            ear: 眼睛纵横比
            mar: 嘴部纵横比
        """
        self.ear_buffer.append(ear)
        self.mar_buffer.append(mar)

```

```
45          # 检测眨眼
46          if ear < self.EAR_CLOSED:
47              if not self.was_blinking:
48                  self.blink_count += 1
49                  self.was_blinking = True
50          else:
51              self.was_blinking = False
52
53          # 检测打哈欠
54          if mar > self.MAR_YAWN:
55              if not self.is_yawning:
56                  self.yawn_count += 1
57                  self.is_yawning = True
58          else:
59              self.is_yawning = False
60
61      def get_perclos(self):
62          """计算PERCLOS"""
63          if len(self.ear_buffer) == 0:
64              return 0.0
65
66          closed_frames = sum(1 for ear in self.ear_buffer if ear <
    self.EAR_BLINK)
67          return closed_frames / len(self.ear_buffer)
68
69      def get_state(self):
70          """获取当前疲劳状态"""
71          perclos = self.get_perclos()
72
73          if perclos >= self.PERCLOS_SEVERE or self.yawn_count >= 3:
74              return FatigueState.SEVERE
75          elif perclos >= self.PERCLOS_MILD:
76              return FatigueState.MILD
77          else:
78              return FatigueState.AWAKE
79
80      def get_statistics(self):
81          """获取统计信息"""
82          return {
83              "perclos": self.get_perclos(),
```

```
            "blink_count": self.blink_count,
            "yawn_count": self.yawn_count,
            "avg_ear": np.mean(self.ear_buffer) if self.ear_buffer
    else 0.0,
            "avg_mar": np.mean(self.mar_buffer) if self.mar_buffer
    else 0.0
        }
```

Listing 6: 疲劳状态跟踪器

4.2.3 完整检测流程实现

```
import cv2
import dlib
import numpy as np
from imutils import face_utils
from detectors.face_detector import FaceDetector
from detectors.landmark_detector import LandmarkDetector
from features.ear_calculator import EARCalculator
from fatigue.state_tracker import FatigueTracker
from utils.visualizer import Visualizer

class FatigueDetector:
    """疲劳检测系统"""

    def __init__(self, model_path: str, config: dict = None):
        """
        初始化检测系统

        Args:
            model_path: 68点模型文件路径
            config: 配置参数
        """
        # 初始化人脸检测器
        self.face_detector = FaceDetector()

        # 初始化关键点检测器
        self.landmark_detector = LandmarkDetector(model_path)

        # 初始化疲劳跟踪器
```

```
        window_size = config.get('window_size', 60) if config else 60
        self.tracker = FatigueTracker(window_size=window_size)

        # 初始化可视化器
        self.visualizer = Visualizer()

        # 配置参数
        self.config = config or {}

    def process(self, frame: np.ndarray) -> tuple:
        """
        处理单帧图像

        Args:
            frame: 输入图像 (BGR格式)

        Returns:
            (result_image, fatigue_state, statistics)
        """
        # 预处理
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # 人脸检测
        faces = self.face_detector.detect(gray)

        for face in faces:
            # 关键点检测
            shape = self.landmark_detector.detect(gray, face)
            shape = face_utils.shape_to_np(shape)

            # 计算EAR
            avg_ear, left_ear, right_ear = EARCalculator.
compute_avg_ear(shape)

            # 计算MAR
            mouth = shape[48:68]
            mar = self.calculate_mar(mouth)

            # 更新跟踪器
            self.tracker.update(avg_ear, mar)
```

```
68
69              # 获取状态
70              state = self.tracker.get_state()
71              stats = self.tracker.get_statistics()
72
73              # 可视化
74              self.visualizer.draw_landmarks(frame, shape)
75              self.visualizer.draw_status(frame, state, avg_ear, mar,
    stats)
76
77          return frame, state, stats
78
79      def calculate_mar(self, mouth_points):
80          """计算嘴部纵横比"""
81          A = np.linalg.norm(mouth_points[2] - mouth_points[10])  #
    66-62
82          B = np.linalg.norm(mouth_points[4] - mouth_points[8])   #
    64-60
83          C = np.linalg.norm(mouth_points[0] - mouth_points[6])   #
    48-54
84          mar = (A + B) / (2.0 * C)
85          return mar
86
87
88  # 主程序
89  def main():
90      """主函数"""
91      # 配置参数
92      config = {
93          'window_size': 60,
94          'frame_width': 640,
95          'frame_height': 480
96      }
97
98      # 初始化检测器
99      detector = FatigueDetector(
100         model_path="models/shape_predictor_68_face_landmarks.dat",
101         config=config
102     )
103
```

```
    # 打开摄像头
    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # 调整尺寸
        frame = cv2.resize(frame, (config['frame_width'], config['
    frame_height']))

        # 检测疲劳
        result, state, stats = detector.process(frame)

        # 显示结果
        cv2.imshow("Fatigue Detection", result)

        # 按ESC退出
        if cv2.waitKey(1) & 0xFF == 27:
            break

    cap.release()
    cv2.destroyAllWindows()


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

Listing 7: 完整的疲劳检测流程

4.3 常见问题及解决方案

表 3: 常见问题排查表

问题现象	可能原因	解决方案
无法检测到人脸	光照不足或角度过大	增加直方图均衡化，调整拍摄角度
检测到多个人脸	画面中有多人	只取面积最大的人脸
EAR 值异常	关键点检测错误	检查模型文件路径，重新下载模型
误报率高	阈值设置不当	调整 EAR/MAR/PERCLOS 阈值参数
程序运行缓慢	分辨率过高	适当降低 frame_width 和 frame_height
Dlib 安装失败	缺少 C++ 编译器	使用预编译 wheel 文件安装

4.4 配置参数说明

```python
# config.py

"""
疲劳检测系统配置文件
"""

# 人脸检测配置
FACE_DETECTION_CONFIG = {
    'scale_factor': 1.0,          # 图像缩放因子
    'min_neighbors': 5,           # 最小邻居数
    'min_size': (30, 30),         # 最小人脸尺寸
}

# 特征计算配置
FEATURE_CONFIG = {
    'ear_threshold': 0.2,         # EAR阈值
    'mar_threshold': 0.5,         # MAR阈值
    'ear_closed': 0.15,           # 闭眼判定阈值
```

```
}

# 疲劳判定配置
FATIGUE_CONFIG = {
    'window_size': 60,           # 时间窗口大小(帧)
    'perclos_mild': 0.2,         # 轻度疲劳阈值
    'perclos_severe': 0.4,       # 重度疲劳阈值
    'yawn_trigger_count': 3,     # 打哈欠告警次数
}

# 可视化配置
VISUALIZATION_CONFIG = {
    'show_landmarks': True,      # 显示关键点
    'show_status': True,         # 显示状态信息
    'show_ear': True,            # 显示EAR值
    'show_mar': True,            # 显示MAR值
}
```

Listing 8: 完整配置文件示例

5 测试与评估

5.1 测试数据集

5.1.1 公开数据集

表 4: 疲劳检测公开数据集

数据集	描述	下载地址
YawDD	打哈欠检测数据集	https://www.kaggle.com/datasets/enider/yawdd-da
UTA-RLDD	实时疲劳检测数据集	https://sites.google.com/view/utarldd/home
NTHU-DDD	疲劳驾驶视频数据	http://cv.cs.nthu.edu.tw/php/callforpaper/datas
CEW	闭眼检测数据集	https://opendatalab.com/OpenDataLab/CEW_Closed_

数据集详细信息：

1. **YawDD（Yawning Detection Dataset）**
   - **内容：**322 个视频（数据集 1）+ 29 个视频（数据集 2）
   - **特点：**包含男/女驾驶员，有/无眼镜，使用车内摄像头录制
   - **备用下载：**
     - IEEE DataPort: https://ieee-dataport.org/open-access/yawdd-yawning-detect
     - Roboflow: https://universe.roboflow.com/utarlddv1/yawdd-nx0vr
2. **UTA-RLDD（Real-Life Drowsiness Dataset）**
   - **内容：**约 30 小时 RGB 视频，60 名参与者，180 个视频
   - **特点：**目前最大的真实疲劳检测数据集，专注于微表情捕捉
   - **备用下载:**Kaggle https://www.kaggle.com/datasets/rishab260/uta-reallife-drowsi
3. **NTHU-DDD（Driver Drowsiness Dataset）**
   - **内容：**包含训练集和评估集的完整视频数据
   - **特点：**多样化条件（不同光照、眼镜、头部姿态），清醒和疲劳状态
   - **备用下载:**Kaggle https://www.kaggle.com/datasets/ismailnasri20/driver-drowsines
4. **CEW（Closed Eyes in the Wild）**
   - **内容：**2,423 张照片（闭眼 1,192 张，睁眼 1,231 张）

- **特点**：图片数据集，易于上手，适合初学者
- **备用下载**:http://parnec.nuaa.edu.cn/_upload/tpl/02/db/731/template731/pages/xtan/ClosedEyeDatabases.html

下载建议：

- **初学者**：推荐使用 CEW（图片数据集）
- **完整测试**：推荐使用 UTA-RLDD（最全面）
- **特定功能**：推荐使用 YawDD（打哈欠检测）

5.1.2 自建测试集

建议学生自行录制包含以下场景的测试数据：

1. **正常状态**：白天、光照良好、正常驾驶状态
2. **疲劳状态**：模拟眨眼、打哈欠、点头等行为
3. **光照变化**：正常光照、弱光、逆光
4. **角度变化**：正面、轻微侧脸、点头动作
5. **多人测试**：邀请 2-3 位同学协助录制

5.2 性能评估指标

评估指标定义

1. **准确率 (Accuracy)**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **精确率 (Precision)**

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. **召回率 (Recall)**

$$\text{Recall} = \frac{TP}{TP + FN}$$

4. **F1 分数 (F1-Score)**

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. **帧率 (FPS)** 每秒处理帧数，反映实时性能

其中：

- TP（True Positive）：正确检测为疲劳
- TN（True Negative）：正确判定为清醒
- FP（False Positive）：误报（清醒判为疲劳）
- FN（False Negative）：漏报（疲劳判为清醒）

5.3 测试代码示例

```
import os
import time
import cv2
import numpy as np
from typing import Dict, List
import pandas as pd

class Evaluator:
    """性能评估器"""

    def __init__(self, detector):
        """
        初始化评估器

        Args:
            detector: 疲劳检测器实例
        """
        self.detector = detector

    def evaluate(self, test_dir: str, ground_truth: Dict[str, str])
    -> Dict:
        """
        评估检测性能

        Args:
            test_dir: 测试视频目录
            ground_truth: 真实标签 {filename: state}

        Returns:
```

```
        评估结果字典
    """
    results = {
        'total': 0,
        'correct': 0,
        'tp': 0,  # True Positive
        'tn': 0,  # True Negative
        'fp': 0,  # False Positive
        'fn': 0,  # False Negative
        'processing_times': [],
        'errors': []
    }

    for filename, true_state in ground_truth.items():
        video_path = os.path.join(test_dir, filename)

        if not os.path.exists(video_path):
            results['errors'].append(f"{filename}: 文件不存在")
            continue

        # 处理视频
        cap = cv2.VideoCapture(video_path)
        frame_predictions = []

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # 记录处理时间
            start_time = time.time()
            _, pred_state, _ = self.detector.process(frame)
            elapsed = (time.time() - start_time) * 1000
            results['processing_times'].append(elapsed)

            frame_predictions.append(pred_state.value)

        cap.release()

        # 投票决定最终预测
```

```
            if frame_predictions:
                pred_state = max(set(frame_predictions),
                                 key=frame_predictions.count)

                results['total'] += 1

                # 统计混淆矩阵
                if true_state == "疲劳":
                    if pred_state == "重度疲劳" or pred_state == "轻度疲劳":
                        results['tp'] += 1
                        results['correct'] += 1
                    else:
                        results['fn'] += 1
                else:  # 清醒
                    if pred_state == "清醒":
                        results['tn'] += 1
                        results['correct'] += 1
                    else:
                        results['fp'] += 1

        # 计算各项指标
        results['accuracy'] = (
            results['correct'] / results['total'] * 100
            if results['total'] > 0 else 0
        )

        results['precision'] = (
            results['tp'] / (results['tp'] + results['fp']) * 100
            if (results['tp'] + results['fp']) > 0 else 0
        )

        results['recall'] = (
            results['tp'] / (results['tp'] + results['fn']) * 100
            if (results['tp'] + results['fn']) > 0 else 0
        )

        results['f1'] = (
            2 * results['precision'] * results['recall'] /
            (results['precision'] + results['recall'])
```

```
            if (results['precision'] + results['recall']) > 0 else 0
        )

        results['avg_fps'] = (
            1000 / np.mean(results['processing_times'])
            if results['processing_times'] else 0
        )

        return results

    def generate_report(self, results: Dict, output_path: str = '
    report.txt'):
        """生成评估报告"""
        report = f"""
========================================
疲劳检测系统性能评估报告
========================================

一、总体指标
----------------------------------------
测试样本总数: {results['total']}
正确分类数: {results['correct']}

二、分类指标
----------------------------------------
准确率 (Accuracy): {results['accuracy']:.2f}%
精确率 (Precision): {results['precision']:.2f}%
召回率 (Recall): {results['recall']:.2f}%
F1分数: {results['f1']:.2f}%

三、混淆矩阵
----------------------------------------
                预测清醒     预测疲劳
真实清醒        {results['tn']}          {results['fp']}
真实疲劳        {results['fn']}          {results['tp']}

四、性能指标
----------------------------------------
平均帧率: {results['avg_fps']:.2f} FPS

```

```
五、错误样本
----------------------------------------
"""

        if results['errors']:
            for error in results['errors']:
                report += f"- {error}\n"
        else:
            report += "无错误\n"

        report += "========================================\n"

        # 保存报告
        with open(output_path, 'w', encoding='utf-8') as f:
            f.write(report)

        print(report)
        return report


# 使用示例
if __name__ == '__main__':
    from main import FatigueDetector

    # 创建检测器
    detector = FatigueDetector("models/
   shape_predictor_68_face_landmarks.dat")

    # 创建评估器
    evaluator = Evaluator(detector)

    # 定义真实标签
    ground_truth = {
        'test001.mp4': '清醒',
        'test002.mp4': '疲劳',
        'test003.mp4': '清醒',
        # ... 更多测试样本
    }

    # 运行评估
```

```
    results = evaluator.evaluate('./data/test', ground_truth)

    # 生成报告
    evaluator.generate_report(results, 'evaluation_report.txt')
```

Listing 9: 性能评估代码

5.4 测试报告模板

测试报告内容要求

提交的测试报告应包含以下内容：

1. **系统概述**
   - 系统架构图
   - 技术路线说明
   - 创新点（如有）
2. **实现细节**
   - 核心算法说明
   - 关键代码片段
   - 参数调优过程
3. **测试结果**
   - 测试数据集描述
   - 各项性能指标
   - 典型案例分析
4. **问题分析**
   - 遇到的问题及解决方案
   - 系统局限性
   - 改进方向

5.5 常见问题诊断

表 5: 问题诊断指南

症状	诊断	调优建议
检测率低	人脸检测失败	增加光照补偿，调整检测器参数
误报过多	EAR 阈值过低	调高 EAR 阈值，增加时间窗口
漏报严重	PERCLOS 阈值过高	降低 PERCLOS 阈值
速度慢	算法效率低	降低分辨率，使用跳帧处理
内存占用高	缓冲区过大	减小 window_size 参数

6 扩展思考

6.1 系统优化方向

6.1.1 算法优化

1. **多模态融合**
   - 融合头部姿态信息（点头、摇头检测）
   - 融合视线方向（眼神呆滞检测）
   - 结合心率监测（如配备摄像头）
2. **深度学习增强**
   - 使用 CNN 提取更鲁棒的眼部特征
   - 使用 LSTM 建模时序信息
   - 注意力机制聚焦关键区域
3. **自适应阈值**
   - 个性化 EAR 基准值学习
   - 动态调整疲劳判定阈值
   - 环境自适应（光照、角度）

6.1.2 工程优化

1. **性能优化**
   - GPU 加速人脸检测和关键点提取
   - 多线程并行处理
   - 模型量化和剪枝
2. **鲁棒性增强**
   - 光照补偿算法
   - 侧面人脸检测支持
   - 遮挡情况处理
3. **边缘部署**
   - 树莓派等嵌入式设备部署
   - 移动端移植（Android/iOS）
   - 车载系统集成

6.2 进阶功能扩展

6.2.1 驾驶员行为分析

表 6: 可扩展的驾驶行为检测功能

功能	检测方法	应用场景
分心检测	视线方向、头部姿态	防止开车看手机
抽烟检测	嘴部动作、手势识别	禁烟提醒
打电话检测	手持物体动作检测	防止开车打电话
情绪分析	面部表情识别	情绪异常预警

6.2.2 车载系统集成

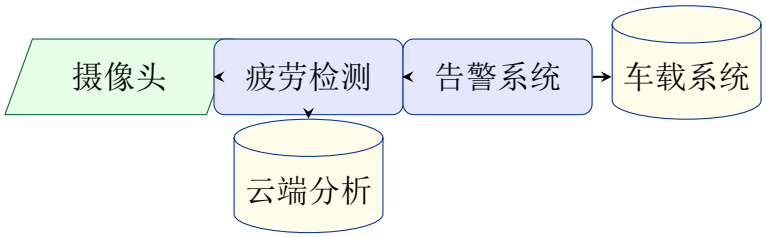

图 6: 车载系统集成架构

6.2.3 Web 界面开发

推荐技术栈：

- **后端**：Flask / FastAPI
- **前端**：Vue.js / React
- **实时通信**：WebSocket

功能模块：

- 实时视频流显示
- 疲劳状态监控面板
- 历史数据统计
- 告警记录查询

6.3 学术研究方向

6.3.1 前沿技术

1. **3D 面部重建**
    - 从 2D 视频恢复 3D 面部信息
    - 更精确的姿态估计
    - 深度特征提取
2. **轻量化模型**
    - MobileNet、EfficientNet 等轻量级网络
    - 知识蒸馏技术
    - 模型压缩与加速
3. **联邦学习**
    - 保护用户隐私
    - 分布式模型训练
    - 个性化自适应
4. **对抗训练**
    - 提高模型鲁棒性
    - 生成困难样本
    - 领域自适应

6.3.2 应用拓展

- **智能交通：**驾驶员状态监控、辅助驾驶
- **安防监控：**异常行为检测、疲劳值班预警
- **医疗健康：**睡眠监测、疲劳综合征诊断
- **人机交互：**疲劳状态下的交互调整

6.4 开源项目推荐

表 7: 相关开源项目

项目名称	语言	特点
dlib	C++/Python	人脸检测与关键点提取
MediaPipe	Python/JS	谷歌开源，轻量级人脸网格
OpenFace	Python/Torch	面部行为分析工具包
MTCNN	Python/PyTorch | 多任务级联人脸检测	

A 附录 A：OpenCV 常用函数参考

表 8: OpenCV 常用函数

函数	说明
cv2.VideoCapture()	打开视频文件或摄像头
cv2.cvtColor()	颜色空间转换
cv2.resize()	调整图像尺寸
cv2.equalizeHist()	直方图均衡化
cv2.GaussianBlur()	高斯滤波
cv2.rectangle()	绘制矩形
cv2.putText()	绘制文字
cv2.polylines()	绘制多边形
cv2.waitKey()	等待键盘输入

B 附录 B：Dlib 常用函数参考

表 9: Dlib 常用函数

函数	说明
dlib.get_frontal_face_detector()	获取人脸检测器
dlib.shape_predictor()	加载关键点检测模型
detector(gray, 0)	检测人脸，返回人脸区域列表
predictor(gray, face)	检测面部关键点，返回 68 个点
face_utils.shape_to_np()	将 shape 转换为 NumPy 数组
face_utils.rect_to_bb()	将人脸矩形转换为边界框

C 附录 C：完整代码框架

```python
"""
疲劳检测系统主程序
"""

import cv2
import dlib
import numpy as np
import argparse
```

```
from pathlib import pathlib

from detectors.face_detector import FaceDetector
from detectors.landmark_detector import LandmarkDetector
from features.ear_calculator import EARCalculator
from fatigue.state_tracker import FatigueTracker
from utils.visualizer import Visualizer


def parse_args():
    """解析命令行参数"""
    parser = argparse.ArgumentParser(description='疲劳检测系统')
    parser.add_argument('--camera', type=int, default=0, help='摄像头设备ID')
    parser.add_argument('--model', type=str,
                        default='models/shape_predictor_68_face_landmarks.dat',
                        help='模型文件路径')
    parser.add_argument('--width', type=int, default=640, help='视频宽度')
    parser.add_argument('--height', type=int, default=480, help='视频高度')
    parser.add_argument('--window', type=int, default=60, help='时间窗口大小')
    return parser.parse_args()


def main():
    """主函数"""
    args = parse_args()

    # 检查模型文件
    model_path = pathlib.Path(args.model)
    if not model_path.exists():
        print(f"错误: 模型文件不存在 - {args.model}")
        return

    # 初始化模块
    print("初始化检测器...")
    face_detector = FaceDetector()
```

```
    landmark_detector = LandmarkDetector(str(model_path))
    fatigue_tracker = FatigueTracker(window_size=args.window)
    visualizer = Visualizer()

    # 打开摄像头
    print(f"打开摄像头 {args.camera}...")
    cap = cv2.VideoCapture(args.camera)

    if not cap.isOpened():
        print("错误: 无法打开摄像头")
        return

    cap.set(cv2.CAP_PROP_FRAME_WIDTH, args.width)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, args.height)

    print("开始检测（按ESC退出）...")

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # 预处理
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # 人脸检测
        faces = face_detector.detect(gray)

        for face in faces:
            # 关键点检测
            shape = landmark_detector.detect(gray, face)
            shape = shape.astype(np.int32)

            # 计算EAR
            avg_ear, left_ear, right_ear = EARCalculator.
compute_avg_ear(shape)

            # 计算MAR
            mouth = shape[48:68]
            mar = calculate_mar(mouth)
```

```
83
84              # 更新跟踪器
85              fatigue_tracker.update(avg_ear, mar)
86
87              # 获取状态和统计
88              state = fatigue_tracker.get_state()
89              stats = fatigue_tracker.get_statistics()
90
91              # 可视化
92              visualizer.draw_landmarks(frame, shape)
93              visualizer.draw_status(frame, state, avg_ear, mar, stats)
94
95          # 显示结果
96          cv2.imshow("Fatigue Detection", frame)
97
98          # 按ESC退出
99          if cv2.waitKey(1) & 0xFF == 27:
100             break
101
102     # 释放资源
103     cap.release()
104     cv2.destroyAllWindows()
105     print("程序结束")
106
107
108 def calculate_mar(mouth_points):
109     """计算嘴部纵横比"""
110     A = np.linalg.norm(mouth_points[2] - mouth_points[10])
111     B = np.linalg.norm(mouth_points[4] - mouth_points[8])
112     C = np.linalg.norm(mouth_points[0] - mouth_points[6])
113     mar = (A + B) / (2.0 * C)
114     return mar
115
116
117 if __name__ == '__main__':
118     main()
```

Listing 10: main.py 完整框架

D 附录 D：提交材料清单

课程设计提交清单

代码部分：

- 完整的源代码（含注释）
- requirements.txt 依赖文件
- README.md 说明文档
- 配置文件（如有）

文档部分：

- 课程设计报告（docx 格式）
- 测试报告
- 使用说明书

演示部分：

- 测试视频集（至少 3 段）
- 检测结果截图
- 演示视频

E 附录 E：参考文献

1. Soukupová T, Čech J. Eye blink detection using facial landmarks[J]. 2016.
2. Dlib C++ Library. http://dlib.net/
3. King D E. Dlib-ml: A Machine Learning Toolkit[J]. Journal of Machine Learning Research, 2009, 10(Jul):1755-1758.
4. Kazemi V, Sullivan J. One millisecond face alignment with an ensemble of regression trees[C]//CVPR. 2014.
5. Viola P, Jones M. Rapid object detection using a boosted cascade of simple features[C]//CVPR. 2001.
6. OpenCV 官方文档: https://docs.opencv.org/

7. Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media.

祝各位同学课程设计顺利！

如有问题，请联系课程组