



北京石油化工学院
BEIJING INSTITUTE OF PETROCHEMICAL TECHNOLOGY

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	5
1.1 课程信息	5
1.2 设计背景	5
1.3 设计目标	5
1.4 技术要求	6
1.5 系统功能要求	6
1.6 评估标准	7
1.7 时间安排建议	7
2 理论基础	8
2.1 计算机视觉基础概念	8
2.1.1 数字图像表示	8
2.1.2 图像处理基本操作	8
2.2 疲劳检测技术路线	9
2.2.1 传统图像处理方法	9
2.2.2 深度学习方法	10
2.2.3 混合方法（推荐）	10
2.3 人脸检测技术	11
2.3.1 基于 Haar 特征的方法	11
2.3.2 基于 HOG 特征的方法	11
2.4 面部关键点检测	11
2.4.1 68 点模型结构	11
2.5 疲劳检测指标	12
2.5.1 PERCLOS 准则	12
2.5.2 眼睛纵横比（EAR）	12
2.5.3 嘴部纵横比（MAR）	13
3 系统设计	14
3.1 整体架构设计	14
3.2 模块详细设计	15
3.2.1 预处理模块	15
3.2.2 人脸检测模块	15
3.2.3 关键点提取模块	16
3.2.4 特征计算模块	16
3.2.5 疲劳判定模块	16
3.3 项目目录结构	16

3.4 接口设计	17
4 实现指南	19
4.1 开发环境配置	19
4.1.1 安装 Python 依赖	19
4.1.2 验证安装	19
4.2 核心算法实现	20
4.2.1 EAR 计算实现	20
4.2.2 疲劳状态跟踪实现	21
4.2.3 完整检测流程实现	24
4.3 常见问题及解决方案	28
4.4 配置参数说明	28
5 测试与评估	30
5.1 测试数据集	30
5.1.1 公开数据集	30
5.1.2 自建测试集	31
5.2 性能评估指标	31
5.3 测试代码示例	32
5.4 测试报告模板	37
5.5 常见问题诊断	38
6 扩展思考	39
6.1 系统优化方向	39
6.1.1 算法优化	39
6.1.2 工程优化	39
6.2 进阶功能扩展	40
6.2.1 驾驶员行为分析	40
6.2.2 车载系统集成	40
6.2.3 Web 界面开发	40
6.3 学术研究方向	41
6.3.1 前沿技术	41
6.3.2 应用拓展	41
6.4 开源项目推荐	42
A 附录 A: OpenCV 常用函数参考	43
B 附录 B: Dlib 常用函数参考	43

C 附录 C：完整代码框架	43
D 附录 D：提交材料清单	47
E 附录 E：参考文献	47

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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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：

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

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

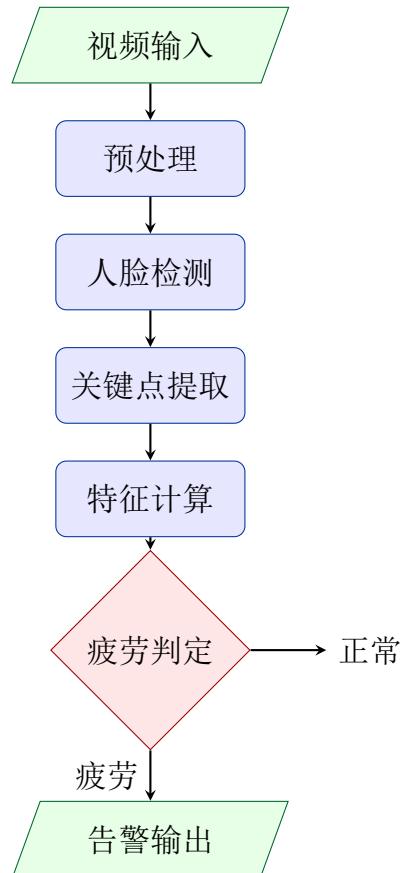


图 1: 传统疲劳检测流程

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

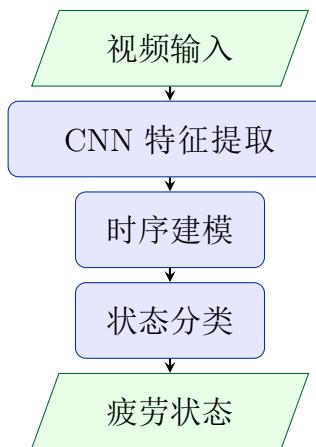


图 2: 深度学习疲劳检测流程

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。

$$\text{PERCLOS} = \frac{t_{\text{closed}}}{t_{\text{total}}} \times 100\% \quad (2)$$

其中：

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

判定标准：

- $\text{PERCLOS} < 20\%$: 清醒状态
- $20\% \leq \text{PERCLOS} < 40\%$: 轻度疲劳
- $\text{PERCLOS} \geq 40\%$: 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。

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

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。

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

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

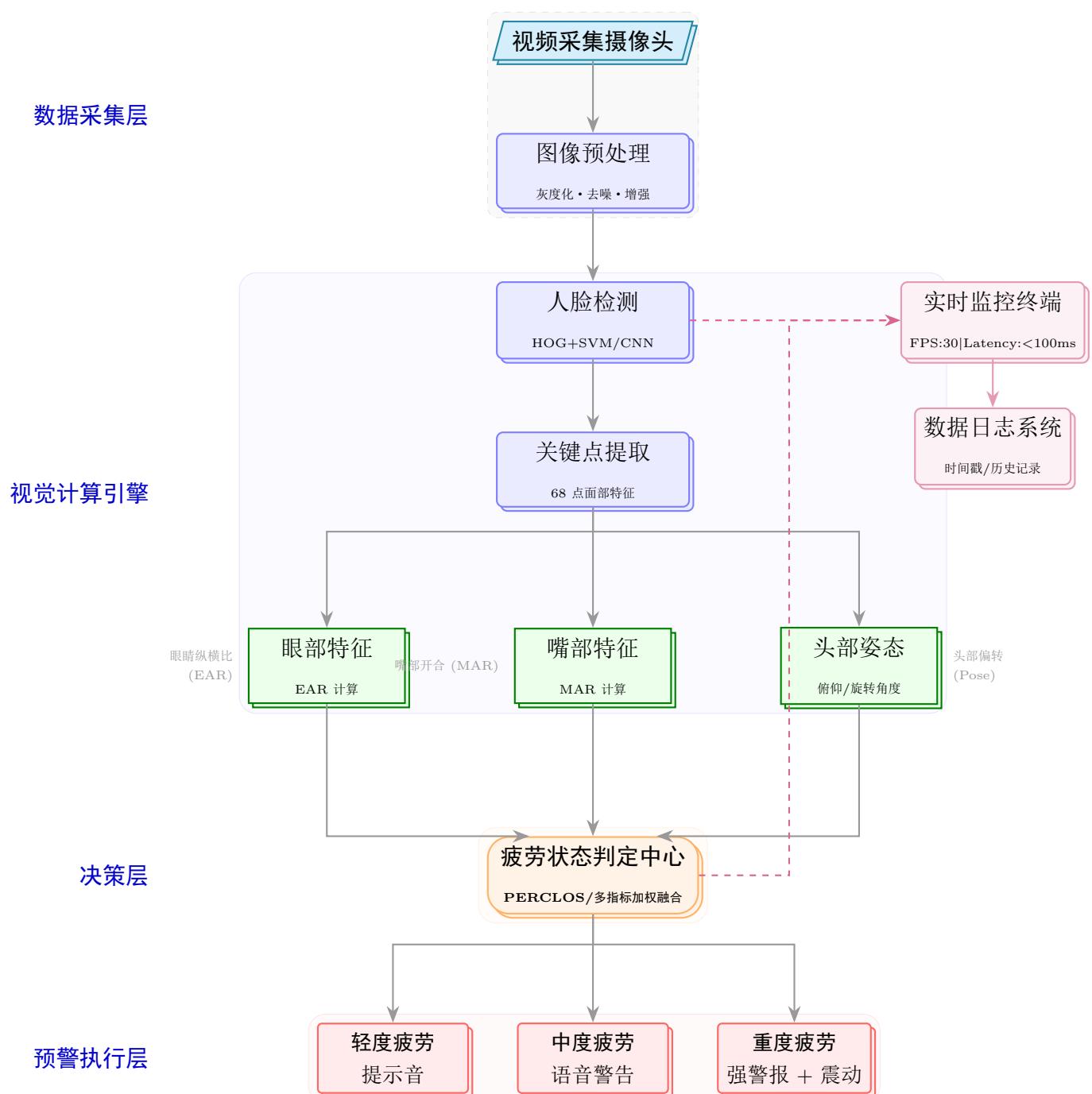


图 3: 系统整体架构

3.2 模块详细设计

3.2.1 预处理模块

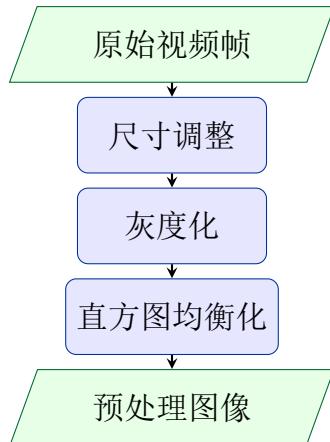


图 4: 预处理流程

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

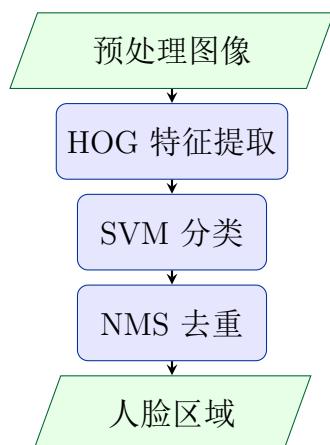


图 5: 人脸检测流程

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： $EAR \geq 0.2$, $PERCLOS < 20\%$
- 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

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

```

5 README.md          # 项目说明文档
6 detectors/
7     __init__.py
8     face_detector.py      # 人脸检测
9     landmark_detector.py  # 关键点检测
10    features/
11       __init__.py
12       ear_calculator.py   # EAR计算
13       mar_calculator.py   # MAR计算
14       pose_estimator.py   # 姿态估计(可选)
15    fatigue/
16       __init__.py
17       state_tracker.py    # 状态跟踪
18       fatigue_detector.py # 疲劳判定
19    utils/
20       __init__.py
21       visualizer.py       # 可视化工具
22       logger.py           # 日志记录
23    models/               # 模型文件目录
24       shape_predictor_68_face_landmarks.dat
25    data/                 # 测试数据
26    tests/                # 测试代码
27       test.py

```

Listing 1: 推荐的项目目录结构

3.4 接口设计

```

1 # 人脸检测模块接口
2 def detect_faces(image: np.ndarray) -> list:
3     """
4     人脸检测
5     Args:
6         image: 输入图像 (灰度图)
7     Returns:
8         人脸区域列表 [(left, top, right, bottom), ...]
9     """
10    # 关键点检测模块接口

```

```
12 def detect_landmarks(image: np.ndarray, face_rect: tuple) -> np.  
13     ndarray:  
14         """  
15             关键点检测  
16             Args:  
17                 image: 输入图像  
18                 face_rect: 人脸区域 (left, top, right, bottom)  
19             Returns:  
20                 68个关键点坐标 (68, 2)  
21             """  
22  
23 # 疲劳判定模块接口  
24 def detect_fatigue(ear_history: list, mar: float) -> dict:  
25     """  
26         疲劳判定  
27         Args:  
28             ear_history: EAR历史值列表  
29             mar: 当前MAR值  
30         Returns:  
31             判定结果 {'state': str, 'perclos': float, 'confidence': float}  
32     """  
33
```

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

```
1 # 创建虚拟环境（推荐）
2 python -m venv venv
3 source venv/bin/activate # Linux/Mac
4 # 或
5 venv\Scripts\activate # Windows
6
7 # 安装核心依赖
8 pip install opencv-python numpy scipy
9
10 # 安装Dlib
11 # 方法1：直接安装（需要编译环境）
12 pip install dlib
13
14 # 方法2：使用预编译wheel文件（Windows推荐）
15 # 从 https://pypi.org/project/dlib/#files 下载对应版本
16 pip install dlib-19.x.x-cpxx-cpxx-win_amd64.whl
17
18 # 安装其他工具
19 pip install imutils matplotlib tqdm
```

Listing 3: 安装命令

4.1.2 验证安装

```
1 import cv2
2 import dlib
3 import numpy as np
4 print(f"OpenCV版本: {cv2.__version__}")
5 print(f"Dlib版本: {dlib.__version__}")
6 print(f"NumPy版本: {np.__version__}")
7 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

```
1 import numpy as np
2
3 class EARCalculator:
4     """眼睛纵横比计算器"""
5
6     # 左眼关键点索引 (Dlib 68点模型)
7     LEFT_EYE_INDICES = [36, 37, 38, 39, 40, 41]
8     # 右眼关键点索引
9     RIGHT_EYE_INDICES = [42, 43, 44, 45, 46, 47]
10
11     @staticmethod
12     def calculate_ear(eye_points):
13         """
14             计算眼睛纵横比
15
16             参数：
17                 eye_points: 眼睛的6个关键点坐标 (N, 2)
18
19             返回：
20                 ear: 眼睛纵横比
21         """
22
23         # 计算垂直距离
24         vertical_1 = np.linalg.norm(eye_points[1] - eye_points[5])
25         vertical_2 = np.linalg.norm(eye_points[2] - eye_points[4])
26
27         # 计算水平距离
28         horizontal = np.linalg.norm(eye_points[0] - eye_points[3])
29
30         # EAR公式
31         ear = (vertical_1 + vertical_2) / (2.0 * horizontal)
32
33         return ear
34
35     @staticmethod
36     def extract_eyes(shape):
37         """
```

```
37     从 68 点模型中提取左右眼关键点
38
39     参数：
40         shape: 68 个关键点坐标 (68, 2)
41
42     返回：
43         left_eye: 左眼 6 个点
44         right_eye: 右眼 6 个点
45         """
46
47         left_eye = shape[EARCalculator.LEFT_EYE_INDICES]
48         right_eye = shape[EARCalculator.RIGHT_EYE_INDICES]
49         return left_eye, right_eye
50
51     @staticmethod
52     def compute_avg_ear(shape):
53         """
54             计算双眼平均EAR
55
56             参数：
57                 shape: 68 个关键点坐标
58
59             返回：
60                 avg_ear: 平均EAR
61                 left_ear: 左眼EAR
62                 right_ear: 右眼EAR
63         """
64
65         left_eye, right_eye = EARCcalculator.extract_eyes(shape)
66         left_ear = EARCcalculator.calculate_ear(left_eye)
67         right_ear = EARCcalculator.calculate_ear(right_eye)
68         avg_ear = (left_ear + right_ear) / 2.0
69         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

```
1 from collections import deque
2 from enum import Enum
3
4 class FatigueState(Enum):
```

```
5     """疲劳状态枚举"""
6     AWAKE = "清醒"
7     MILD = "轻度疲劳"
8     SEVERE = "重度疲劳"
9
10
11 class FatigueTracker:
12     """疲劳状态跟踪器"""
13
14     def __init__(self, window_size=60):
15         self.window_size = window_size
16         self.ear_buffer = deque(maxlen=window_size)
17         self.mar_buffer = deque(maxlen=window_size)
18
19         # 统计计数
20         self.blink_count = 0
21         self.yawn_count = 0
22
23         # 配置阈值
24         self.EAR_CLOSED = 0.15          # 闭眼阈值
25         self.EAR_BLINK = 0.2           # 眨眼阈值
26         self.MAR_YAWN = 0.5            # 打哈欠阈值
27         self.PERCLOS_MILD = 0.2        # 轻度疲劳阈值
28         self.PERCLOS_SEVERE = 0.4      # 重度疲劳阈值
29
30         # 状态标记
31         self.was_blinking = False
32         self.is_yawning = False
33
34     def update(self, ear, mar):
35         """
36             更新状态
37
38             参数：
39                 ear: 眼睛纵横比
40                 mar: 嘴部纵横比
41
42             self.ear_buffer.append(ear)
43             self.mar_buffer.append(mar)
44
```

```
45     # 检测眨眼
46
47     if ear < self.EAR_CLOSED:
48
49         if not self.was_blinking:
50
51             self.blink_count += 1
52
53             self.was_blinking = True
54
55     else:
56
57         self.was_blinking = False
58
59
60     # 检测打哈欠
61
62     if mar > self.MAR_YAWN:
63
64         if not self.is_yawning:
65
66             self.yawn_count += 1
67
68             self.is_yawning = True
69
70     else:
71
72         self.is_yawning = False
73
74
75     def get_perclos(self):
76
77         """计算PERCLOS"""
78
79         if len(self.ear_buffer) == 0:
80
81             return 0.0
82
83
84         closed_frames = sum(1 for ear in self.ear_buffer if ear <
85 self.EAR_BLINK)
86
87         return closed_frames / len(self.ear_buffer)
88
89
90     def get_state(self):
91
92         """获取当前疲劳状态"""
93
94         perclos = self.get_perclos()
95
96
97         if perclos >= self.PERCLOS_SEVERE or self.yawn_count >= 3:
98             return FatigueState.SEVERE
99
100        elif perclos >= self.PERCLOS_MILD:
101            return FatigueState.MILD
102
103        else:
104            return FatigueState.AWAKE
105
106
107    def get_statistics(self):
108
109        """获取统计信息"""
110
111        return {
112
113            "perclos": self.get_perclos(),
```

```
84         "blink_count": self.blink_count,
85         "yawn_count": self.yawn_count,
86         "avg_ear": np.mean(self.ear_buffer) if self.ear_buffer
87     else 0.0,
88     "avg_mar": np.mean(self.mar_buffer) if self.mar_buffer
89     else 0.0
90 }
```

Listing 6: 疲劳状态跟踪器

4.2.3 完整检测流程实现

```
1 import cv2
2 import dlib
3 import numpy as np
4 from imutils import face_utils
5 from detectors.face_detector import FaceDetector
6 from detectors.landmark_detector import LandmarkDetector
7 from features.ear_calculator import EARCalculator
8 from fatigue.state_tracker import FatigueTracker
9 from utils.visualizer import Visualizer
10
11 class FatigueDetector:
12     """疲劳检测系统"""
13
14     def __init__(self, model_path: str, config: dict = None):
15         """
16             初始化检测系统
17
18             Args:
19                 model_path: 68点模型文件路径
20                 config: 配置参数
21
22             # 初始化人脸检测器
23             self.face_detector = FaceDetector()
24
25             # 初始化关键点检测器
26             self.landmark_detector = LandmarkDetector(model_path)
27
28             # 初始化疲劳跟踪器
29             self.fatigue_tracker = FatigueTracker()
```

```
29         window_size = config.get('window_size', 60) if config else 60
30         self.tracker = FatigueTracker(window_size=window_size)
31
32     # 初始化可视化器
33     self.visualizer = Visualizer()
34
35     # 配置参数
36     self.config = config or {}
37
38 def process(self, frame: np.ndarray) -> tuple:
39     """
40     处理单帧图像
41
42     Args:
43         frame: 输入图像 (BGR格式)
44
45     Returns:
46         (result_image, fatigue_state, statistics)
47     """
48
49     # 预处理
50     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
51
52     # 人脸检测
53     faces = self.face_detector.detect(gray)
54
55     for face in faces:
56         # 关键点检测
57         shape = self.landmark_detector.detect(gray, face)
58         shape = face_utils.shape_to_np(shape)
59
60         # 计算EAR
61         avg_ear, left_ear, right_ear = EARCalculator.
62         compute_avg_ear(shape)
63
64         # 计算MAR
65         mouth = shape[48:68]
66         mar = self.calculate_mar(mouth)
67
68         # 更新跟踪器
69         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,
76     stats)
77
78     return frame, state, stats
79
80
81     def calculate_mar(self, mouth_points):
82         """计算嘴部纵横比"""
83         A = np.linalg.norm(mouth_points[2] - mouth_points[10]) #
66-62
84         B = np.linalg.norm(mouth_points[4] - mouth_points[8]) #
64-60
85         C = np.linalg.norm(mouth_points[0] - mouth_points[6]) #
48-54
86         mar = (A + B) / (2.0 * C)
87
88     return mar
89
90
91     # 主程序
92     def main():
93         """主函数"""
94         # 配置参数
95         config = {
96             'window_size': 60,
97             'frame_width': 640,
98             'frame_height': 480
99         }
100
101
102         # 初始化检测器
103         detector = FatigueDetector(
104             model_path="models/shape_predictor_68_face_landmarks.dat",
105             config=config
106         )
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
823
824
825
825
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1001
1002
1003
```

```
104 # 打开摄像头
105 cap = cv2.VideoCapture(0)
106
107 while True:
108     ret, frame = cap.read()
109     if not ret:
110         break
111
112     # 调整尺寸
113     frame = cv2.resize(frame, (config['frame_width'], config['
114 frame_height']))
115
116     # 检测疲劳
117     result, state, stats = detector.process(frame)
118
119     # 显示结果
120     cv2.imshow("Fatigue Detection", result)
121
122     # 按ESC退出
123     if cv2.waitKey(1) & 0xFF == 27:
124         break
125
126     cap.release()
127     cv2.destroyAllWindows()
128
129 if __name__ == "__main__":
130     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

```

1 # config.py
2
3 """
4 疲劳检测系统配置文件
5 """
6
7 # 人脸检测配置
8 FACE_DETECTION_CONFIG = {
9     'scale_factor': 1.0,          # 图像缩放因子
10    'min_neighbors': 5,           # 最小邻居数
11    'min_size': (30, 30),        # 最小人脸尺寸
12}
13
14 # 特征计算配置
15 FEATURE_CONFIG = {
16    'ear_threshold': 0.2,         # EAR 阈值
17    'mar_threshold': 0.5,          # MAR 阈值
18    'ear_closed': 0.15,          # 闭眼判定阈值

```

```
19 }
20
21 # 疲劳判定配置
22 FATIGUE_CONFIG = {
23     'window_size': 60,          # 时间窗口大小(帧)
24     'perclos_mild': 0.2,       # 轻度疲劳阈值
25     'perclos_severe': 0.4,     # 重度疲劳阈值
26     'yawn_trigger_count': 3,   # 打哈欠告警次数
27 }
28
29 # 可视化配置
30 VISUALIZATION_CONFIG = {
31     'show_landmarks': True,    # 显示关键点
32     'show_status': True,       # 显示状态信息
33     'show_ear': True,          # 显示EAR值
34     'show_mar': True,          # 显示MAR值
35 }
```

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-dataset
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/datasets.html
CEW	闭眼检测数据集	https://opendatalab.com/OpenDataLab/CEW_Closed_Eyes_in_the_Wild/

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-detection-dataset>
 - 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-drowsiness-dataset>

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-drowsiness-dataset>

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)

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

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

```
1 import os
2 import time
3 import cv2
4 import numpy as np
5 from typing import Dict, List
6 import pandas as pd
7
8 class Evaluator:
9     """性能评估器"""
10
11     def __init__(self, detector):
12         """
13             初始化评估器
14
15         Args:
16             detector: 疲劳检测器实例
17         """
18         self.detector = detector
19
20     def evaluate(self, test_dir: str, ground_truth: Dict[str, str]) -> Dict:
21         """
22             评估检测性能
23
24         Args:
25             test_dir: 测试视频目录
26             ground_truth: 真实标签 {filename: state}
27
28         Returns:
```

```
29     评估结果字典
30
31     results = {
32         'total': 0,
33         'correct': 0,
34         'tp': 0,    # True Positive
35         'tn': 0,    # True Negative
36         'fp': 0,    # False Positive
37         'fn': 0,    # False Negative
38         'processing_times': [],
39         'errors': []
40     }
41
42     for filename, true_state in ground_truth.items():
43         video_path = os.path.join(test_dir, filename)
44
45         if not os.path.exists(video_path):
46             results['errors'].append(f"{filename}: 文件不存在")
47             continue
48
49         # 处理视频
50         cap = cv2.VideoCapture(video_path)
51         frame_predictions = []
52
53         while True:
54             ret, frame = cap.read()
55             if not ret:
56                 break
57
58             # 记录处理时间
59             start_time = time.time()
60             _, pred_state, _ = self.detector.process(frame)
61             elapsed = (time.time() - start_time) * 1000
62             results['processing_times'].append(elapsed)
63
64             frame_predictions.append(pred_state.value)
65
66         cap.release()
67
68         # 投票决定最终预测
```

```
69         if frame_predictions:
70             pred_state = max(set(frame_predictions),
71                               key=frame_predictions.count)
72
73             results['total'] += 1
74
75             # 统计混淆矩阵
76             if true_state == "疲劳":
77                 if pred_state == "重度疲劳" or pred_state == "轻
78                 度疲劳":
79                     results['tp'] += 1
80                     results['correct'] += 1
81                 else:
82                     results['fn'] += 1
83             else: # 清醒
84                 if pred_state == "清醒":
85                     results['tn'] += 1
86                     results['correct'] += 1
87                 else:
88                     results['fp'] += 1
89
90             # 计算各项指标
91             results['accuracy'] = (
92                 results['correct'] / results['total'] * 100
93                 if results['total'] > 0 else 0
94             )
95
96             results['precision'] = (
97                 results['tp'] / (results['tp'] + results['fp']) * 100
98                 if (results['tp'] + results['fp']) > 0 else 0
99             )
100
101            results['recall'] = (
102                 results['tp'] / (results['tp'] + results['fn']) * 100
103                 if (results['tp'] + results['fn']) > 0 else 0
104             )
105
106            results['f1'] = (
107                 2 * results['precision'] * results['recall'] /
108                 (results['precision'] + results['recall']))
```

```
108         if (results['precision'] + results['recall']) > 0 else 0
109     )
110
111     results['avg_fps'] = (
112         1000 / np.mean(results['processing_times']))
113     if results['processing_times'] else 0
114 )
115
116     return results
117
118 def generate_report(self, results: Dict, output_path: str = 'report.txt'):
119     """生成评估报告"""
120     report = f"""
121 =====
122 疲劳检测系统性能评估报告
123 =====
124
125 一、 总体指标
126 -----
127 测试样本总数: {results['total']}
128 正确分类数: {results['correct']}
129
130 二、 分类指标
131 -----
132 准确率 (Accuracy): {results['accuracy']:.2f}%
133 精确率 (Precision): {results['precision']:.2f}%
134 召回率 (Recall): {results['recall']:.2f}%
135 F1分数: {results['f1']:.2f}%
136
137 三、 混淆矩阵
138 -----
139             预测清醒      预测疲劳
140 真实清醒          {results['tn']}          {results['fp']}
141 真实疲劳          {results['fn']}          {results['tp']}
142
143 四、 性能指标
144 -----
145 平均帧率: {results['avg_fps']:.2f} FPS
146
```

```
147 五、错误样本
148 -----
149 """
150
151     if results['errors']:
152         for error in results['errors']:
153             report += f"- {error}\n"
154     else:
155         report += "无错误\n"
156
157 report += "=====\\n"
158
159 # 保存报告
160 with open(output_path, 'w', encoding='utf-8') as f:
161     f.write(report)
162
163 print(report)
164 return report
165
166
167 # 使用示例
168 if __name__ == '__main__':
169     from main import FatigueDetector
170
171     # 创建检测器
172     detector = FatigueDetector("models/
shape_predictor_68_face_landmarks.dat")
173
174     # 创建评估器
175     evaluator = Evaluator(detector)
176
177     # 定义真实标签
178     ground_truth = {
179         'test001.mp4': '清醒',
180         'test002.mp4': '疲劳',
181         'test003.mp4': '清醒',
182         # ... 更多测试样本
183     }
184
185     # 运行评估
```

```
186 results = evaluator.evaluate('./data/test', ground_truth)  
187  
188 # 生成报告  
189 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

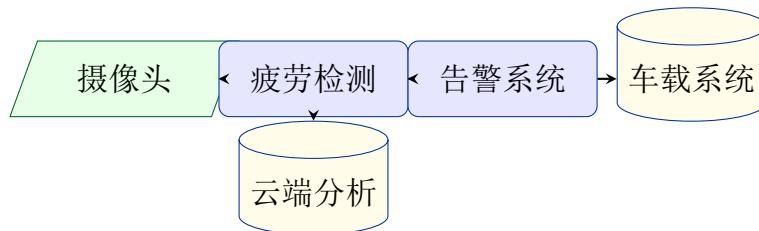


图 6: 车载系统集成架构

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
<code>cv2.VideoCapture()</code>	打开视频文件或摄像头
<code>cv2.cvtColor()</code>	颜色空间转换
<code>cv2.resize()</code>	调整图像尺寸
<code>cv2.equalizeHist()</code>	直方图均衡化
<code>cv2.GaussianBlur()</code>	高斯滤波
<code>cv2.rectangle()</code>	绘制矩形
<code>cv2.putText()</code>	绘制文字
<code>cv2.polylines()</code>	绘制多边形
<code>cv2.waitKey()</code>	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
<code>dlib.get_frontal_face_detector()</code>	获取人脸检测器
<code>dlib.shape_predictor()</code>	加载关键点检测模型
<code>detector(gray, 0)</code>	检测人脸，返回人脸区域列表
<code>predictor(gray, face)</code>	检测面部关键点，返回 68 个点
<code>face_utils.shape_to_np()</code>	将 shape 转换为 NumPy 数组
<code>face_utils.rect_to_bb()</code> 将人脸矩形转换为边界框	

C 附录 C: 完整代码框架

```

1 """
2 疲劳检测系统主程序
3 """
4
5 import cv2
6 import dlib
7 import numpy as np
8 import argparse

```

```
9 from pathlib import pathlib
10
11 from detectors.face_detector import FaceDetector
12 from detectors.landmark_detector import LandmarkDetector
13 from features.ear_calculator import EARCalculator
14 from fatigue.state_tracker import FatigueTracker
15 from utils.visualizer import Visualizer
16
17
18 def parse_args():
19     """解析命令行参数"""
20     parser = argparse.ArgumentParser(description='疲劳检测系统')
21     parser.add_argument('--camera', type=int, default=0, help='摄像头设备ID')
22     parser.add_argument('--model', type=str,
23                         default='models/
shape_predictor_68_face_landmarks.dat',
24                         help='模型文件路径')
25     parser.add_argument('--width', type=int, default=640, help='视频宽度')
26     parser.add_argument('--height', type=int, default=480, help='视频高度')
27     parser.add_argument('--window', type=int, default=60, help='时间窗口大小')
28
29     return parser.parse_args()
30
31
32 def main():
33     """主函数"""
34     args = parse_args()
35
36     # 检查模型文件
37     model_path = pathlib.Path(args.model)
38     if not model_path.exists():
39         print(f"错误：模型文件不存在 - {args.model}")
40         return
41
42     # 初始化模块
43     print("初始化检测器...")
44     face_detector = FaceDetector()
```

```
44 landmark_detector = LandmarkDetector(str(model_path))
45 fatigue_tracker = FatigueTracker(window_size=args.window)
46 visualizer = Visualizer()
47
48 # 打开摄像头
49 print(f"打开摄像头 {args.camera}...")
50 cap = cv2.VideoCapture(args.camera)
51
52 if not cap.isOpened():
53     print("错误：无法打开摄像头")
54     return
55
56 cap.set(cv2.CAP_PROP_FRAME_WIDTH, args.width)
57 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, args.height)
58
59 print("开始检测（按ESC退出）...")
60
61 while True:
62     ret, frame = cap.read()
63     if not ret:
64         break
65
66     # 预处理
67     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
68
69     # 人脸检测
70     faces = face_detector.detect(gray)
71
72     for face in faces:
73         # 关键点检测
74         shape = landmark_detector.detect(gray, face)
75         shape = shape.astype(np.int32)
76
77         # 计算EAR
78         avg_ear, left_ear, right_ear = EARCalculator.
79         compute_avg_ear(shape)
80
81         # 计算MAR
82         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
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