驾驶行为检测系统技术实现说明

系统概述

该系统是一个基于深度学习的驾驶行为检测系统，能够检测图片和视频中的驾驶行为，如安全驾驶、疲劳驾驶、分心驾驶、玩手机等。系统采用前后端分离架构，前端使用纯HTML/CSS/JavaScript实现，后端使用Python的Flask框架构建REST API服务。

技术架构

前端技术栈

- HTML5 + CSS3 + JavaScript
- Chart.js：用于数据可视化
- Font Awesome：提供图标支持

后端技术栈

- Python 3.x
- Flask：Web框架
- Flask-CORS：处理跨域请求
- Ultralytics YOLO：对象检测模型
- OpenCV (cv2)：图像和视频处理
- PIL (Pillow)：图像处理
- FFmpeg：视频格式转换（可选）

数据流程

整个系统的数据流程如下：

1. 用户在前端上传图片或视频
2. 前端将文件通过HTTP请求发送到后端API
3. 后端接收文件并保存到临时目录
4. 后端加载YOLO模型并对图片/视频进行处理
5. 后端将处理结果返回给前端

6. 前端展示检测结果、统计信息和可视化图表

关键技术实现

1. 文件上传与传输

前端实现

前端通过 `FormData` 对象封装上传的文件，并使用 `fetch` API发送到后端：

```
const formData = new FormData();
formData.append('file', selectedFile);

const response = await fetch(`${API_BASE_URL}${endpoint}`, {
    method: 'POST',
    body: formData
});
```

后端实现

后端使用Flask的 `request.files` 接收上传的文件：

```
file = request.files['file']
file_id = str(uuid.uuid4())  # 生成唯一ID
input_path = os.path.join(UPLOAD_FOLDER, f"{file_id}.jpg/mp4")
file.save(input_path)  # 保存文件
```

2. 图像处理与目标检测

模型加载

系统使用Ultralytics YOLO模型进行目标检测：

```
def load_model():
    global model_car_inside_detection
    try:
        model_car_inside_detection = YOLO(r'.\models\car_inside_detect.pt')
        return True
    except Exception as e:
        print(f"模型加载失败: {e}")
        return False
```

图像处理流程

1. 加载保存的图像
2. 使用YOLO模型进行预测
3. 将预测结果绘制到原图上
4. 提取检测到的对象信息、置信度等
5. 将标注后的图像保存并返回URL

```
# 使用模型进行预测
results = model(input_path, conf=0.25, iou=0.45)

# 保存标注后的图片
for r in results:
    im_array = r.plot()  # 获取绘制后的图像数组
    im = Image.fromarray(im_array[..., ::-1])  # RGB to BGR
    im.save(output_path)
```

3. 视频处理技术

视频处理是系统的核心难点，主要涉及以下技术：

视频帧提取与处理

使用OpenCV逐帧读取视频并处理：

```
cap = cv2.VideoCapture(input_path)
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # 保存当前帧为临时图像
    temp_frame_path = os.path.join(temp_dir, f"frame_{frame_count}.jpg")
    cv2.imwrite(temp_frame_path, frame)

    # 使用YOLO模型进行预测
    results = model(temp_frame_path, conf=0.25, iou=0.45)

    # 处理检测结果
    # ...
```

连续行为分析

系统实现了对驾驶行为的连续跟踪和分析：

```
# 检查是否连续检测到相同类别
if len(class_history) == consecutive_threshold:
    # 检查是否所有元素都相同
    if all(cls == class_history[0] for cls in class_history) and class_history[0] != "未知":
        current_confirmed_class = class_history[0]
```

视频生成

处理完所有帧后，使用OpenCV的VideoWriter将结果合成为新视频：

```
# 创建视频写入器
fourcc = cv2.VideoWriter_fourcc(*'avc1')
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

# 写入处理后的帧
out.write(annotated_frame)
```

视频格式转换

为提高浏览器兼容性，系统支持使用FFmpeg将视频转换为WebM格式：

```
command = [
    ffmpeg_cmd, '-i', input_abs_path,
    '-c:v', 'libvpx-vp9', '-crf', '30', '-b:v', '0',
    output_abs_path
]
result = subprocess.run(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
```

4. 实时进度反馈

系统实现了视频处理进度的实时反馈机制：

后端实现

后端维护一个全局字典存储处理进度：

```
processing_progress = {}

# 更新处理进度
current_progress = frame_count / total_frames
processing_progress[file_name] = current_progress
```

前端实现

前端定期轮询后端获取进度：

```
function startProgressCheck(fileName) {
    progressCheckInterval = setInterval(async () => {
        const response = await fetch(`${API_BASE_URL}/progress?filename=${encodeURIComponent(fi
        const data = await response.json();

        if (data.progress !== undefined) {
            const progressPercent = Math.round(data.progress * 100);
            progressBarProcessing.style.width = `${progressPercent}%`;
            progressBarProcessing.setAttribute('data-progress', `${progressPercent}%`);
        }
    }, 1000);
}
```

5. 结果展示与可视化

统计信息分析

系统会分析检测结果并生成统计信息：

```
# 计算各类别统计信息
cls_stat = class_stats[mapped_cls_name]
cls_stat['count'] += 1
cls_stat['conf_sum'] += conf
cls_stat['max_conf'] = max(cls_stat['max_conf'], conf)
cls_stat['min_conf'] = min(cls_stat['min_conf'], conf)
```

数据可视化

前端使用Chart.js创建交互式图表：

```
categoryCountChart = new Chart(countCtx, {
    type: 'bar',
    data: {
        labels: labels,
        datasets: [{
            label: '检测数量',
            data: counts,
            backgroundColor: primaryColor,
            borderColor: primaryColor,
            borderWidth: 1,
            borderRadius: 4
        }]
    },
    // 配置选项...
});
```

关键算法

行为连续检测算法

系统采用滑动窗口方法检测持续的驾驶行为：

1. 维护一个固定长度的类别历史记录队列

2. 每处理一帧，将当前帧检测到的最可能类别加入队列
3. 如果队列已满，移除最早的记录
4. 检查队列中是否所有元素都是相同类别
5. 如果是，则确认当前行为类别

```
class_history.append(main_class)
if len(class_history) > consecutive_threshold:
    class_history.pop(0)

if len(class_history) == consecutive_threshold:
    if all(cls == class_history[0] for cls in class_history) and class_history[0] != "未知":
        current_confirmed_class = class_history[0]
```

类别映射算法

系统实现了类别映射，将相似的检测类别合并：

```
def map_class_name(cls_name):
    """
    合并特定类别到指定类别
    """
    if cls_name.startswith("与乘客交谈"):
        return "安全驾驶"
    return cls_name
```

持续时间计算

系统计算每种驾驶行为的持续时间：

```
# 计算每个类别的连续帧段
segments = []
segment_start = frames[0]
prev_frame = frames[0]

for frame in frames[1:]:
    # 如果与前一帧不连续(差距大于1)，则创建新段
    if frame > prev_frame + 1:
        segments.append((segment_start, prev_frame))
        segment_start = frame
    prev_frame = frame

# 添加最后一个段
segments.append((segment_start, prev_frame))

# 计算总帧数并转换为时间
duration_seconds = (segment_data['total_frames'] / fps) if fps > 0 else 0
```

系统优化

内存管理

为避免处理大型视频时内存溢出，系统实现了分批处理策略：

```
# 为了减轻内存压力，可以定期处理帧并写入视频
if frame_count % 100 == 0 or frame_count == total_frames - 1:
    # 处理并写入视频文件
    for i in range(max(0, frame_count-100), frame_count):
        if i in frame_results:
            # 处理当前帧
            # ...
            # 写入处理后的帧
            out.write(annotated_frame)
            # 从字典中移除已写入的帧，释放内存
            del frame_results[i]
```

文件管理

系统会自动清理临时文件，只保留最新的N个文件：

```python
def cleanup_folder(folder_path, max_files=MAX_FILES_PER_FOLDER):
    # 获取文件夹中所有文件
    files = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if os.path.isfile(os

    # 按文件修改时间排序
    files.sort(key=lambda x: os.path.getmtime(x))

    # 如果文件数量超过最大值，删除最旧的文件
    if len(files) > max_files:
        for f in files[:-max_files]:
            os.remove(f)
```

部署注意事项

1. 确保安装所有必要的Python依赖包
2. 模型文件应放置在正确的目录中（./models/car_inside_detect.pt）
3. 如需使用视频格式转换功能，请安装FFmpeg并设置正确的路径
4. 确保上传和输出目录具有正确的读写权限
5. 端口冲突时，系统会自动寻找可用端口

结论

这个驾驶行为检测系统通过整合多种技术，实现了对图片和视频中驾驶行为的智能检测与分析。系统架构清晰，前后端分离，技术路线合理，可靠性高，具有良好的用户体验和扩展性。