DrowsiSense: A Real-Time Drowsiness Detection System

PROJECT FOR 20XD68 - DEEP LEARNING LAB

Chinnahyata Poorvika (22PD10)
Sarnika Sanjiv Kumar (22PD31)

Table of Contents

1. Introduction

1.1 Motivation and Background

Driver drowsiness is a leading contributor to road accidents worldwide. Fatigue reduces reaction times and awareness, increasing the risk of collisions. DrowsiSense was conceived to address this critical issue by developing a system that monitors driver alertness in real time and offers timely warnings when signs of drowsiness are detected. The project is built upon well-established computer vision techniques and advances in deep learning, targeting a reduction in fatigue-induced incidents.

1.2 Project Objectives and Scope

The primary objectives of the DrowsiSense project are:

- **Real-Time Monitoring:** Capture and process video from a live feed (e.g., a webcam) to track driver facial features.
- **Accurate Drowsiness Detection:** Use robust metrics like the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), supplemented by a Convolutional Neural Network (CNN), to reliably determine drowsiness.
- **Immediate Alerts:** Generate audio and visual warnings promptly to alert the driver.
- **Modular and Extensible Design:** Maintain a code structure that supports easy upgrades and enhancements.

The scope of this project spans from a proof-of-concept implementation to a potential deployable driver-assistance solution that could be integrated with various vehicle systems.

2. Theoretical Background

2.1 Computer Vision and Deep Learning Approaches

Computer vision has been successfully applied to numerous real-time detection tasks in safety applications. The DrowsiSense project leverages classical techniques—such as Haar Cascades and the dlib facial landmark detector—alongside modern deep learning methods. Convolutional Neural Networks (CNNs) automatically learn discriminative features from raw images, enhancing the accuracy of drowsiness detection and mitigating false positives that can occur when relying solely on heuristic thresholds.

3. System Overview and Architecture

3.1 Overall System Description

DrowsiSense is composed of several integrated components:

- **Video Capture:** Capturing frames from a live feed from a webcam.
- **Facial Landmark Detection:** Utilizing a pre-trained facial landmark detector to extract crucial points on the face.
- **Metric Computation:** Calculation of EAR and MAR, which serve as quantitative measures of eye closure and yawning.
- **CNN Verification:** Supplementing heuristic calculations with a CNN-based classification for enhanced reliability.
- **Alert Mechanisms:** Immediate triggering of visual and audible alerts if drowsiness is detected.
- **Data Visualization:** Real-time plotting of metrics to provide performance feedback.

3.2 Hardware and Software Requirements

Hardware Requirements:

- A webcam or video capturing device.

Software Requirements:

- **Libraries:**
    - OpenCV for image and video processing.
    - dlib for facial detection and landmark extraction.
    - TensorFlow and Keras for deep learning.
    - pygame for audio alert management.
    - streamlit for a web-based user interface.
    - Additional libraries such as numpy, imutils, matplotlib, and collections.
- **Model and Dataset Files:**
    - `Shape_predictor_68_face_landmarks.dat`
    - Trained model file: `drowsiness_cnn_model.h5`

- Alarm audio file: `alarm.wav`
- Training and validation datasets stored in a directory structure.

4. Detailed Design and Code Components

4.1 Main Application (app.py)

The **app.py** file contains the application's primary workflow. It integrates video capture, real-time processing, alert generation, and data visualization using a streamlit interface. Below is an explanation of its primary sections and code segments.

4.1.1 Module Imports and Initial Setup

- **cv2, dlib, imutils, face_utils:** For video processing and facial landmark detection.
- **load_model:** To load the pre-trained CNN.
- **pygame:** For handling audio alerts.
- **streamlit:** To create a web-based UI.
- **tempfile, os, time:** For file handling, path management, and time-based computations.
- **deque:** To maintain a buffer for smoothing EAR values.
- **matplotlib.pyplot:** Used to plot performance metrics such as EAR and MAR over time.

4.1.2 Audio Alert Initialization

This is critical for managing the playback of the alert audio (`alarm.wav`).

4.1.3 Streamlit Configuration

Sidebars allow users to adjust various thresholds and parameters, such as frame thresholds, EAR/MAR thresholds, and CNN probability thresholds.

4.1.4 Helper Functions for Metric Calculation

The EAR and MAR are computed using helper functions:

- **calculate_ear:** Uses Euclidean distances between specific eye landmarks.

- **calculate_mar:** Computes the mouth aspect ratio based on vertical and horizontal distances between mouth landmarks.

4.1.5 Alarm Sound Handling

The functions `play_alarm()` and `stop_alarm()` control the audio alerts. These functions ensure that the alarm sounds are played and stopped based on detection results.

4.1.6 Streamlit Sidebar Controls

The sidebar is used to let the user change detection parameters dynamically.

4.1.7 Loading Models

Using Streamlit's caching mechanism, models (face detector, shape predictor, and CNN) are loaded. This avoids re-loading the models on every re-run.

4.1.8 The DrowsinessDetector Class

This class encapsulates the detection logic:

- **State Management:**
  It stores history data (EAR, MAR, detection events) in deques for smoothing and plotting.

- **Detection Logic:**
  The `detect_drowsiness()` method processes each frame:

  - Converts the frame to grayscale.
  - Detects faces via OpenCV's Haar cascades.
  - Uses dlib to extract 68 facial landmarks.
  - Computes EAR and MAR.
  - Applies time-based threshold logic to detect prolonged eye closures.
  - Implements yawning detection by counting consecutive frames where MAR exceeds the threshold.
  - Optionally runs a CNN prediction on the region-of-interest (ROI) to validate drowsiness.

- **Plotting Functions:**
  Two functions, `create_ratio_plot()` and `create_detection_plot()`, generate real-time plots of the EAR/MAR ratios and detection events, respectively.

4.1.9 The Main Processing Loop

The main function sets up the user interface, selects the video source, and begins processing frames in a loop. Here, the application continuously updates the video feed and plots while monitoring for drowsiness.

4.2 Convolutional Neural Network Module (cnn.py)

The **cnn.py** file defines the architecture and training process of the CNN used to support the primary heuristic detection.

4.2.1 Data Preprocessing and Directory Setup

Paths for the training and test datasets are defined. The `ImageDataGenerator` is used to perform rescaling. Data generators for the training and validation sets are then prepared.

4.2.2 CNN Model Architecture

The CNN is built using the Sequential API. This architecture comprises multiple convolutional layers with Batch Normalization, pooling, and dropout for regularization. The final Dense layer produces probabilities over four classes, which might indicate different states (e.g., alert, drowsy, yawning, or another state).

4.2.3 Compilation, Training, and Saving

The model is compiled with a low learning rate to carefully adjust weights. Finally, the trained model is saved for later use in the detection system.

4.3 Auxiliary Files and Resources

The project uses several auxiliary resources, including:

- **Facial Landmark Predictor:**
  `shape_predictor_68_face_landmarks.dat` is essential for detecting facial landmarks used in computing EAR and MAR.

- **Alarm Audio File:**
  `alarm.wav` provides the audible alert for drowsiness.

- **Dataset Directory:**
  Located in `dataset_new/`, this directory contains the images used for training and validation of the CNN model.

- **Dependency Management:**
  A `requirements.txt` file enumerates all required Python libraries to set up the environment.

5. Algorithmic Foundations

5.1 Facial Landmark Detection and ROI Extraction

The detection process begins by converting each video frame to grayscale and detecting faces using OpenCV's Haar cascades. With each detected face, the dlib shape predictor identifies 68 key landmarks. These landmarks are used to segment the region of interest (the eyes and mouth) for further analysis.

5.2 Calculation of EAR and MAR

- **Eye Aspect Ratio (EAR):**
  The EAR is computed from the distances between six landmarks surrounding each eye. A drop in EAR indicates prolonged eye closure—a primary indicator of drowsiness.

- **Mouth Aspect Ratio (MAR):**
  MAR is calculated from landmarks around the mouth. An increased MAR is associated with yawning. Both metrics are smoothed using a deque buffer to avoid false triggers from transient facial movements.

5.3 CNN Architecture and Training Process

The CNN, detailed in **cnn.py**, is designed to classify the facial state by learning feature representations from face images. Training involves several convolutional layers that extract features, followed by batch normalization, dropout for regularization, and fully connected layers culminating in a softmax classifier. This model reinforces the heuristic-based detection by confirming drowsiness when the predicted probability exceeds a set threshold.

6. Implementation Details and Integration

6.1 Real-Time Video Processing and Data Visualization

The system processes video frames in near real time. After detection:

- The application overlays polyline drawings on the detected eyes and mouth to visualize landmarks.

- It continuously updates plots of EAR/MAR ratios and detection events, providing immediate visual feedback of system performance. These plots are generated using matplotlib and updated periodically.

6.2 Alarm and Alert Mechanisms

When the processed metrics indicate drowsiness (either by threshold criteria or CNN prediction), the system triggers:

- **Visual Alerts:**
  Text overlays on the video feed indicate "DROWSY" status.

- **Audio Alerts:**
  The audio file (`alarm.wav`) is played in a loop, controlled via pygame, until the driver's state returns to normal.

7. Experimental Evaluation

7.1 Dataset and Training Details

The training dataset stored in `dataset_new/train` contains labeled images that illustrate various states (alert, drowsy, yawning, etc.). The validation data (or test set) is used to evaluate the model's accuracy, precision, recall, and overall robustness. Data preprocessing via `ImageDataGenerator` standardizes image inputs and scales pixel values.

7.2 Performance Metrics and Model Validation

During training, the model is evaluated against metrics such as:

- **Accuracy:** The percentage of correct predictions.

- **Loss:** Categorical cross-entropy loss that is minimized during training.

- **Confusion Matrix:** To understand the classification errors among the four classes. Experimental evaluation of the integrated detection system also involves visual inspection of the plotted EAR, MAR, and detection events over time.

8. Strengths, Limitations, and Challenges

8.1 Key Strengths

- **Multi-Layered Detection:**
  Combining classical computer vision techniques with a CNN improves detection reliability.

- **Real-Time Feedback:**
  The system's ability to generate live video feedback and plots allows for easy monitoring and debugging.

- **Flexibility and Extensibility:**
  The modular architecture permits quick integration of additional features or improvements, such as advanced sensor data or new deep learning models.

8.2 Limitations and Potential Challenges

- **Environmental Variability:**
  Factors such as low lighting or occluded facial features can reduce detection accuracy.

- **Processing Overhead:**
  The requirement for real-time processing may demand higher computational resources.

- **Data-Driven Performance:**
  The system's performance is dependent on the quality and diversity of the training data.

9. Future Work and Recommendations

Future enhancements to DrowsiSense could include:

- **Enhanced Training Data:**
  Expanding the dataset to cover more diverse driver profiles and conditions.

- **Advanced Sensor Fusion:**
  Integrating non-visual data (e.g., infrared images) to improve performance in adverse environments.

- **Adaptive Thresholding:**
  Developing algorithms that adjust detection thresholds in real time based on the driver's habits.

- **Edge Processing:**
  Optimizing the pipeline to run efficiently on embedded systems or low-powered devices.

10. Conclusion

DrowsiSense is a comprehensive system that leverages real-time video processing, robust facial landmark detection, and state-of-the-art CNN models for drowsiness detection. By integrating multiple modalities and providing instant alerts through visual and audio feedback, the system aims to enhance road safety. Although there remain challenges such as processing speed and environmental variability, the project lays a solid foundation for further research and potential real-world applications. Its modular design ensures that the system can evolve with new technological advancements and improved training data.

11. References

1. Research articles and technical documentation on computer vision for facial landmark detection (e.g., dlib and OpenCV documentation).
2. Studies on driver fatigue and the quantitative analysis of eye and mouth movements.
3. Tutorials and documentation on TensorFlow and Keras for designing and training CNNs.
4. Streamlit documentation for building dynamic web-based applications.
5. Publications on real-time embedded systems for driver assistance technologies.