Drowsiness of Detection using Face Recognition and Deep Learning

Jayanth Kande
jaiykool9@gmail.com
https://doi.org/10.34293/ICETCS2024.ch001

Abstract - *Drowsiness is a significant contributor to road traffic accidents, often leading to catastrophic outcomes. Existing drowsiness detection solutions heavily rely on intricate algorithms and costly hardware, such as Electroencephalography (EEG) for monitoring brain activity and Electrocardiography (ECG) for measuring heart rates. However, these systems are often impractical for use in dynamic driving environments due to their complexity and lack of user-friendliness. This study explores an alternative approach that leverages facial behavior analysis, including eye and mouth movements, to develop a more accessible and efficient drowsiness detection mechanism using camera-based systems. It also addresses challenges such as variable lighting conditions and the driver's head orientation to enhance detection accuracy. By analyzing existing methodologies and integrating video or webcam technologies, this work aims to propose a robust framework for real-time drowsiness monitoring.*

INTRODUCTION

A. About the Project

Detecting drowsiness in drivers plays a vital role in enhancing automobile safety by identifying when a driver is fatigued and at risk of causing accidents. Numerous studies indicate that driver fatigue is a significant contributor to road incidents, accounting for 20% of all traffic collisions and as much as 50% of accidents on highways. Alarmingly, statistics suggest that fatigued driving leads to approximately 1,200 fatalities and over 76,000 injuries annually. As a result, developing systems that can detect or prevent driver fatigue is becoming an urgent challenge for accident-avoidance technologies. Sleepiness can significantly impair driving ability, which is why it is essential to develop strategies to counteract its effects. On certain routes, up to 5% of the traffic is affected by driver fatigue. This project aims to create a simulation for detecting sleepiness, with a focus on determining whether a driver's eyes and mouth are open or closed. By monitoring eye movements, early signs of drowsiness can be detected, potentially preventing accidents. Yawning, a common indicator of fatigue, will also be monitored as a measure of driver alertness. Yawning is often a subconscious response to tiredness, with drivers frequently yawning in an attempt to stay awake.

Fatigue detection relies on a sequence of facial images where the duration of eye and mouth openings is observed. One popular method for detecting eye closure is PERCLOS, which measures the proportion of time during which the eyes remain closed. Image processing techniques in facial analysis are extensively studied, with applications such as facial recognition, identity verification, and security surveillance. This project specifically focuses on detecting the positioning of the eyes and mouth on a face image, applying image processing algorithms to locate and track these features. Once the eyes and mouth are

located, the system will assess whether they are open or closed, providing an early warning for fatigue and drowsiness.

Figure 1 Examples of Fatigue and Drowsiness Condition

B. Background Study

1) Facial Landmark Detection: Kazemi and Sullivan intro- duced a method for precisely estimating facial landmarks by training on a dataset of labeled facial images. This approach enables real-time facial feature detection, identifying key points on the face. Specifically, the method detects 68 facial landmarks, which include points for the jawline, eyebrows, eyes, nose, and mouth. The following figure illustrates the placement of these 68 key facial points.

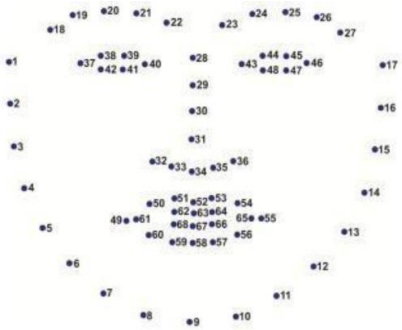

Figure 2 Illustration of 68 Landmarks on a Human Face

These facial landmarks are categorized as follows: jaw (points 1–17), right eyebrow (points 18–22), left eyebrow (points 23–27), nose (points 28–36), right eye (points 37–42),left eye (points 43–48), and mouth (points 49–68). For this study, we focus on 32 of these 68 feature points, specifically those corresponding to the eyes and mouth, to calculate the Eye Aspect Ratio (EAR) and Lip Distance (LIP) for detecting eye and mouth openings.

C. Yawning Detection

Yawning is typically characterized by the mouth opening widely. Similar to blink detection, facial landmarks are used to determine when the mouth is open. The mouth-opening measurement, referred to as the LIP distance, is calculated by finding the difference between the upper and lower mouth points [1], as expressed in the equation:

LIP distance = /LIP Top − LIP Low/If the LIP distance exceeds a predefined threshold for a

specific duration, it is identified as yawning. An alert is triggered if yawning persists beyond a set threshold, with temporary deviations due to talking or eating being ignored.

D. Blink Detection

Facial feature detection is applied to pinpoint various features on the face, including the eyes, nose, mouth, and eyebrows [2]. For blink detection, we focus on the eye region, which is represented by six specific coordinates, as shown in the following figure.

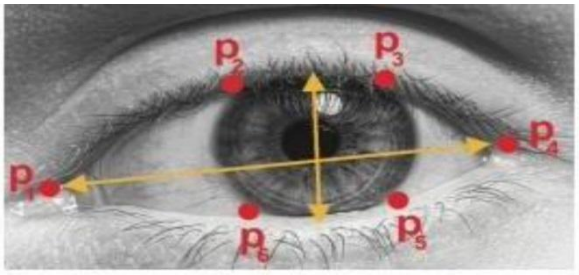

Figure 3 Illustration of Eye Calculation for Blink Detection

The Eye Aspect Ratio (EAR) is computed using the following formula:

EAR = (|p2-p6| + |p3-p5|) / (2|p1-p4|)

This equation, derived from the work of Soukupova and Cech [2], calculates the vertical and horizontal distances between key eye landmarks. The numerator measures the vertical distances between the eye landmarks, while the denominator computes the horizontal distance. This ratio remains relatively constant when the eye is open, but it drops significantly when the eye closes during blinking. Using this method, we can detect blinking events without the need for complex image processing techniques.

E. Behavioral Parameter-based Techniques

These techniques focus on assessing driver fatigue without relying on invasive methods. They analyze various behavioral parameters, such as the frequency of eye closure, blink rate, yawning, and head positioning. The eye-closure ratio is often used as a key parameter in these systems to gauge driver alertness.

F. Physiological Parameter-based Techniques

This approach measures driver fatigue by monitoring phys- iological indicators such as heart rate, body temperature, and respiratory rate. Although these methods provide highly accurate results, they require drivers to wear monitoring equip- ment, which can be uncomfortable and impractical. Ensuring that the driver consistently wears such devices may lead to inefficiencies, and hence, these approaches are not always suitable for real-time fatigue detection during driving.

Behavioral-based methods, relying on visual monitoring through cameras, are more practical. These techniques do not require physical attachments to the driver, making them

applicable in a wide range of vehicles without the need for modifications. This approach offers a flexible and non- intrusive solution for detecting driver fatigue.

G. Image Processing

Digital image processing refers to manipulating two- dimensional images using computational algorithms. It involves transforming images into a format that a computer can process, enabling various operations such as enhancement, analysis, and feature extraction. The primary advantage of digital image processing lies in its precision, flexibility, and repeatability.

Pixel: A pixel is the smallest unit of an image, representing a single color value. In an 8-bit grayscale image, pixel values range from 0 to 255. The intensity of light at a given point corresponds to the pixel's value.

Digital Image: A digital image consists of a grid of pixels, each representing variations in color or intensity.

Gray Level: The pixel value indicates the intensity of light at that point, also known as the gray level. To convert an image to grayscale, the pixel values for red, green, and blue channels are averaged, or weighted according to their perceived intensity as follows: Grayscale = (0.3 * R) + (0.59 * G) + (0.11 * B).

H. Significance

- To reduce the number of accidents caused by driver fatigue.
- To monitor facial expressions and assess the level of driver fatigue.
- To employ deep learning algorithms to improve the accuracy of fatigue detection.

I. Problem Statement

A significant number of road accidents are caused by driver drowsiness. Detecting drowsiness early can help reduce such accidents. This can be achieved by monitoring the driver through continuous video capture via a mobile device or camera. The goal is to develop a model that can accurately detect if a driver is experiencing fatigue. The system will take an image every second, analyze eye-blinking patterns, and compute the duration for which the eyes are closed using deep learning algorithms. If the blink rate increases and the eyes remain shut for a set period, the model will infer drowsiness and alert the driver.

J. Research Questions

1. What kind of data will be utilized for training the model?
2. How do behavioral cues contribute to identifying drowsy driving?
3. What methodologies exist for facial expression recognition?
4. Is this solution financially feasible?
5. How can deep learning techniques be employed to ad- dress this issue effectively?

K. Limitations

- The current system only triggers an alert; integrating IoT technology could enhance its functionality.
- Performance could improve by using a larger training dataset and incorporating advanced technological approaches.

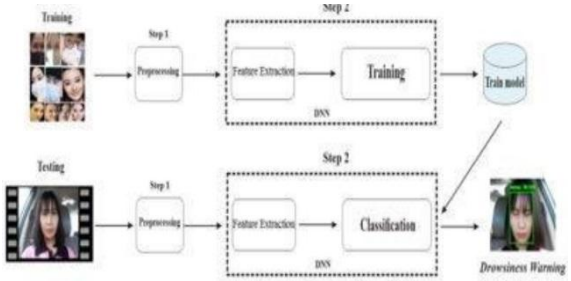

Figure 4 Architecture Diagram of the Proposed Network

RELATED WORK

Study 1

Title: A Robust Approach for Driver Drowsiness Detection Using Deep Learning

Summary: This research leverages advanced deep learning techniques, specifically the transfer learning strategy. Facial images are detected through the SSD-ResNet-10 model and processed through a custom network designed to detect drowsiness [3]. Two adaptive deep neural networks, based on MobileNet-V2 and ResNet-50V2 architectures, are employed. Enhancements in certain layers improve the accuracy of drowsiness detection. The process involves training the model on pre-processed datasets and evaluating its performance using a testing dataset.

Study 2

Title: Real-time Driver Drowsiness Monitoring Using Visual Cues and Machine Learning

Summary: An adaptive thresholding method has been developed to detect driver fatigue in real-time. This system performs efficiently using synthetic data [4]. Feature values are stored, and machine learning models are employed for classification. The study explores the effectiveness of Bayesian classifiers, FLDA, and SVM, with FLDA and SVM showing superior performance over Bayesian methods.

Study 3

Title: A Comprehensive Survey on Vehicle Drowsiness Detection Systems

Summary: This paper presents a non-invasive approach to drowsiness detection using eye-tracking and image processing. A robust eye detection algorithm accounts for variations in lighting conditions and driver posture. Six key measures, including eyelid closure percentage, blink frequency, and eye movement velocities, are calculated. The video-based detection system achieves 86% accuracy, though further improvements are

necessary [5]. The system is designed to protect both commercial and personal vehicles from theft, offering features like automatic car locking and accident prevention. By integrating drowsiness detection, the system aims to reduce accident risks.

Conclusion: Traditional drowsiness detection methods often rely on behavioral indicators or expensive sensors, which can distract the driver. Real-time, video-based monitoring using eye-tracking provides a more efficient solution. The technol- ogy can be enhanced using OpenCV and Haar Classifiers, offering a reliable method for fatigue detection. This system has the potential to be commercially applied in the automo- tive industry, especially when combined with data reduction techniques for improved accuracy.

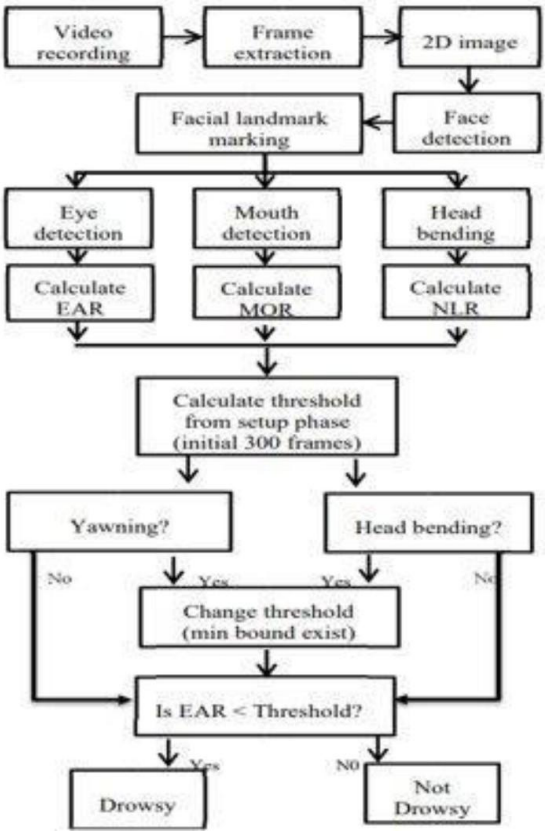

Figure 5 System Block Diagram for Drowsiness Detection

METHODOLOGY

Current Approaches

Existing drowsiness detection systems use physiological sensors to monitor parameters such as respiration, heart rate, and blood pressure. These devices often cause discomfort to the driver and may not be reliable if they malfunction or are not worn consistently. Furthermore, current systems struggle in low-light environments, reducing their accuracy in detecting the driver's face and eyes.

Proposed Solution

The proposed system utilizes a webcam to monitor the driver's face in real-time.

Using image processing and deep learning techniques, the system will detect signs of drowsiness from facial features.

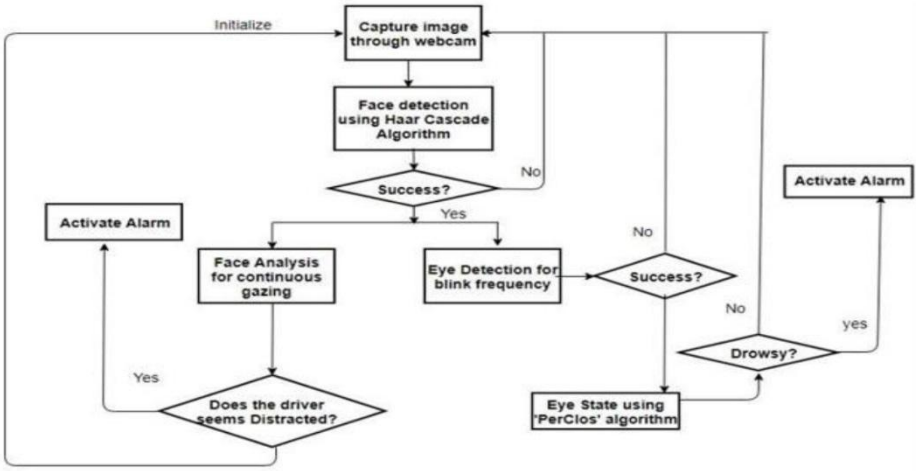

Figure 6 Proposed System Architecture

Key Components of the System

- Data Acquisition: Video frames are captured and processed.
- Face Detection: Faces in the frames are identified and isolated.
- Feature Extraction: Relevant facial features are extracted.
- Model Training: A facial recognition model is trained using the extracted features.

Execution Workflow

The process involves six distinct components:

1) Facial Recognition
2) Ocular Identification
3) Facial Monitoring
4) Eye Monitoring
5) Fatigue Detection
6) Attention Deviation Analysis

Facial Recognition: The initial phase identifies the driver's face within the video stream, utilizing Haar cascades specifically tailored for frontal face detection.

Ocular Identification: Following face recognition, the eyes are detected using the Haar Cascade Eye Detector to focus on identifying signs of fatigue.

Facial Monitoring: This step ensures that the driver's facial region is continuously observed and tracked.

Eye Monitoring: The condition of the eyes is closely monitored to determine indications of tiredness.

Fatigue Detection: If the eyes remain shut for a pro- longed duration, a warning signal is initiated.

Attention Deviation Analysis: The system examines the driver's focus, flagging instances of inattention or prolonged staring without blinking.

Software Development Life Cycle

The development process follows a well-established methodology, with the Waterfall model being the most suitable for this project. This model ensures a structured, step-by-step approach through the phases of planning, design, development, testing, and deployment. The five primary phases of the Waterfall model are illustrated below:

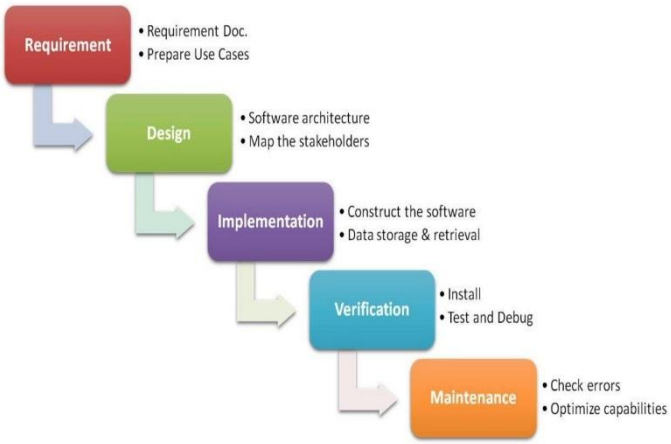

Figure 7 Waterfall Model Process

The key phases are:

- **Requirements:** All project requirements are documented before starting development.
- **Design:** The system design is prepared based on the requirements, focusing on user interface and functionality.
- **Development:** Actual coding and system building occur in this phase.
- **Verification:** Testing is performed to ensure that the system works as expected.
- **Maintenance:** After deployment, the system enters main- tenance, with any discovered issues being addressed.

EXPERIMENTAL EVALUATION AND OUTCOMES

System Configuration

Software Requirements

The following software setup is essential to execute this project:

Operating System: Compatible with Windows 10/8/7 (64-bit versions included), macOS, or Linux

Programming Language: Python version 3

IDE: JetBrains PyCharm Community Edition 2019.1.3 (64-bit)

Hardware Requirements

CPU: A quad-core processor with a minimum clock speed of 2.5 GHz per core (64-bit)

RAM: 4 GB or more of system memory
Storage: At least 20 GB of available hard drive space
Screen: Two monitors with a resolution of XGA (1024 x 768) or higher
Webcam: Detachable camera device
Input Device: Standard QWERTY keyboard

Testing Procedures

The main goal of testing is to identify defects. It involves checking all aspects of the system to find potential issues or weaknesses. Testing ensures that components, subassemblies, and the final product perform as expected. The process involves running the software to verify that it meets the defined requirements and a user expectation, ensuring the system's functionality is adequate. There are several types of tests, each focused on a specific area of the system's performance.

Variants of Testing

Unit Testing: Unit testing is focused on validating that each individual unit of code behaves as expected. It involves creating test cases that assess the internal logic of the software to ensure that program inputs lead to the expected outputs. Each decision point and internal flow of the application is tested. This type of testing is conducted after completing a unit and before integration into the overall system. Unit tests are invasive and structural, designed with an understanding of the software's inner workings. They validate whether the smallest functional unit of the system performs according to its specifications.

Functional Testing: Functional testing checks that the software operates according to the functional requirements outlined in business documents, system specifications, and user guides. This form of testing primarily focuses on ensuring that each function works as intended and that user inputs are processed correctly.

Core focus areas of Functional Testing include

- **Valid Inputs:** Recognizing and accepting valid input data.
- **Invalid Inputs:** Rejecting inputs that do not meet the requirements.
- **Functions:** Ensuring identified functions are correctly executed.
- **Outputs:** Verifying that outputs are generated as expected.
- **Procedures:** Checking that inter-system procedures work as intended.

The preparation and organization of functional tests focus on the critical system processes, exception handling, and key procedures. Systematic coverage of business processes, data fields, and predefined workflows is essential for the testing process. Functional testing involves identifying key business flows, verifying the system's data handling, and considering exceptional cases for further testing.

Integration Testing: Integration testing examines the inter- action between combined components of the system to ensure they work as one cohesive unit. This process typically involves event-driven tests that focus on verifying the output when components

interact. The aim is to uncover issues that may arise when previously tested units are integrated into the larger system.

Testing Plan

A test plan is a comprehensive document that outlines the scope, methodology, resources, and timeline of the testing phase. It identifies critical test items, lists the features to be examined, allocates testing responsibilities, defines the testing environment, and provides a structured approach to the entire testing process. This document also includes entry and exit criteria, outlining when testing should begin and when it is considered complete.

Testing Strategy Design

The testing strategy for this project involves determining the necessary efforts and resources for testing different com- ponents, both functional and hardware-related. The design of this strategy includes calculating the effort required to meet the objectives of the system.

Steps for Testing an Application

- Define the scope of testing.
- Determine the appropriate types of tests required.
- Identify potential risks and issues.
- Develop the logistics for the testing process.

Setting Test Objectives: The goal of testing is to ensure that the system is free from defects and is ready for end-user deployment. The test objectives are specified by identifying critical software features and the corresponding expected re- sults. Successful tests are those where the system behaves as outlined in the specifications.

Establishing Test Criteria: Test criteria are standards or guidelines used to determine the success of a test. Two main types of test criteria include:

- **Suspension Criteria:** If a certain threshold of test failures is reached, the testing process should be paused until the issues are resolved.
- **Exit Criteria:** These criteria signify when a particular test phase has been successfully completed.

Resource Planning: Resource planning involves outlining the necessary human, material, and equipment resources re- quired to successfully complete testing tasks.

Test Summary Report: During the testing phase, the system is tested through various test cases under multiple conditions to simulate different real-world scenarios. The following tables display some of the test cases and their outcomes.

Table 1 Face Detection Unit Testing

Test Case ID:	UTC-1
Test Name:	Face Detection
Item Tested:	Face
Input:	Video with a person's face
Expected Result:	Face detected in the video frame
Actual Result:	Matches expected output
Remarks:	Test successful

Table 2: Eye Detection Unit Testing

Test Case ID:	UTC-2
Test Name:	Eye Detection
Item Tested:	Eye
Input:	Driver's face
Expected Result:	Eye detection
Actual Result:	Matches expected output
Remarks:	Test successful

Table 3: Functional test for Drowsiness Detection based on Face and Eye Analysis

Test Case ID:	FTC-2
Test Description:	Driver Drowsiness Detection
Component Under Test:	Face and Eyes of the Driver
Test Input:	Image of the driver's face and eyes
Expected Outcome:	Detection of drowsiness
Actual Outcome:	Matches the expected result
Comments:	Test passed successfully

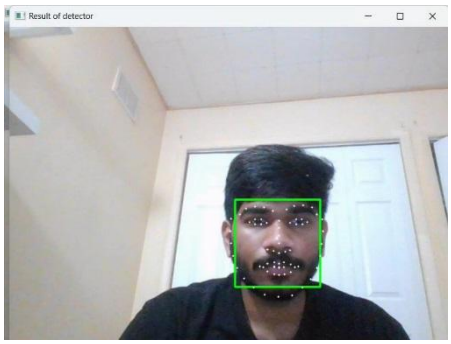

Figure 8 Facial Detection Process

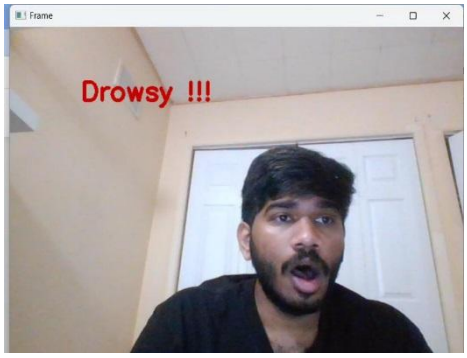

Figure 9 Drowsiness Detection

Figure 10 Sleep Detection

CONCLUSION

This work presents an innovative automated system designed to detect signs of driver fatigue. The system continuously processes video data to identify drowsiness in real-time. The core of this solution involves deep learning techniques, particularly for facial and ocular recognition. The approach uses a specialized library to detect facial landmarks and calculates the blink frequency, applying a predefined algorithm to assess fatigue levels. When a driver's blinking rate falls below a threshold, indicating potential sleepiness, an alert is triggered to notify the driver.

APPENDIX A

Example Code for Driver Drowsiness Detection

```
import cv2
import numpy as np
import dlib
from imutils import face_utils
# Initialize the face detection module
face_detector = dlib.get_frontal_face_detector()
# Load the model for predicting facial landmarks
landmark_model = dlib.shape_predictor("path_to_model\\shape_predictor
_68_face_landmarks.dat")
# Initialize counters and status variables
```

```
asleep_count = 0
drowsiness_count = 0
alert_count = 0 current_status = "" status_color = (0, 0, 0)
# Function to calculate the Euclidean distance
def calculate_distance(pointA, pointB): distance = np.linalg.norm(pointA - pointB) return
distance
# Function to determine if an eye is blinking
def detect_blink(p1, p2, p3, p4, p5, p6): vertical = calculate_distance(p2, p4) +
calculate_distance(p3, p5) horizontal = calculate_distance(p1, p6) ratio = vertical / (2.0 *
horizontal) if ratio > 0.25:
return 2
elif 0.21 <= ratio <= 0.25:
return 1
else:
return 0
# Begin video frame analysis
while True:
_, video_frame = cap.read()
# Convert the frame to grayscale
grayscale_frame = cv2.cvtColor(video_frame, cv2.COLOR_BGR2GRAY)
detected_faces = face_detector(grayscale_frame)
for single_face in detected_faces: x_start = single_face.left()
y_start = single_face.top()
face_landmarks[44],
face_landmarks[47],
face_landmarks[46],
face_landmarks[45])
if (left_eye_status == 0 or
right_eye_status == 0):
asleep_count += 1
drowsiness_count = 0
alert_count = 0
if (asleep_count > 6): current_status = "SLEEPING!!!" status_color = (255, 0, 0)
elif (left_eye_status == 1 or
right_eye_status == 1):
asleep_count = 0
drowsiness_count += 1
alert_count = 0
if (drowsiness_count > 6): current_status = "Drowsy!!!" status_color = (0, 0, 200)
```

```
else:
asleep_count = 0
drowsiness_count = 0
alert_count += 1
if (alert_count > 6): current_status = "Active :)" status_color = (45, 50, 100)
cv2.putText(video_frame, current_status, (100, 100), cv2.FONT_HERSHEY_SIMPLEX,
1.2, status_color, 3)
for idx in range(0, 68):
(x_coord, y_coord) = face_landmarks[idx] cv2.circle(face_snapshot, (x_coord,
y_coord), 1, (255, 255, 255), -1)
cv2.imshow("Video Frame", video_frame) cv2.imshow("Detection Result",
face_snapshot)
exit_key = cv2.waitKey(1)
if exit_key == 27:
break
```

Listing 1. Python code for Drowsiness Detection

```
x_end = single_face.right() y_end = single_face.bottom()
face_snapshot = video_frame.copy() cv2.rectangle(face_snapshot, (x_start,
y_start), (x_end, y_end), (0, 255, 0),
2)
face_landmarks = landmark_model(grayscale_frame, single_face)
face_landmarks = face_utils.shape_to_np(face_landmarks)
left_eye_status = detect_blink(face_landmarks[36], face_landmarks[37],
face_landmarks[38], face_landmarks[41], face_landmarks[40], face_landmarks[39])
right_eye_status =
detect_blink(face_landmarks[42], face_landmarks[43],
```

REFERENCES

[1] Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., & Pantic, M. (2013). 300 faces in-the-wild challenge: The first facial landmark localization challenge. *IEEE International Conference on Computer Vision Workshops*.

[2] Soukupova, T., &Cech J. (2016). Real-time eye blink detection using facial landmarks. *Proceedings of the 21st Computer Vision Winter Workshop*.

[3] Phan, A. C., Nguyen, N.-H.-Q., Trieu, T.-N., & Phan, T.-C. (2021).An Efficient Approach for Detecting Driver Drowsiness Based on Deep Learning. *Applied Sciences*.

[4] Kumar, A., & Patra, R. (2018). Driver drowsiness monitoring system using visual behaviour and machine learning. *IEEE Symposium on Computer Applications & Industrial Electronics*.

[5] Walke, K. G., Shete, H., Asknani, D., Gadekar, R., & Mulani, A. (2021). Survey on driver drowsiness detection systems. *IJIRSET*.