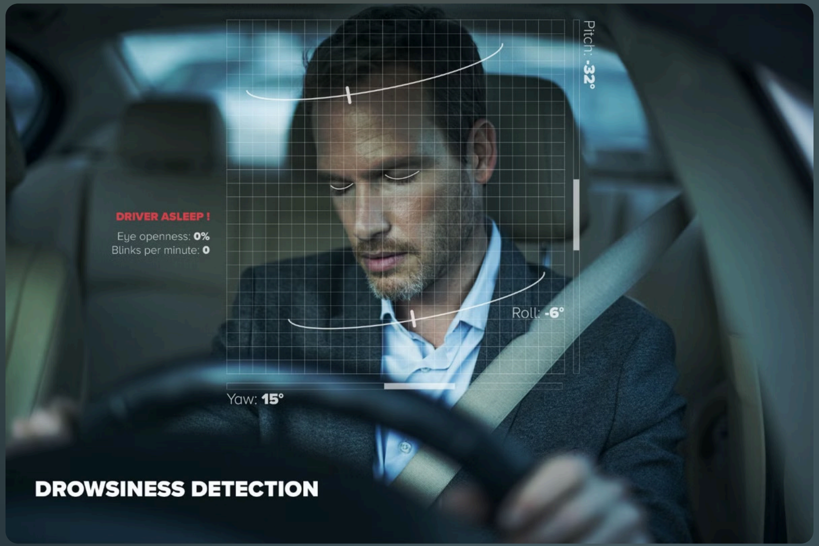


Driver Drowsiness Detection System

NAME	ROLL NO
SRIYA PANADA	2106265
ANIKET V	2106295
UJJWAL SINGH	2106274
HIMANSHU DASH	2106117
PRAKASH KUMAR	2106132
ALTAMASH DANYAL	2106032

Introduction

Objective

Reduce accidents by proactively detecting driver fatigue before it leads to dangerous situations.

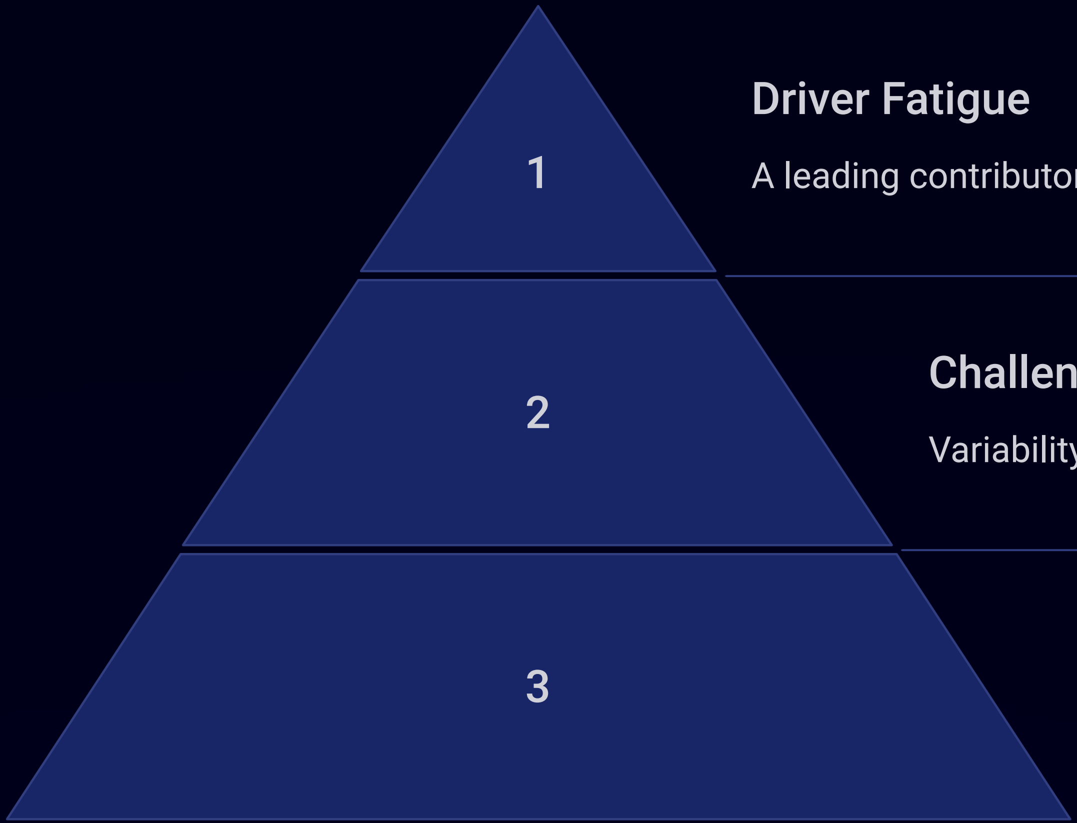
Significance

- Road safety enhancement.
- Integration of cutting-edge deep learning techniques.

Scope

- Applications for personal, commercial, and public transportation.
- Real-time monitoring and alert systems.

Problem Statement



Driver Fatigue

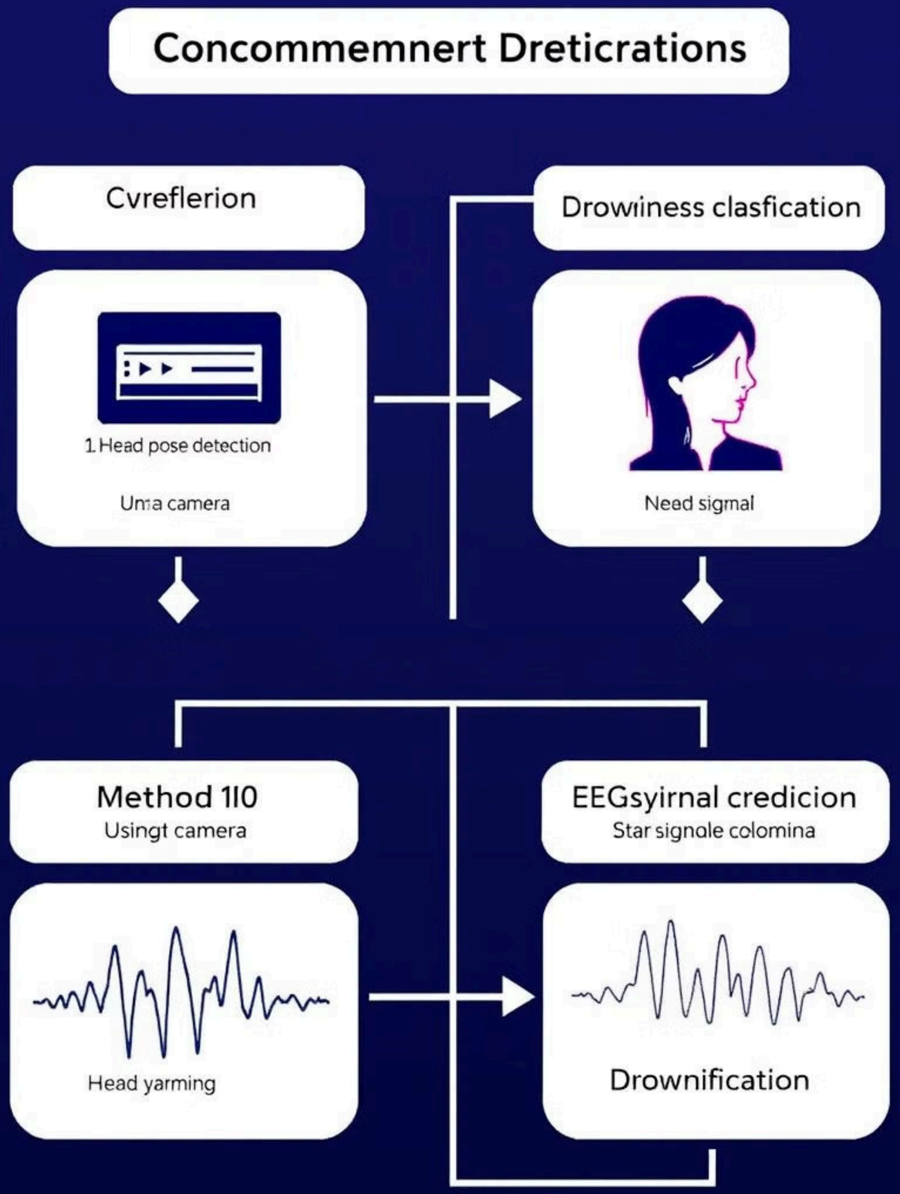
A leading contributor to road accidents globally.

Challenges

Variability in lighting, obstructions, and diverse physiological profiles.

Proposed Solution

Real-time, reliable drowsiness detection and alert system, scalable for diverse use cases.



Proposed Methods

YOLOv5-Based Detection

- Object detection for facial features.
- High-speed processing (~30 FPS).

CNN and Flask Integration

Real-time analysis with TensorFlow-based CNN, providing a user-friendly web interface via Flask.

Yolo Implementation

1 Model Testing

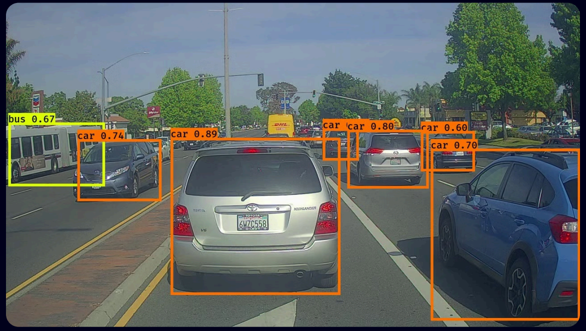
```python
import torch
from matplotlib import pyplot as plt #
import numpy as np
import cv2
```

```python
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
img = "./ImagePath"
```

```python
results = model(img)
```

```python
plt.imshow(np.squeeze(results.render()))
plt.show()
```

Testing a YOLO model evaluates its detection accuracy and speed on unseen data.



2 Data Preparation

Prepare YOLO data by labeling images and formatting annotations.

```python
import uuid   # Unique identifier
import os
import time
```

```python
IMAGES_PATH = os.path.join('data', 'images') #/data/images
labels = ['awake', 'drowsy']
number_imgs = 20
```

```python
cap = cv2.VideoCapture(0)
for label in labels:
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for img_num in range(number_imgs):
        print(' image number {}'.format img_num))
        ret, frame = cap.read()
        imgname = os.path.join(IMAGES_PATH, label+'.'+
                               str(uuid.uuid1())+'.jpg')
        cv2.imwrite(imgname, frame)
        cv2.imshow('Image Collection', frame)
        time.sleep(2)f
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

3 Model Training

feeding labeled images into a neural network to optimize weights for detecting and classifying objects within bounding boxes in real-time.

```
python train.py --img 320 --batch 16 --epochs 500
    --data dataset.yml --weights yolov5s.pt
```

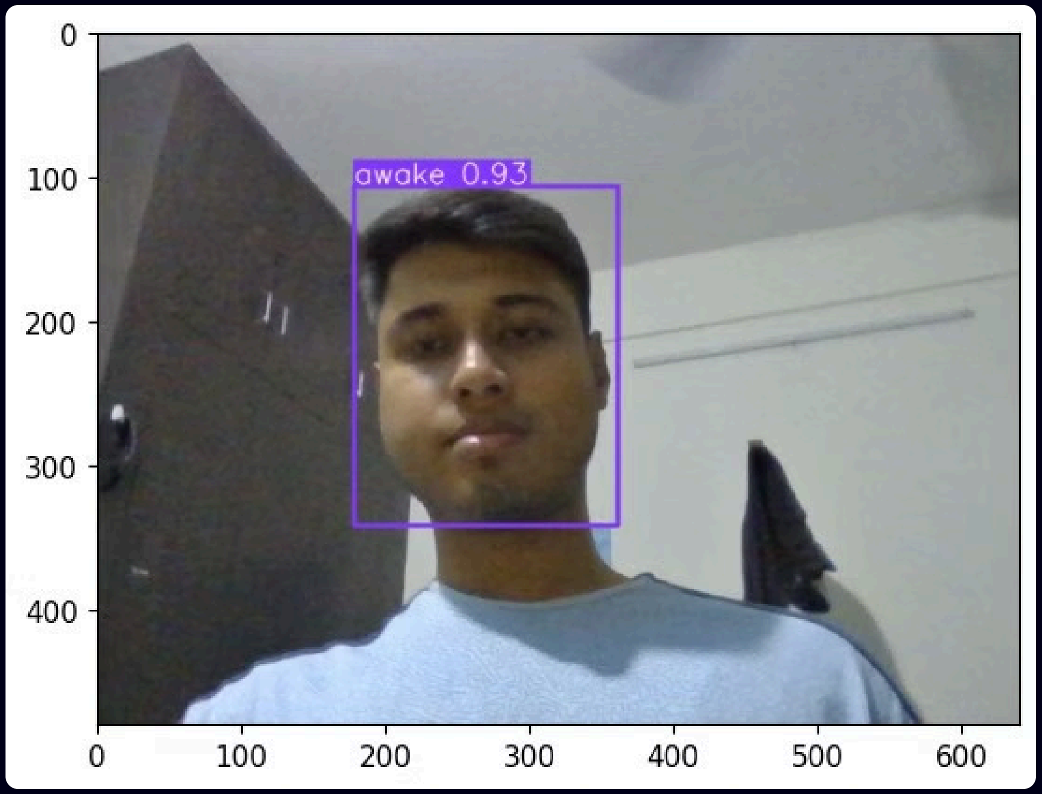
```
model = torch.hub.load('ultralytics/yolov5', 'custom',
path='yolov5/runs/train/exp15/weights/last.pt', force_reload=True)

img = os.path.join('data', 'images', 'awake.4b2e29ef.jpg')
results = model(img)
results.print()

%matplotlib inline
plt.imshow(np.squeeze(results.render()))
plt.show()
```

Custom Model Testing

A custom YOLO model was tested on an input image to detect specific features or objects. The detection results, including bounding boxes and class labels, were visualized directly in a Jupyter Notebook using Matplotlib's pyplot.



Output

The YOLO model successfully detects the presence of a person in the frame and classifies their state as "awake". The bounding box highlights the detected region, demonstrating the model's effectiveness in identifying and classifying human states.

Result and Problems with Yolo

Our YOLO-based model is successfully detecting whether a person is drowning or awake, providing critical insights for safety monitoring. However, the current implementation faces limitations in tracking finer details, such as the eyes of the person. This highlights a need for further refinement or integration with specialized models to enhance its capability for detecting smaller, detailed features critical for advanced analysis.

Implementation

Data Preparation

Utilizing datasets like MRL Eye for model training and evaluation.

Model Training

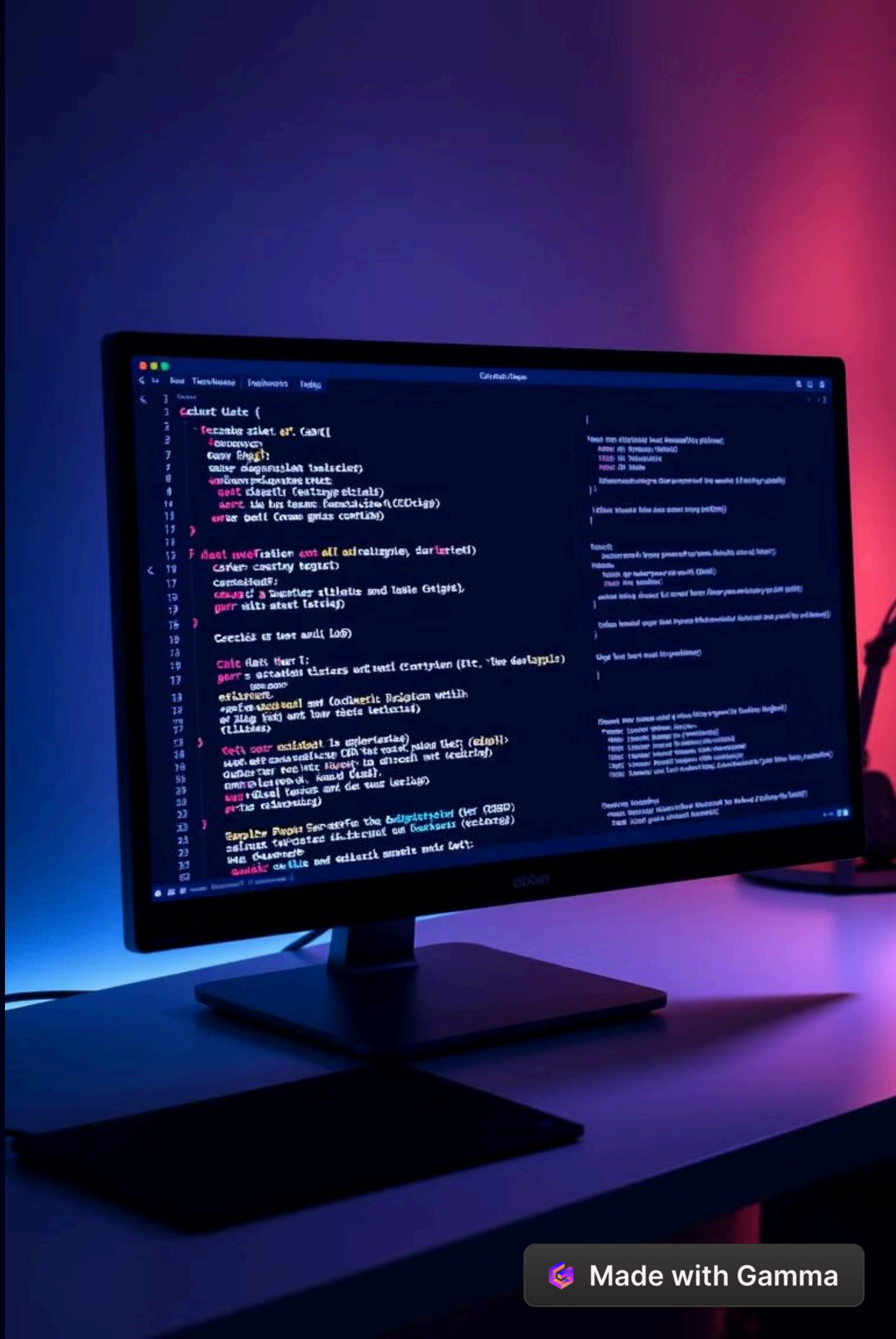
Fine-tuning YOLOv5 and CNN models for object detection and drowsiness classification.

Backend Integration

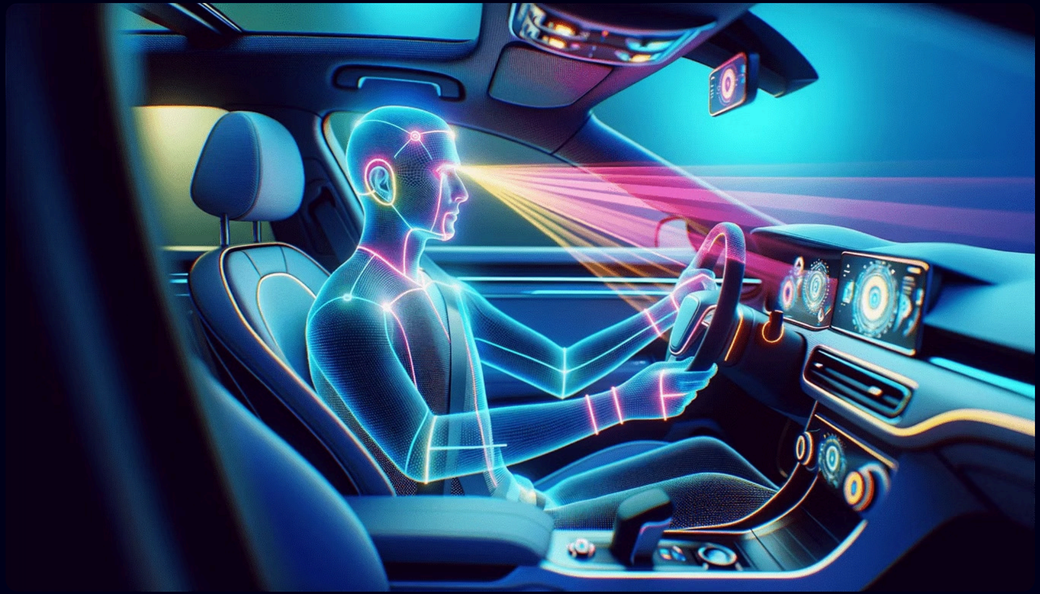
Leveraging Flask for API development and real-time model interaction.

User Interface

Developing web-based (HTML, CSS, JS) and desktop-based (Tkinter) interfaces for user interaction and analysis.



CNN and Flask Integration for Drowsiness Detection



Overview

- **Objective:** Real-time drowsiness detection using CNN.
- **Components:**
  - TensorFlow CNN for eye-state classification.
  - Flask backend for video processing and analytics.

Real-Time Drowsiness Detection Using CNN

The drowsiness detection system employs a Convolutional Neural Network (CNN) to analyze video frames, specifically focusing on the driver's eyes. This system uses real-time video processing to detect signs of drowsiness, ensuring a proactive approach to driver safety.

Key Features & Technology Stack

Key Features

- Web-based interface for real-time updates and analytics.
- High-accuracy CNN for drowsiness detection using OpenCV.
- Configurable alerts via web interface.

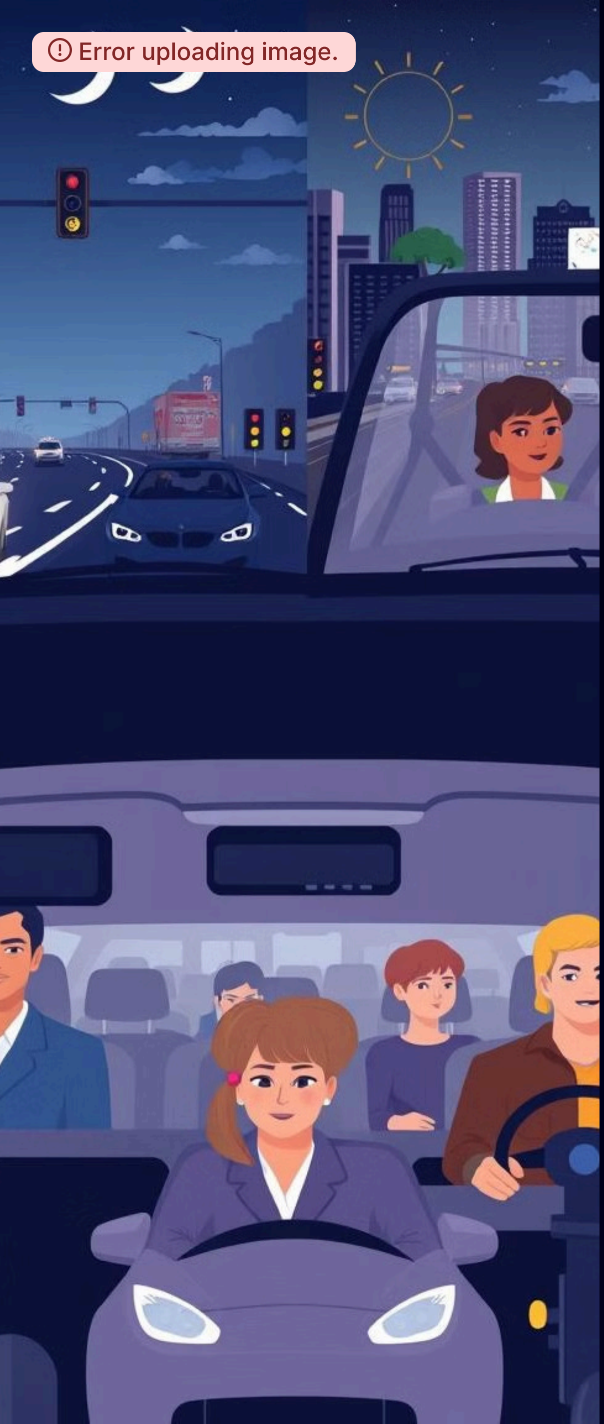
This system enables proactive driver safety by identifying drowsiness and providing timely alerts.

Technology Stack

- HTML, CSS, and JavaScript for the frontend user interface.
- Flask and TensorFlow for the backend logic and model processing.
- OpenCV for real-time frame analysis and image processing.
- Pygame for audio notifications.

Use Cases

- **Personal Vehicle Monitoring:** For long drives, the system can help drivers stay alert and avoid fatigue-related accidents.
- **Fleet Management:** Companies can implement the system in their fleet vehicles to ensure driver safety and reduce risks associated with drowsiness.
- **Enhanced Road Safety:** By detecting and alerting drivers to drowsiness, the system contributes to safer roads for all drivers.



Future Enhancements

Head Pose Estimation

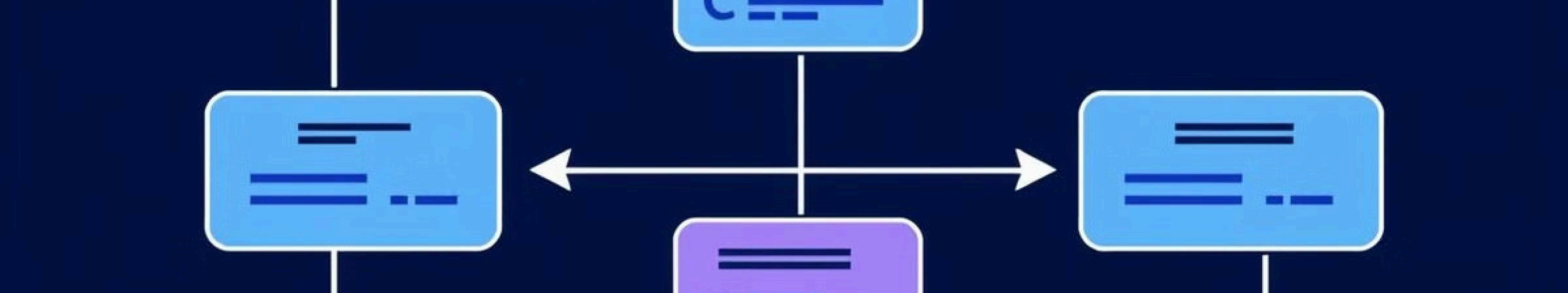
Incorporate head pose estimation to detect drowsiness signs beyond eye closure, improving accuracy.

Mobile Interface

Develop a mobile-friendly interface for broader accessibility and convenience.

Adaptive Lighting

Enhance the system to adapt to varying lighting conditions, ensuring consistent performance.



System Design

1. **Input**

   Real-time video feed from various sources, like car cameras or webcams.

2. **Processing**

   Preprocessing, feature extraction, and drowsiness detection using the trained model.

3. **Output**

   Visual overlays on the video, audio alerts, and an analytics dashboard for monitoring.

Design Constraints:

- Hardware requirements (e.g., webcams).
- Challenges with lighting and obstructions.

System Workflow

1

Data Preparation

The dataset consists of labeled images of open and closed eyes. This data is crucial for training the CNN.

2

Model Training

A CNN is trained on the preprocessed dataset to learn the patterns of open and closed eyes. The model achieves high accuracy in classifying eye states.

3

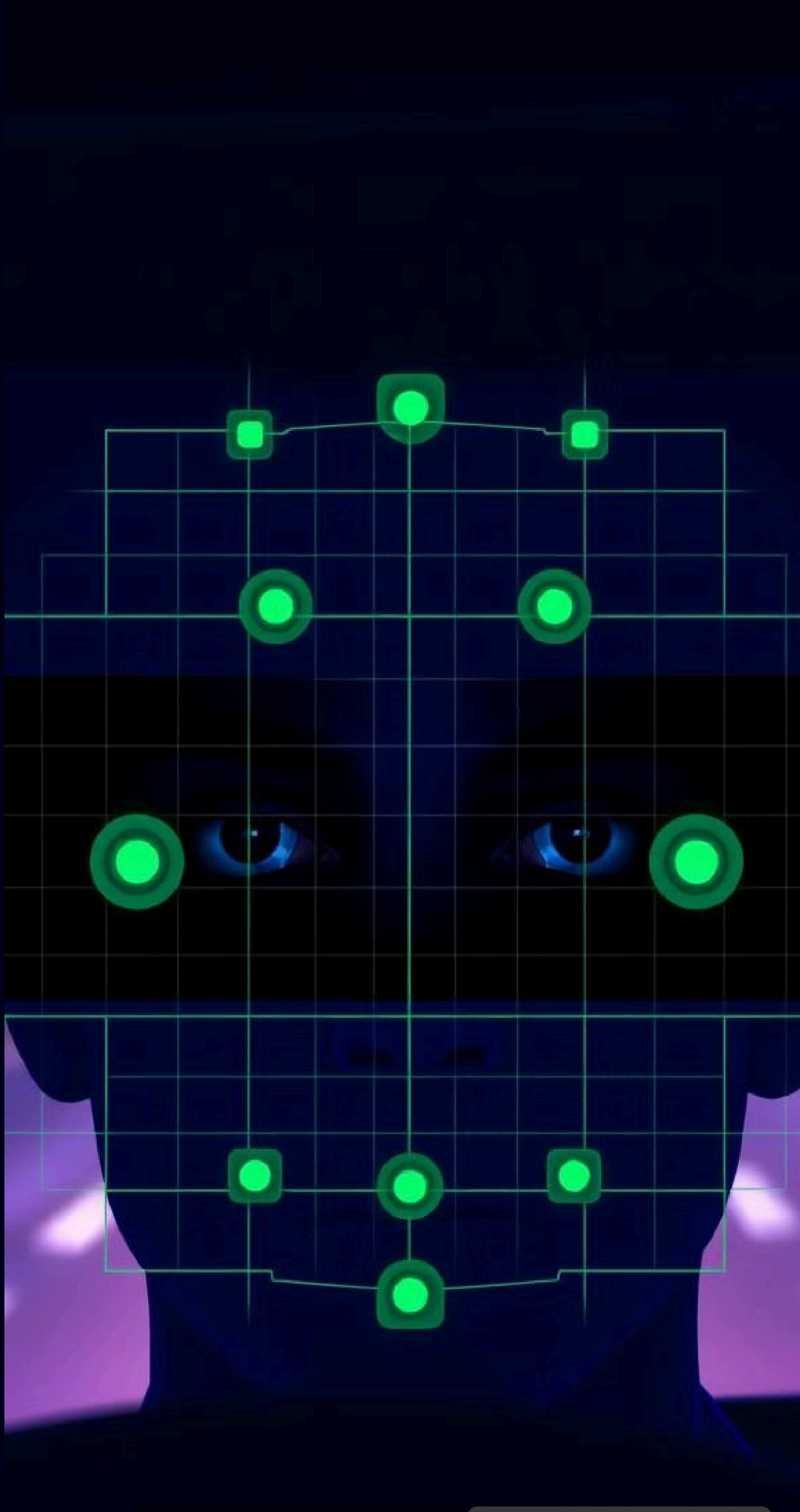
Real-Time Detection

The Flask backend processes video frames captured from the driver's camera in real-time.

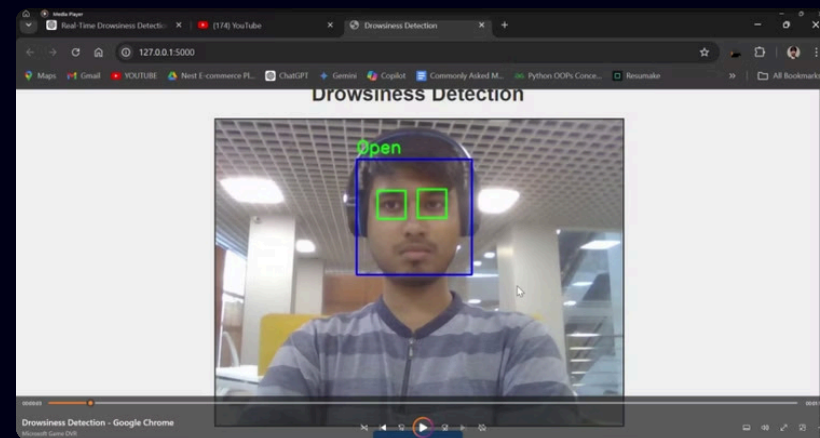
4

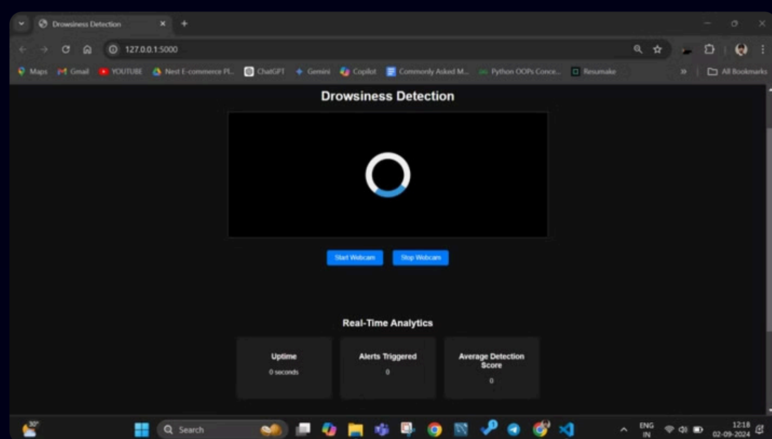
Analytics Dashboard

The system tracks various metrics for analysis and performance monitoring.



Results and Discussions





Results Overview:

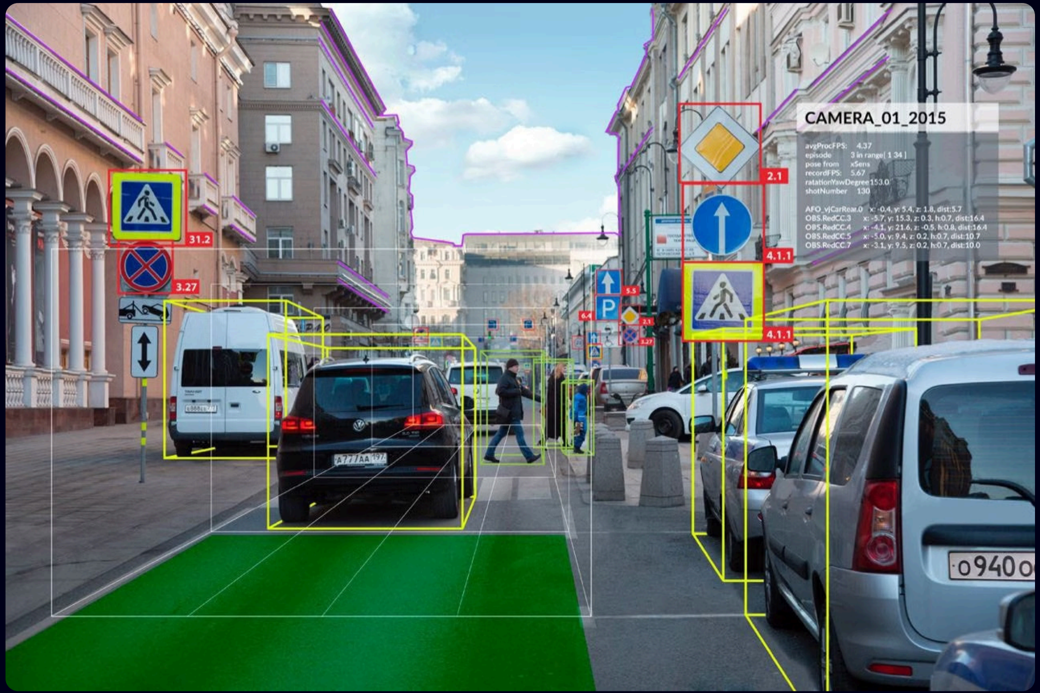
- High accuracy and real-time detection.
- Scalability across platforms.

Observations:

- Strengths: Robust detection, low latency.
- Limitations: Challenges with obstructions, lack of head pose detection.

Future Scope

- **IoT Integration**: Enable seamless deployment in smart vehicles.
- **Mobile Applications**: Develop apps for Android and iOS platforms.
- **Head Pose Estimation**: Add algorithms to track head tilts or nodding.
- **Blink Rate Monitoring**: Include abnormal blinking patterns for early fatigue detection.
- **Advanced Algorithms**: Explore 3D-CNNs for enhanced analysis.
- **Environmental Adaptation**: Implement sensitivity adjustments based on lighting and environmental conditions.
- **Yawn Detection**: Extend detection capabilities with mouth movement analysis.
- **Integration with Vehicle Systems**: Include features like automatic braking and alerting the passenger system.



Conclusion

Summary:

- Effective real-time detection.
- Scalability for personal and commercial use.