19EEE362 Deep Learning for Visual Computing

Project Report

Driver Drowsiness Detection and Alert System

Team members

1.	**CB.EN.U4CCE21001**	**Abburu Sahasranshu**
2.	**CB.EN.U4CCE21040**	**Nithin G D**
3.	**CB.EN.U4CCE21062**	**Santosh Kumar**
4.	**CB.EN.U4CCE21077**	**Krishna Moorthy**

Introduction

Road accidents due to driver drowsiness are a significant concern globally, leading to substantial loss of life and property. According to the National Highway Traffic Safety Administration (NHTSA), drowsy driving is responsible for thousands of fatalities each year. Despite advancements in vehicle safety technology, driver fatigue remains a critical challenge. There is an urgent need for an effective solution to detect driver drowsiness in real-time and provide timely alerts to prevent potential accidents.

Problem Statement

To develop a robust driver drowsiness detection system using deep learning techniques integrated with OpenCV. The system aims to continuously monitor the driver's facial expressions to detect signs of drowsiness and issue an alert to wake the driver and suggestion to engage auto pilot mode, thereby enhancing road safety.

Methodology

The system involves the following components:

1. **Video Capture**: Continuous capture of video frames using an in-car camera focused on the driver's face.
2. **Facial Landmark Detection**: Used HaarCascadeClassifier to detect key facial landmarks such as eyes, mouth, and head position.
3. **Feature Extraction**: Extract features associated with drowsiness, including eye closure duration, blinking frequency, and yawning.
4. **Deep Learning Model**: A pre-trained CNN model fine-tuned to classify drowsiness based on the extracted features.
5. **Alert Mechanism**: Trigger an audible alarm if the model detects drowsiness for a specified period.

Dataset

The dataset used to train the model has images belonging to 4 different classes – yawn, no_yawn, open and closed. Each class has about 725 different images that are used as training/validation set to build the model.

Code

Real-time drowsiness detection code

```
import cv2
import numpy as np

from keras.models import load_model
from keras.preprocessing.image import img_to_array
from playsound import playsound
from threading import Thread



def start_alarm(sound):
    """Play the alarm sound"""
    playsound('data/alarm.mp3')



classes = ['Closed', 'Open']
face_cascade =
cv2.CascadeClassifier("data/haarcascade_frontalface_default.xml")
left_eye_cascade =
cv2.CascadeClassifier("data/haarcascade_lefteye_2splits.xml")
right_eye_cascade =
cv2.CascadeClassifier("data/haarcascade_righteye_2splits.xml")
cap = cv2.VideoCapture(0)
model = load_model("drowiness_new7.h5")
```

```
count = 0
alarm_on = False
alarm_sound = "data/alarm.mp3"
status1 = ''
status2 = ''

while True:
    _, frame = cap.read()
    height = frame.shape[0]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 1)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        left_eye = left_eye_cascade.detectMultiScale(roi_gray)
        right_eye = right_eye_cascade.detectMultiScale(roi_gray)
        for (x1, y1, w1, h1) in left_eye:
            cv2.rectangle(roi_color, (x1, y1), (x1 + w1, y1 + h1), (0, 255, 0), 1)
            eye1 = roi_color[y1:y1+h1, x1:x1+w1]
            eye1 = cv2.resize(eye1, (145, 145))
            eye1 = eye1.astype('float') / 255.0
            eye1 = img_to_array(eye1)
            eye1 = np.expand_dims(eye1, axis=0)
            pred1 = model.predict(eye1)
            status1=np.argmax(pred1)
            #print(status1)
```

```
            #status1 = classes[pred1.argmax(axis=-1)[0]]
            break

        for (x2, y2, w2, h2) in right_eye:
            cv2.rectangle(roi_color, (x2, y2), (x2 + w2, y2 + h2), (0, 255, 0), 1)
            eye2 = roi_color[y2:y2 + h2, x2:x2 + w2]
            eye2 = cv2.resize(eye2, (145, 145))
            eye2 = eye2.astype('float') / 255.0
            eye2 = img_to_array(eye2)
            eye2 = np.expand_dims(eye2, axis=0)
            pred2 = model.predict(eye2)
            status2=np.argmax(pred2)
            #print(status2)
            #status2 = classes[pred2.argmax(axis=-1)[0]]
            break

        # If the eyes are closed, start counting
        if status1 == 2 and status2 == 2:
        #if pred1 == 2 and pred2 == 2:
            count += 1
            cv2.putText(frame, "Eyes Closed, Frame count: " + str(count), (10, 30),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 1)
            # if eyes are closed for 10 consecutive frames, start the alarm
            if count >= 10:
                cv2.putText(frame, "Drowsiness Alert!!!", (100, height-20),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)
                if not alarm_on:
                    alarm_on = True
```

```
                # play the alarm sound in a new thread
                t = Thread(target=start_alarm, args=(alarm_sound,))
                t.daemon = True
                t.start()
        else:
            cv2.putText(frame, "Eyes Open", (10, 30),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 1)
            count = 0
            alarm_on = False

    cv2.imshow("Drowsiness Detector", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Neural network training code

1) Yawn detection

```python
def face_for_yawn(direc=r"archive\train", face_cas_path=r"archive(1)\haarcascade_frontalface_default.xml"):
    yaw_no = []
    IMG_SIZE = 145
    categories = ["yawn", "no_yawn"]
    for category in categories:
        path_link = os.path.join(direc, category)
        class_num1 = categories.index(category)
        print(class_num1)
        for image in os.listdir(path_link):
            image_array = cv2.imread(os.path.join(path_link, image), cv2.IMREAD_COLOR)
            face_cascade = cv2.CascadeClassifier(face_cas_path)
            faces = face_cascade.detectMultiScale(image_array, 1.3, 5)
            for (x, y, w, h) in faces:
                img = cv2.rectangle(image_array, (x, y), (x+w, y+h), (0, 255, 0), 2)
                roi_color = img[y:y+h, x:x+w]
                resized_array = cv2.resize(roi_color, (IMG_SIZE, IMG_SIZE))
                yaw_no.append([resized_array, class_num1])
    return yaw_no
```

2) Eye closed/open detection

```python
def get_data(dir_path=r"archive\train", face_cas=r"archive(1)\haarcascade_frontalface_default.xml", eye_cas=r"archive(1)\haarcascade.xml"):
    labels = ['Closed', 'Open']
    IMG_SIZE = 145
    data = []
    for label in labels:
        path = os.path.join(dir_path, label)
        class_num = labels.index(label)
        class_num +=2
        print(class_num)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_COLOR)
                resized_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                data.append([resized_array, class_num])
            except Exception as e:
                print(e)
    return data
```

3) Model

```python
model = Sequential()

model.add(Conv2D(256, (3, 3), activation="relu", input_shape=(145,145,3)))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="softmax"))

model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer="adam")
```

4) Training the model

```
history = model.fit(train_generator, epochs=50, validation_data=test_generator, shuffle=True, validation_steps=len(test_generator))
43/43 [==============================] - 247s 6s/step - loss: 0.2934 - accuracy: 0.8775 - val_loss: 0.2595 - val_accuracy: 0.8858
Epoch 7/50
43/43 [==============================] - 6473s 154s/step - loss: 0.2589 - accuracy: 0.9035 - val_loss: 0.2098 - val_accuracy: 0.9048
Epoch 8/50
43/43 [==============================] - 207s 5s/step - loss: 0.2530 - accuracy: 0.8990 - val_loss: 0.2265 - val_accuracy: 0.8979
Epoch 9/50
43/43 [==============================] - 388s 9s/step - loss: 0.2421 - accuracy: 0.9072 - val_loss: 0.2006 - val_accuracy: 0.9048
Epoch 10/50
43/43 [==============================] - 395s 9s/step - loss: 0.2165 - accuracy: 0.9079 - val_loss: 0.1884 - val_accuracy: 0.9170
Epoch 11/50
43/43 [==============================] - 392s 9s/step - loss: 0.1997 - accuracy: 0.9161 - val_loss: 0.1857 - val_accuracy: 0.9187
Epoch 12/50
43/43 [==============================] - 387s 9s/step - loss: 0.1970 - accuracy: 0.9109 - val_loss: 0.1609 - val_accuracy: 0.9239
Epoch 13/50
43/43 [==============================] - 385s 9s/step - loss: 0.2037 - accuracy: 0.9169 - val_loss: 0.2036 - val_accuracy: 0.9100
Epoch 14/50
43/43 [==============================] - 397s 9s/step - loss: 0.1765 - accuracy: 0.9339 - val_loss: 0.1495 - val_accuracy: 0.9343
Epoch 15/50
43/43 [==============================] - 389s 9s/step - loss: 0.1801 - accuracy: 0.9265 - val_loss: 0.1812 - val_accuracy: 0.9170
```

5) Performance metrics

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(accuracy))

plt.plot(epochs, accuracy, "b", label="trainning accuracy")
plt.plot(epochs, val_accuracy, "r", label="validation accuracy")
plt.legend()
plt.show()

plt.plot(epochs, loss, "b", label="trainning loss")
plt.plot(epochs, val_loss, "r", label="validation loss")
plt.legend()
plt.show()
```

Model architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 143, 143, 256)	7,168
max_pooling2d (MaxPooling2D)	(None, 71, 71, 256)	0
conv2d_1 (Conv2D)	(None, 69, 69, 128)	295,040
max_pooling2d_1 (MaxPooling2D)	(None, 34, 34, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	73,792
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 14, 14, 32)	18,464
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 32)	0
flatten (Flatten)	(None, 1568)	0
dropout (Dropout)	(None, 1568)	0
dense (Dense)	(None, 64)	100,416
dense_1 (Dense)	(None, 4)	260

1) Input Layer:

- **Conv2D Layer**: This layer applies 256 filters, each of size 3x3, to the input image. The input shape=(145,145,3) specifies that the input images are 145x145 pixels with 3 color channels (RGB).
- **Activation**: relu (Rectified Linear Unit) introduces non-linearity to the model, allowing it to learn more complex features.
- This layer extracts low-level features such as edges and textures from the input image.

2) Pooling Layer:

- **MaxPooling2D Layer**: This layer performs down-sampling by taking the maximum value over a 2x2 window. It reduces the spatial dimensions of the feature maps, which decreases the computational cost and helps in achieving translation invariance.

3) Additional Convolutional and Pooling Layers:

```
model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))

model.add(Conv2D(32, (3, 3), activation="relu"))
model.add(MaxPooling2D(2, 2))
```

- These additional convolutional and pooling layers further extract more complex and abstract features as the image data flows deeper into the network.
- As the number of filters decreases (256, 128, 64, 32), the model transitions from extracting detailed features to more abstract features.

4) Flatten Layer:

- The **Flatten Layer** converts the 2D feature maps from the last convolutional layer into a 1D vector, preparing the data for the fully connected layers.

5) Dropout Layer:

- The **Dropout Layer** helps in preventing overfitting by randomly setting 50% of the input units to 0 at each update during training time. This forces the model to learn more robust features that are not reliant on specific neurons.

6) Fully Connected Layers:

- **Dense (Fully Connected) Layer**: The first dense layer with 64 neurons and relu activation helps in learning complex representations from the flattened features.
- The final dense layer with 4 neurons and softmax activation outputs a probability distribution over 4 classes. This layer is particularly used when dealing with multi-class classification problems.

Output-

- The deep learning model classifies the driver's state as either "Alert" or "Drowsy" based on the input from the video feed and the autopilot mode is enabled accordingly.
- The classification is performed with high accuracy, minimizing false positives (incorrectly identifying alert drivers as drowsy) and false negatives (failing to detect drowsiness).

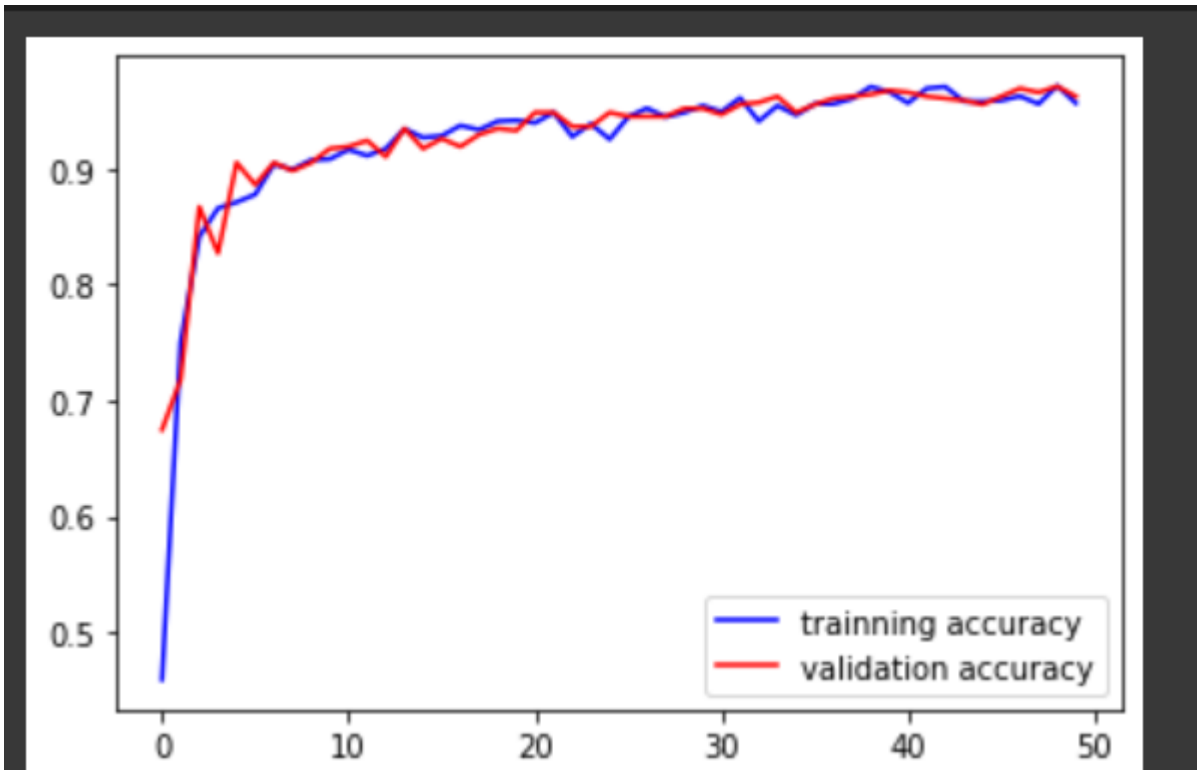


Figure 1Accuracy comparison

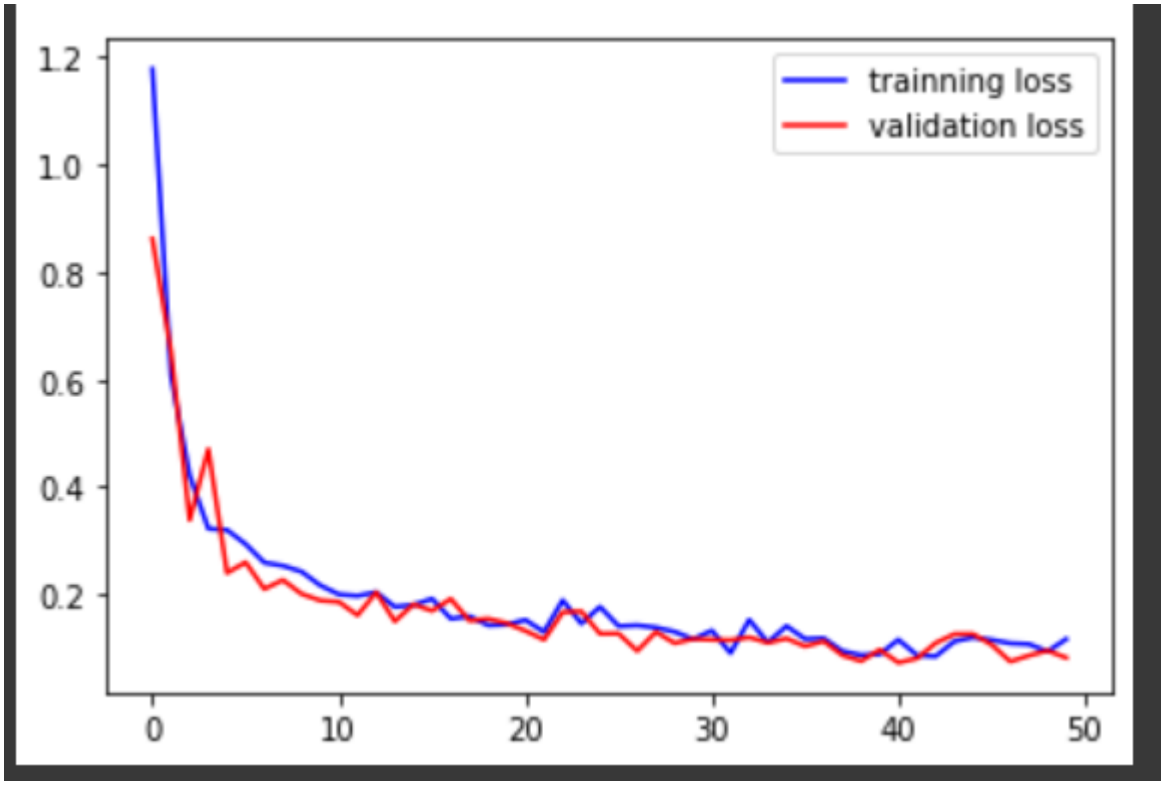


Figure 2Loss curve

	precision	recall	f1-score	support
yawn	0.63	0.95	0.76	63
no_yawn	0.93	0.54	0.68	74
Closed	0.94	0.98	0.96	215
Open	0.98	0.94	0.96	226
accuracy			0.90	578
macro avg	0.87	0.85	0.84	578
weighted avg	0.92	0.90	0.90	578

Figure 3Classification report

0-yawn, 1-no_yawn, 2-Closed, 3-Open

```
[ ]  # prepare("../input/drowsiness-dataset/train/no_yawn/1068.jpg")
     prediction = model.predict([prepare(r"archive\train\no_yawn\1067.jpg")])
     np.argmax(prediction)
```

```
1
```

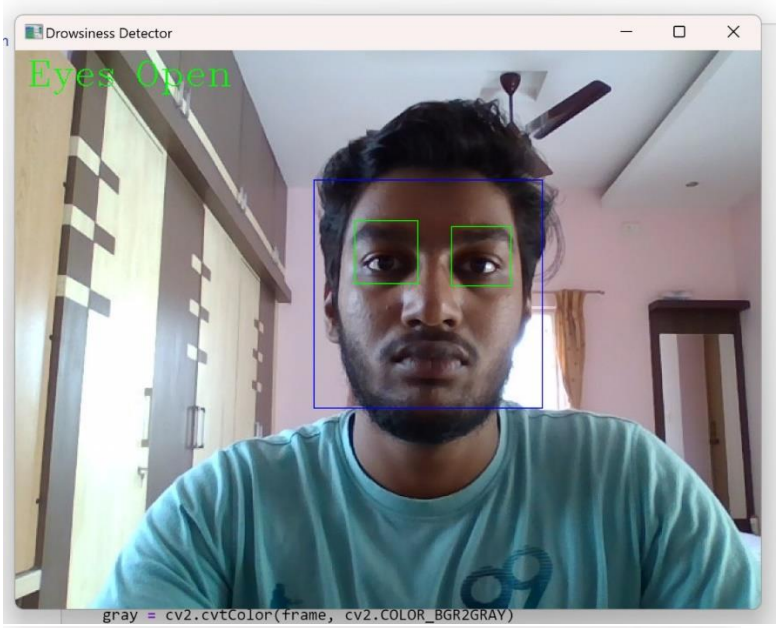
```
[ ]  prediction = model.predict([prepare(r"archive\train\Closed\_101.jpg")])
     np.argmax(prediction)
```

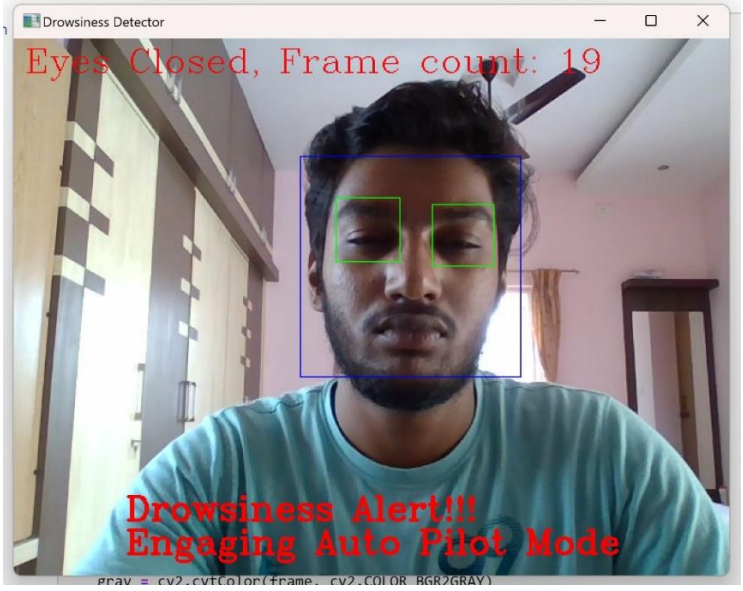
```
2
```

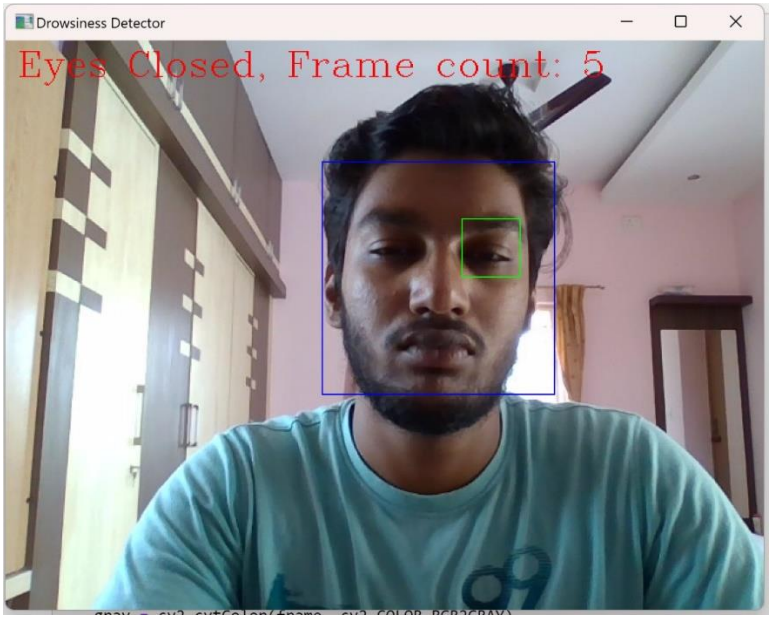
Figure 4Prediction

Results-

The Driver Drowsiness Detection and Alert System successfully achieved real-time monitoring and accurate detection of drowsiness indicators such as prolonged eye closure and yawning. The system consistently triggered timely alerts and auto pilot mode demonstrating high reliability under various conditions. Below is a screenshot illustrating the system's detection capabilities in action, highlighting the identified facial landmarks and the corresponding alert notification.


Drowsiness Detector
Eyes Open
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


Drowsiness Detector
Eyes Closed, Frame count: 19
Drowsiness Alert!!!
Engaging Auto Pilot Mode


Drowsiness Detector
Eyes Closed, Frame count: 5