NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

A STATE UNIVERSITY UNDER DELHI ACT 06 OF 2018,
GOVT. OF NCT OF DELHI
Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi-110078

Driver Drowsiness Detection using TinyML

Submitted By:
Aneesh Gupta (2021UCS1720)
Diksha Kapoor (2021UCS1668)
Lavanya Lall (2021UCS1682)

Submitted To:
Dr. Ankush Jain
Mr. Rohit Kumar

ABSTRACT

Long-distance travellers, particularly truck drivers, bus drivers, and taxi drivers, often face extended hours of driving, making them susceptible to sleep deprivation. The associated risks of fatigue and drowsiness can significantly compromise a driver's reaction time, attention, and overall ability to safely operate a vehicle. According to the National Safety Council (NSC), drowsy driving contributes to approximately 100,000 crashes, 71,000 injuries, and 1,550 fatalities annually. The prevalence of drowsy driving poses a substantial threat to road safety. The consequences of fatigue-related accidents underscore the urgent need for proactive measures to detect and mitigate driver drowsiness. In response to this challenge, we propose the development of a Drowsiness Detection System that leverages image processing and machine learning techniques to alert drivers when signs of fatigue become apparent.

1. INTRODUCTION

This project aims to create an intelligent system capable of identifying tiredness in drivers by monitoring their eye movements hence enhancing road safety by addressing the challenges posed by drowsy driving. Using Python libraries such as OpenCV for image processing and Keras for model development, the system will analyse facial features in real-time video feeds to detect instances where a person's eyes close, indicating potential drowsiness. Upon detection, the system will promptly notify the driver, providing a proactive approach to mitigate the risks associated with drowsy driving.

1.1 Objectives of the Project

The primary objectives of the Drowsiness Detection System are as follows:

- Develop a robust system for real-time monitoring of drivers' facial features, particularly focusing on eye closure.
- Implement image processing techniques using OpenCV to extract relevant information from video streams.
- Train a Convolutional Neural Network (CNN) model with Keras to classify eye states as open or closed.
- Integrate an alert mechanism to promptly notify drivers when signs of drowsiness are detected.

1.2 Significance of the Project

The significance of this project lies in its potential to enhance road safety by addressing the specific challenges posed by drowsy driving. By providing drivers

with timely alerts, the system aims to reduce the occurrence of accidents caused by fatigue and contribute to the overall well-being of travellers.

2. METHODOLOGY

In this project we aim to utilise Deep Learning, a subfield of Machine Learning to detect drowsiness and employ supervised learning - a subset of machine learning where the algorithm is trained on a labelled dataset, meaning that the input data is paired with corresponding output label. Within the realm of supervised learning, our focus is on classification tasks. Classification involves assigning predefined labels or categories to input data based on its features. In the context of drowsiness detection, the system is trained to classify instances of open and closed eyes, providing a binary classification outcome for each frame of the video feed.

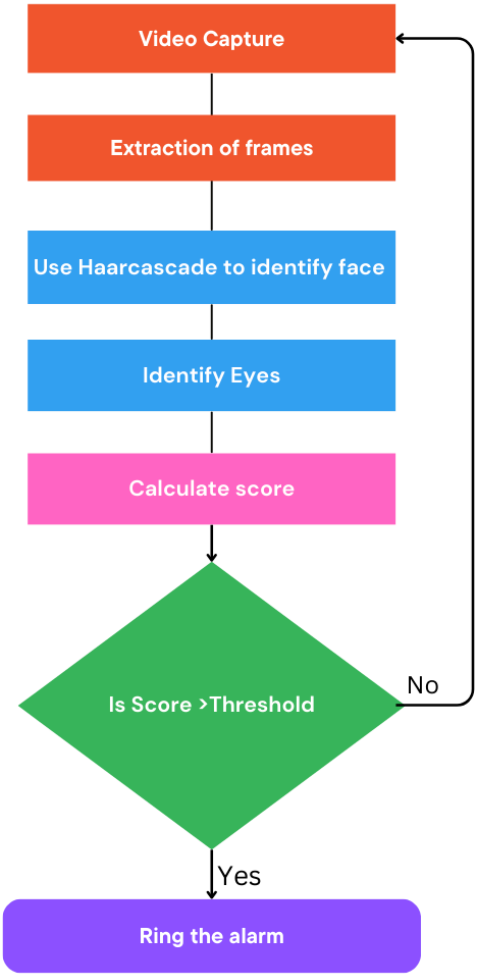


Fig 1. Flowchart of the proposed system

2.1 Algorithms Used - Deep Learning Model (Use of CNN)

Deep learning algorithms excel at automatically extracting hierarchical features from raw input, making them particularly well-suited for image and video analysis tasks. Convolutional Neural Networks (CNNs) are a specific class of deep learning algorithms designed for image processing and pattern recognition. In our drowsiness detection system, a CNN model is employed to analyse facial features and classify the state of the driver's eyes. This enables the system to discern subtle patterns indicative of drowsiness, providing a more sophisticated and accurate detection mechanism. To prevent overfitting, we incorporate a Dropout layer. This layer randomly drops a fraction of neurons during training, promoting model generalisation. Dropout is particularly beneficial in preventing the model from relying too heavily on specific features, ensuring it performs well on new, unseen data.

2.2 Dataset Used

In developing and training our drowsiness detection model, we leverage the Driver Drowsiness Dataset. This dataset encompasses a total of 2900 images, showcasing drivers in different levels of alertness:

- **Alert Drivers** - Images of drivers in an alert state, serving as the baseline for normal behaviour.
- **Drowsy Drivers** - Images capturing drivers exhibiting signs of drowsiness, including moments with closed eyes or other facial cues indicative of fatigue.

2.3 Dataset Preprocessing

In order to prepare the Driver Drowsiness Dataset for effective training, a series of preprocessing steps are applied to the images. This ensures that the data is appropriately formatted and normalised for optimal performance during training.

2.3.1 Colour to Grayscale Conversion

The images in the dataset are initially in colour, capturing a range of facial features. However, for the purpose of our model, colour information may not be crucial. Therefore, the images undergo colour to grayscale conversion. This not only reduces computational complexity but also focuses the model

on essential grayscale features, such as variations in pixel intensity, which are vital for detecting eye states.

2.3.2 Resizing

To facilitate uniform input dimensions for the Convolutional Neural Network (CNN), all images are resized to a predefined size. This step ensures consistency in the spatial dimensions of the input, allowing the model to process information consistently across all images.

2.3.3 Normalisation

Normalisation is a crucial step to standardise pixel values within a specific range. In our case, pixel values are scaled to a range between 0 and 1. Normalisation enhances the convergence of the training process and prevents certain features from dominating others during the learning phase.

2.4 SYSTEM ARCHITECTURE

The system architecture encompasses two main components: face and eye detection, and Convolutional Neural Network (CNN) for drowsiness classification.

2.4.1 Face and Eye Detection

The initial stage of our system involves the detection of the face and both left and right eyes in real-time video streams. This is achieved through the utilisation of Haar Cascade classifiers. The features are based on variations in pixel intensities and capture patterns such as edges, corners, and changes in texture. To speed up the computation of Haar-like features, an integral image is employed which allows for the rapid calculation of the sum of pixel values in any rectangular region of the image.

- **Face Detection:-** Employ the Haar Cascade classifier for face detection, which identifies regions of interest (ROIs) containing potential faces within each frame of the video stream. Draw a rectangle around the detected face to visually highlight the regions of interest.

- **Eye Detection:-** Utilise separate Haar Cascade classifiers for left and right eyes, identifying ROIs within the detected face regions.

2.4.2 Eye State Classification using CNN

The eye state classification module employs a Convolutional Neural Network (CNN) to discern whether the driver's eyes are open or closed. Extracting regions of interest for the right and left eyes, the CNN processes grayscale, normalised images resized to 24x24 pixels. Predictions are made using the pre-trained model, and the system adjusts a cumulative score based on the detected states. If the score surpasses a threshold, indicating potential drowsiness, an alert is triggered. The real-time display communicates the detected eye state, providing immediate feedback to the driver and enhancing overall safety awareness.

3. IMPLEMENTATION DETAILS

3.1 Technology Used

The integration of OpenCV, Keras, Pygame and TensorFlow/TensorFlow Lite empowers the drowsiness detection system with computer vision, deep learning capabilities, audio feedback, versatility, and adaptability to embedded environments.

- **Python Scripting:-** Python serves as the primary scripting language for the drowsiness detection system. Renowned for its readability, versatility, and extensive library support, Python provides a cohesive and maintainable codebase. Its scripting capabilities allow seamless integration with OpenCV, Keras, Pygame, and other libraries, fostering an efficient workflow for implementing and enhancing the system's functionality.

- **OpenCV for Face and Eye Detection:-** OpenCV, an open-source computer vision library, forms the backbone of the system's face and eye detection capabilities. OpenCV's efficiency in handling image processing tasks makes it an ideal choice for the initial stage of the drowsiness detection system, providing a foundation for subsequent analysis.

- **TensorFlow and TensorFlow Lite for MCU Deployment:-** TensorFlow, an open-source machine learning framework, and its lightweight counterpart, TensorFlow Lite, are instrumental in extending the deployment capabilities of the drowsiness detection system to microcontroller units (MCUs). By converting the trained CNN model into a format suitable for resource-constrained environments, TensorFlow Lite ensures the system's adaptability to embedded applications. This deployment strategy broadens

the system's reach, making it suitable for integration into vehicles and devices with limited computational resources.

- **Keras for CNN Model Training:-** Keras, a high-level neural networks API, plays a pivotal role in training the Convolutional Neural Network (CNN) model responsible for eye state classification. With its user-friendly interface and seamless integration with TensorFlow, Keras simplifies the process of building, training, and evaluating complex neural network architectures. Keras facilitates efficient experimentation with neural network configurations, enabling the optimization of the model for accurate and fast inference.

- **Pygame for Alarm Sound:-** Pygame, a cross-platform set of Python modules designed for game development, is utilised for its sound-playing capabilities in the drowsiness detection system. When signs of fatigue are detected, Pygame triggers an audible alarm to alert the driver. This real-time feedback mechanism enhances the system's effectiveness by providing an immediate alert that can be crucial in preventing potential accidents. Pygame's simplicity and ease of use make it a valuable addition for integrating audio feedback into the overall drowsiness detection solution.

3.2 Building and Training the CNN Model

The process of building and training the Convolutional Neural Network (CNN) model for eye state classification is a crucial step in the drowsiness detection system. The Python script below outlines the implementation using the Keras framework:

```
from keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Dropout,Conv2D,Flatten,Dense, MaxPooling2D

def generator(dir, gen=image.ImageDataGenerator(rescale=1./255),
shuffle=True, batch_size=1, target_size=(24,24),
class_mode='categorical'):
  return
gen.flow_from_directory(dir,batch_size=batch_size,shuffle=shuffle,color
_mode='grayscale',class_mode=class_mode,target_size=target_size)

BS= 32
TS=(24,24)
train_batch= generator('data/train',shuffle=True,
batch_size=BS,target_size=TS)
```

```
valid_batch= generator('data/valid',shuffle=True,
batch_size=BS,target_size=TS)
SPE= len(train_batch.classes)//BS
VS = len(valid_batch.classes)//BS
print(SPE,VS)

model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(24,24,1)),
    MaxPooling2D(pool_size=(1,1)),
    Conv2D(32,(3,3),activation='relu'),
    MaxPooling2D(pool_size=(1,1)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(1,1)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=
['accuracy'])
model.fit_generator(train_batch, validation_data=valid_batch,
epochs=15, steps_per_epoch=SPE, validation_steps=VS)
model.save('models/cnnCat2.h5', overwrite=True)
```

Code for Building and Training the Model

- **Data Generation:-** The 'generator' function is defined to facilitate the generation of batches of training and validation data. Leveraging the Keras 'ImageDataGenerator', the function rescales pixel values to a range between 0 and 1, and images are converted to grayscale to simplify processing. These batches are essential for training the CNN model to recognize distinct features associated with open and closed eyes.
- **Model Architecture:-** The CNN architecture is structured to capture hierarchical features from the input images. The model begins with a convolutional layer (Conv2D) with 32 filters, each of size (3, 3), and utilizes the rectified linear unit (ReLU) activation function to introduce non-linearity. Max-pooling layers (MaxPooling2D) follow, reducing spatial dimensions to retain essential features. This process is repeated, introducing additional convolutional layers to extract hierarchical representations of the input data. Dropout layers are strategically inserted to prevent overfitting by

randomly deactivating neurons during training. The final layers consist of a flattening operation to convert the multi-dimensional output to a one-dimensional array, followed by dense layers. The dense layers serve as fully connected layers, learning intricate patterns in the data. The first dense layer has 128 neurons with ReLU activation, and a dropout layer is inserted to enhance model generalization. The output layer, employing the softmax activation function, produces probabilities for the two classes: open and closed eyes.

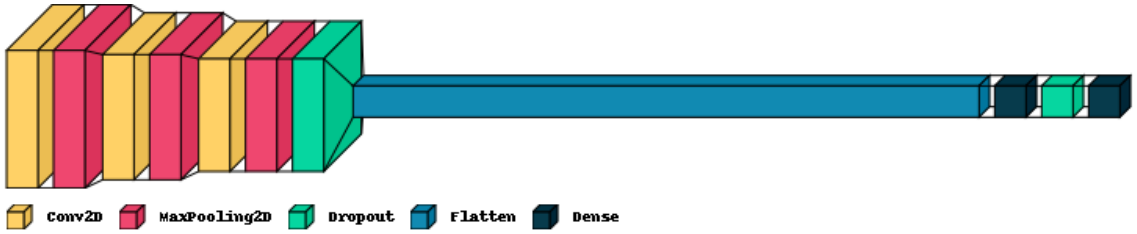


Fig 2. CNN Model for Eye State Classification

- **Training:-** The compiled model is then trained using the generated batches of training and validation data. Training occurs over 15 epochs, where one epoch represents a complete pass through the entire dataset. The specified steps per epoch and validation steps ensure that the model is trained and validated comprehensively.
- **Model Saving:-** Upon completion of training, the model is saved to a file ('cnnCat2.h5') for future use. This file encapsulates the learned parameters, enabling seamless integration into the drowsiness detection system during deployment.

3.3 Face and Eye Detection

The entire face and eye detection process is integrated into the real-time video stream captured by the webcam. As each frame is processed, the system dynamically identifies faces and eyes, creating regions of interest for the subsequent stages of drowsiness detection. This is achieved by leveraging the OpenCV library and utilizing Haar Cascade classifiers. Haar Cascade classifiers are robust machine learning objects employed for object detection, particularly faces and eyes. These classifiers utilize Haar-like features, which are patterns resembling the structure of edges, lines, and rectangles. The training process involves positive and negative images, enabling the classifier to learn discriminative features. This classifier identifies facial features, delineating the boundaries of the face within the video stream. Detected faces serve as regions of interest for subsequent eye detection.

```
import cv2
import os
from keras.models import load_model
import numpy as np
from pygame import mixer

mixer.init()
sound = mixer.Sound('alarm.wav')

face = cv2.CascadeClassifier('haar cascade
files\haarcascade_frontalface_alt.xml')
leye = cv2.CascadeClassifier('haar cascade
files\haarcascade_lefteye_2splits.xml')
reye = cv2.CascadeClassifier('haar cascade
files\haarcascade_righteye_2splits.xml')

lbl=['Close','Open']

model = load_model('models/cnncat2.h5')
path = os.getcwd()
cap = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
count=0
score=0
thicc=2
rpred=[99]
lpred=[99]

while(True):
    ret, frame = cap.read()
    height,width = frame.shape[:2]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces =
face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,2
5))
    left_eye = leye.detectMultiScale(gray)
    right_eye =  reye.detectMultiScale(gray)

    cv2.rectangle(frame, (0,height-50) , (200,height) , (0,0,0) ,
thickness=cv2.FILLED)
```

Code for Face and Eye Detection

This code snippet captures a frame, converts it to grayscale, detects faces, left eyes, and right eyes using Haar Cascade classifiers, and adds a filled rectangle at the bottom of the frame.

- **cap.read():** Captures a frame from the video feed using the VideoCapture object (cap). The frame is stored in the variable frame, and ret is a boolean indicating whether the frame was successfully read.
- **height, width = frame.shape[:2]:** Retrieves the height and width of the frame for later use.
- **gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY):** Converts the color frame to grayscale to simplify processing.
- **faces = face.detectMultiScale(gray, minNeighbors=5, scaleFactor=1.1, minSize=(25,25)):** The `detectMultiScale` function iterates through the video frames, identifying potential faces based on specified parameters such as minimum neighbors, scale factor, and minimum size which control the sensitivity and accuracy of the face detection. Identified faces are then marked with rectangles for further processing.
- **left_eye = leye.detectMultiScale(gray):** Uses a Haar Cascade classifier (leye) to detect the left eye within the grayscale frame.
- **right_eye = reye.detectMultiScale(gray):** Uses another Haar Cascade classifier (reye) to detect the right eye within the grayscale frame.
- **cv2.rectangle(frame, (0, height-50), (200, height), (0, 0, 0), thickness=cv2.FILLED):** Draws a rectangle at the bottom of the frame to display text notifications.

The identified faces and eyes are then utilized for eye state classification using the pre-trained Convolutional Neural Network (CNN) model.

3.4 Eye State Classification using CNN

This stage of the drowsiness detection system involves classifying the state of the driver's eyes, determining whether they are open or closed. This classification is achieved through the application of a Convolutional Neural Network (CNN) model previously trained on a dataset of images containing both alert and drowsy states. The following code snippet represents the steps for classifying the state of the driver's eyes using the pre-trained CNN model, adjusting the score, and triggering drowsiness alerts:

```
for (x,y,w,h) in faces:

  cv2.rectangle(frame, (x,y) , (x+w,y+h) , (100,100,100) , 1 )
```

```
    for (x,y,w,h) in right_eye:
        r_eye=frame[y:y+h,x:x+w]
        count=count+1
        r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)
        r_eye = cv2.resize(r_eye,(24,24))
        r_eye= r_eye/255
        r_eye=  r_eye.reshape(24,24,-1)
        r_eye = np.expand_dims(r_eye,axis=0)
        # rpred = model.predict_classes(r_eye)
        rpred = model.predict(r_eye)
        if(np.argmax(rpred, axis=1)==1):
            lbl='Open'
        if(np.argmax(rpred, axis=1)==0):
            lbl='Closed'
        break

      for (x,y,w,h) in left_eye:
        l_eye=frame[y:y+h,x:x+w]
        count=count+1
        l_eye = cv2.cvtColor(l_eye,cv2.COLOR_BGR2GRAY)
        l_eye = cv2.resize(l_eye,(24,24))
        l_eye= l_eye/255
        l_eye=l_eye.reshape(24,24,-1)
        l_eye = np.expand_dims(l_eye,axis=0)
        lpred = model.predict(l_eye)
        if(np.argmax(lpred, axis=1)==1):
            lbl='Open'
        if(np.argmax(lpred, axis=1)==0):
            lbl='Closed'
```

```
        break


    if(np.any(np.argmax(rpred, axis=0)==0) and np.any(np.argmax(lpred,
axis=1)==0)):

        score=score+1

        cv2.putText(frame,"Closed",(10,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)

    else:

        score=score-1

        cv2.putText(frame,"Open",(10,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)



    if(score<0):

        score=0

    cv2.putText(frame,'Score:'+str(score),(100,height-20), font,
1,(255,255,255),1,cv2.LINE_AA)

    if(score>15):

        #person is feeling sleepy so we beep the alarm

        cv2.imwrite(os.path.join(path,'image.jpg'),frame)

        try:

            sound.play()


        except:  # isplaying = False

            pass

        if(thicc<16):

            thicc= thicc+2

        else:

            thicc=thicc-2

            if(thicc<2):

                thicc=2

        cv2.rectangle(frame,(0,0),(width,height),(0,0,255),thicc)
```

```
    cv2.imshow('frame',frame)

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

        break

cap.release()

cv2.destroyAllWindows()
```

Code for Eye State Classification

i. **Image Preprocessing and Eye State Prediction**:- The extracted eye regions undergo preprocessing to ensure uniformity and optimal input for the CNN model. The preprocessed eye regions are then fed into the CNN model for classification. The model predicts the likelihood of the eyes being open or closed based on the learned features. This prediction is utilized to update a cumulative score that determines the driver's level of drowsiness. The classification results are used to update the system's score, indicating whether the driver's eyes are exhibiting signs of drowsiness.

ii. **Scoring Mechanism:-** Adjust the score based on the predictions for both eyes. If both eyes are classified as closed, increment the score; otherwise, decrement it.

iii. **Real-time Display:-** The real-time display presents the ongoing results to the user, showcasing the detected eye states and the current drowsiness score.

iv. **Drowsiness Alert:-** If the cumulative score surpasses a predefined threshold (15 in this case), indicating a potential state of drowsiness, the system triggers an alert. This alert may be an audible alarm (a sound alert using Pygame) to prompt the driver to regain alertness.

3.5 Converting Keras Model to TensorFlow Lite model

```
model = load_model('models/cnncat2.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()
open("mymodel" + '.tflite', 'wb').write(tflite_model)
```

This will create a file named mymodel.tflite which contains the tensorflow lite code.

To get our tflite file to run on a microcontroller, we need to save it as a constant byte array in a .h or .c file.

```
def hex_to_c_array(hex_data, var_name):
 c_str = ''
 c_str += '#ifndef ' + var_name.upper() + '_H\n'
 c_str += '#define ' + var_name.upper() + '_H\n\n'
 c_str += '\nunsigned int ' + var_name + '_len = ' + str(len(hex_data))
+ ';\n'
 c_str += 'unsigned char ' + var_name + '[] = {'
 hex_array = []

 for i, val in enumerate(hex_data) :
   hex_str = format(val, '#04x')
   if (i + 1) < len(hex_data):
     hex_str += ','
   if (i + 1) % 12 == 0:
     hex_str += '\n '
   hex_array.append(hex_str)

 c_str += '\n ' + format(' '.join(hex_array)) + '\n};\n\n'
 c_str += '#endif //' + var_name.upper() + '_H'
 return c_str

# Write TFLite model to a C source (or header) file
c_model_name="mymodel"
with open(c_model_name + '.h', 'w') as file:
 file.write(hex_to_c_array(tflite_model, c_model_name))
```

Code to create TensorFlow Lite Model

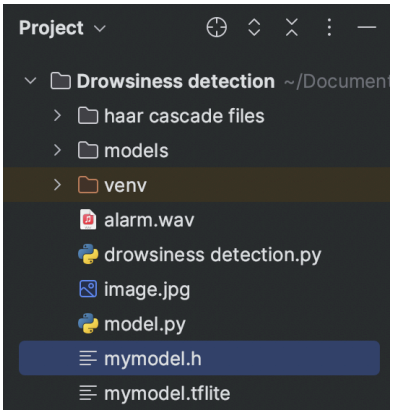


Fig 3. Tensorflow Lite model and .h file have been created

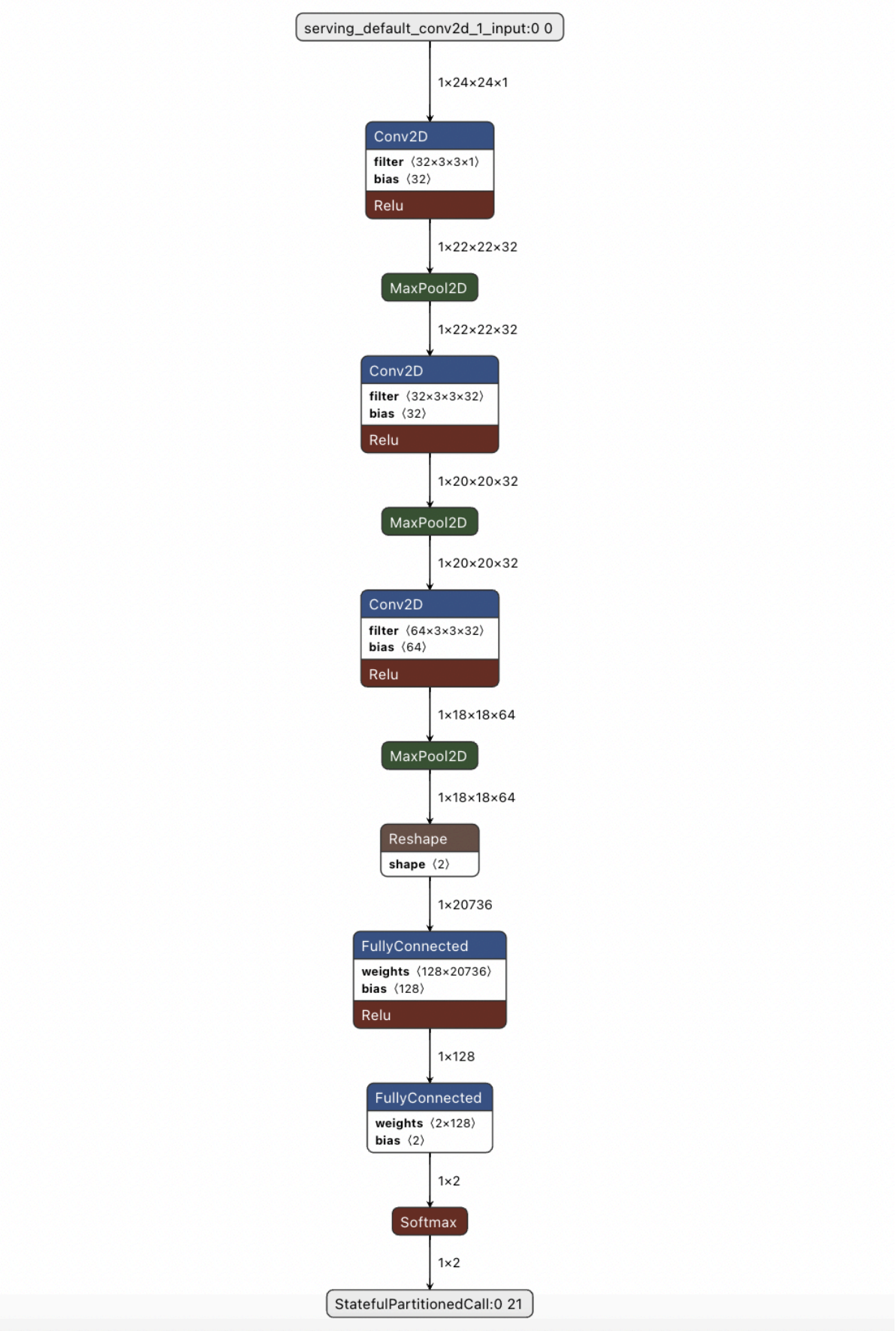
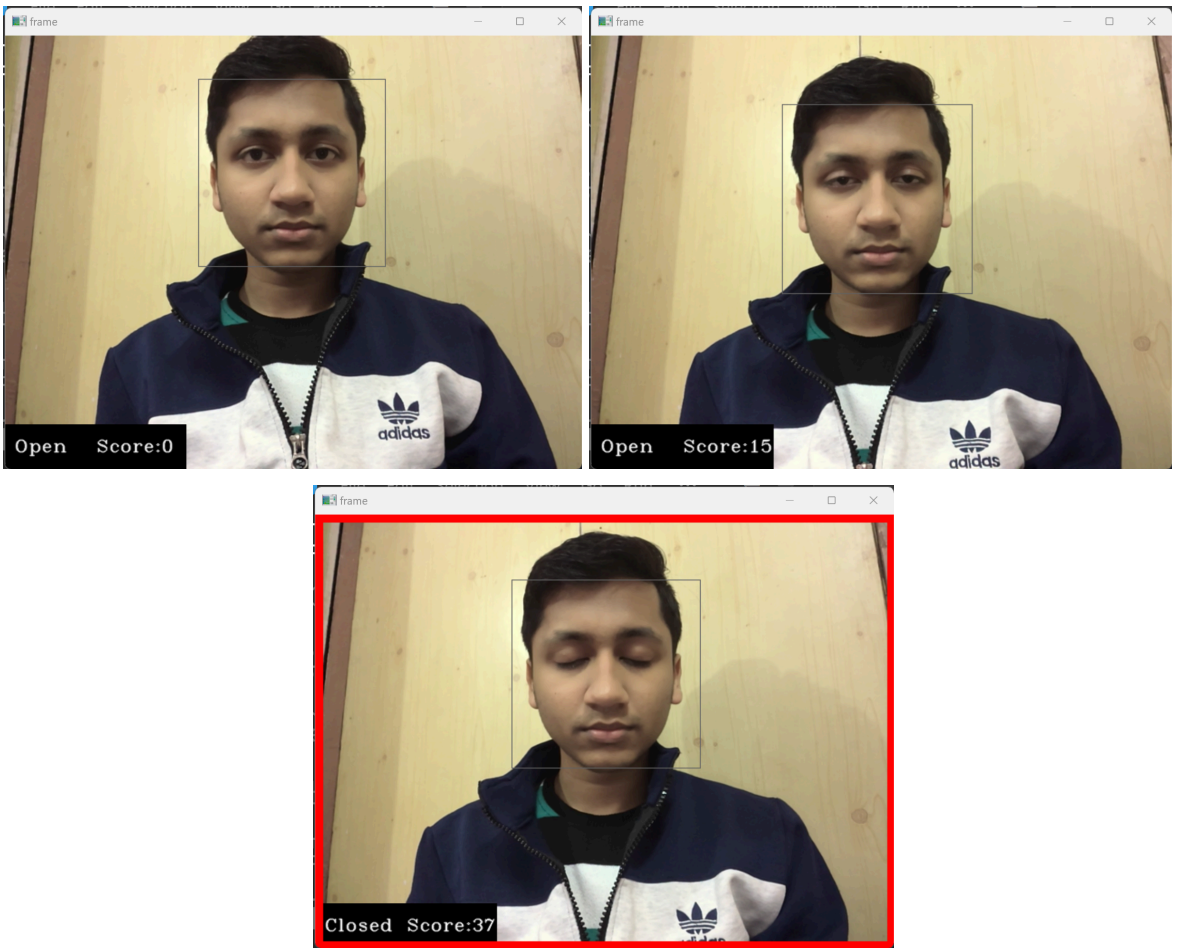


Fig 4. TensorFlow Lite model visualisation using Netron

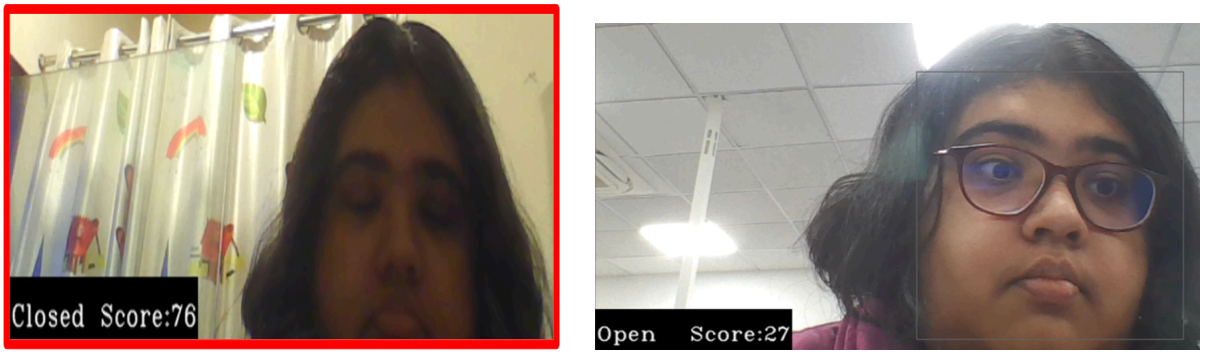
4. OUTPUT

The output of the drowsiness detection system is designed to provide real-time feedback to the user, alerting them to potential signs of drowsiness during driving. The system employs a combination of visual and auditory cues to effectively communicate the driver's level of alertness. The system dynamically updates and displays whether the driver's eyes are classified as "Open" or "Closed" in real time. A cumulative score, determined by the CNN model based on the duration and frequency of closed-eye states, is displayed on the screen. When the cumulative score surpasses a predefined threshold, indicating a potential state of drowsiness, the system triggers an alert to prompt the driver to regain alertness.



(i)

(ii)



(iii)

Fig 5. Output of Driver Drowsiness Detection System

5. CONCLUSION

In conclusion, the drowsiness detection system presented in this project enhances road safety, particularly for long-distance drivers. By leveraging technologies such as Convolutional Neural Networks (CNNs) and Haar Cascade classifiers, the system effectively identifies signs of driver fatigue through real-time analysis of eye states. The integration of OpenCV for face and eye detection, Keras for CNN model development, and Pygame for audio alerts ensures a robust and comprehensive solution which can potentially reduce accidents caused by driver fatigue.

GITHUB REPOSITORY

https://github.com/aneeshgupta25/Drowsiness-Detection-TinyML-Project-/tree/main