VIT BHOPAL UNIVERSITY

PROJECT EXHIBITION 1

(DSN2098)

INTERIM SEMESTER 2024-25

Drowsiness Detection System Using Facial Expression Recognition

Supervisor- Mr. Vishal Singh Bhati

Contributors:

Samarth Khandelwal	**Sarthak Rastogi**	**Adarsh Yadav**	**Sneh Thakkar**	**Kartikey Shastri**
23BCE10647	**23BCE10649**	**23BCE10598**	**23BCE10689**	**23BCE10710**

The current copy of Project is only intended for <u>Review-2</u> and not for the final submission.

ACKNOWLEDGEMENT

We sincerely express our gratitude to everyone who has supported and guided us throughout the initial and developing phases of our project i.e. "Drowsiness Detection System for Drivers." This project is part of Project Exhibition-1 at VIT Bhopal University and is currently under review for further development.

We are deeply thankful to our supervisor, Mr. Vishal Singh Bhati, for his invaluable guidance, constructive feedback, and continuous encouragement. His insights have been essential in shaping the direction of our project. His recommendations have helped us to learn more about the topic and the documentation work.

We also extend our heartfelt thanks to our friends and teammates for their collaboration, ideas, and dedication, which have been crucial in moving the project forward.

Lastly, we are grateful to the VIT Bhopal University for providing us with this platform to showcase our work and for offering the necessary resources and support throughout the process. We look forward to incorporating the feedback received during the review and taking our project to completion.

Regards

Samarth Khandelwal 23BCE10647

Sarthak Rastogi 23BCE10649

Adarsh Yadav 23BCE10598

Sneh Thakkar 23BCE10689

Kartikey Shastri 23BCE10710

CONTENTS

I

INTRODUCTION

1.1 PROBLEM DEFINITION

The facial expression recognition method gives very good interpretation about human behaviour and emotion. Its full potential goes beyond emotion classification; it can be used to predict critical states of behaviour, like drowsiness, preventing mishaps and safety. One of the most relevant applications is drowsiness detection, especially in dangerous scenarios like driving or prolonged work sessions. Drowsiness is indeed one of the serious threats to not only personal wellbeing but also public safety and productivity of the workplace. Mistakes by an overfatigued worker in industrial environments may mean disaster, and drowsiness-related road accidents are very common.

Current drowsiness detection systems utilize almost exclusively the most simplistic signals- namely, yawning, head nodding, or eye closure. These occur, however, when it is already too late to take evasive actions. There exists a delay between when drowsiness sets and when the system responds to this drowsiness. A next-generation system that uses facial expression recognition can therefore exhibit a more granular and real-time sense of the state of alertness of the user. Rather than waiting for these obvious giveaways, such as yawning, the system would follow micro-expressions, subtle eye movements, blinks, physiological features such as heart rate and speedometer readings. These observations will also be close enough to pick up on early signs of fatigue before it becomes dangerous.

The problem within these objectives is in making sure that such a system can operate with a tolerance for inaccuracy under varied types of conditions. For instance, it should still work properly if the user is wearing glasses or a face mask both of which obscure important facial cues. With advanced computer vision methods and machine learning models, the system can adapt easily for the first time appearance of the users as well as varying lighting conditions. It should be designed to detect even the smallest variations from normal alertness, for instance, in rate and speed of blinks, drooping of eyelids, and change in heart rate or other physiological factors.

Perhaps its response mechanism is of equal significance. When sleepiness begins overpowers, it would immediately act on counter measures with the playing of stimulating alarm for invigorating the brain and keeping the user alert. Other stimuli could be added for drivers, including vibrating the seat or giving a gentle auditory alarm, which would make users more attentive. This intervention is thus done in real time to ensure that the user is alert enough before accidents or errors may occur. Relying on isolated system won't do any good to mankind, so we need signals to ensure other vehicles passing by are aware of the drowsy state of the driver.

Thus, the proposed system would continuously observe facial expressions and respond to them in time. This approach could significantly reduce the overall risks associated with fatigue-related accidents, and can be implemented in any environment that is of high risk-be it a vehicle or a workplace, saving lives and boosting productivity eventually. Thus, this approach is an innovative way to proactive safety and marks the first critical step towards behaviour monitoring.

1.2 MOTIVATION AND ETHICS

This project is an inspiration to the alarming numbers of drowsy driving that still one of the significant and neglected causes for road accidents. It accounts for about 20% of all the motor vehicle crashes throughout the world.

The urgency to take effective measures towards such an accident is growing day by day. In India, too, it has been very serious; in 2022, it reached a tremendous 4,61,312 road accidents, with a total of 1,68,491 deaths and 4,43,366 injuries. (According to Press Information Bureau)

Of these incidents, drowsiness is the most important factor as most drivers fail to recognize the extent of impairment caused by drowsiness and mistake it for driving under the influence of alcohol. This proposed bridge takes the gap in today's safety systems and fuses the use of cutting-edge technology that not only detects when one is drowsy but in real-time prevents continuous driving even for the driver alone but is enhanced to alert surrounding vehicles, hence creating a safe driving environment for all.

1.3 APPLICATIONS

This system can be particularly used for awakening or helping the driver to remain in senses which would reduce road accidents and fatalities.

This system is most important for heavy load commercial and public vehicles as they usually drive for more than 14 hours which might cause accidents and cause harm to drivers, passengers, goods, other vehicles on road and even infrastructure, in case of huge accidents.

Not only in driving systems, facial recognition system can be adapted as per needs and can be further be used for monitoring other facial detection and drowsiness detection objectives, such as:

1. Workers in factories working for prolonged hours may feel drowsy which might cause injuries.
2. Facial recognition can be used for educational purpose. A student pretending to study can be caught using his/her eye movements.
3. Facial recognition at malls can be used to prevent thefts.
4. Can be used in surveillance systems.

1.4 CURRENT GAPS AND CHALLENGES

Gap 1: Current systems fail to integrate heart rate sensors. It may very well be possible to analyse physiological data associated with the detection of drowsiness.

We can integrate heart rate sensors such as smartwatches' heart rate variability measures and tag this with our facial recognition-based system in order to increase the precision of detecting alertness in drivers. Since heart rate variability already acts as a good indicator of physiological states like drowsiness, supplementation with visual cues would enhance the accuracy of detection. It can also provide real-time feedback to drivers, assisting them in staying awake. Moreover, such an integration would make the system more flexible by allowing the users to track their level of drowsiness and complement facial recognition ***[3] [7] [15]***.

Gap 2: Although the discovered existing drowsiness detection models fail at low light, this further limits the proper usage of these models in real world applications like night drives.

It can be profoundly boosted to offer greater robustness and real-world application by enhancing it to work well under low-light conditions. This may be done either by simulating low light training data or by infusing infrared and thermal sensors for drowsiness assessment at night or in dimly lit environments. The risk of drowsiness is found to have a high association with night drivers; thus, improving road safety ***[2][6][13].***

Gap 3: The current models do not consider driving speed while detecting the drowsiness, thus neglecting one of the most important cues.

Adding speed data to our system, which is used for drowsiness detection, would provide potentially invaluable information about the behaviour of the driver. Sudden dives or erratic rate patterns often accompany decreased alertness. By merging real-time data of speed with heart rate and facial recognition, an all-rounded system can be provided that detects drowsiness earlier and with high accuracy. As such, the holistic approach may also provide real-time feedback to drivers to forestall accidents since they are alerted of changes in their driving patterns ***[4][9].***

Gap 4: The systems right now are isolated and fail to communicate with nearby cars to alert them and thus poses threat to overall safety, since it only alerts the driver and passengers.

Now, the proposed benefit is on the integration of V2V technology that will expand our safety net system and alert other surrounding vehicles that the driver is dozing off, thereby the nearby drivers being alerted automatically via emergency light flashing or distress signals through V2V technology. This collective safety mechanism will drastically reduce the risk of having drowsy driving accidents to endanger travellers on roads.

1.5 NOVELTY

Our project is innovative in the sense that we integrate a real-time heart rate monitoring scenario. We are making use of data of wearable devices, say smart watch or fitness band or mobile applications, which continue to track the driver's alertness based on his physiological state, such as heart rate. This information we have tried to integrate with visual cues and the output comes out much more accurate in drowsiness detection. The two sources of data, studied together, helped us to make a much more comprehensive and efficient system for drowsiness detection.

We have also ensured that the solution proposed performs equally well at night, during low-light driving conditions. Most systems are failing under present conditions with little or not enough illumination at night. We tested and trained for such challenges in careful deliberations and at all times ensure that the system will work regardless of the time or lighting, making it rather robust for real-world driving scenarios.

Another feature that is quite relevant to the analysis of a driving pattern is the data from the speedometer of the vehicle. Monitoring the driver's changes in speed and connecting these to the symptoms that would suggest sleepiness reveals some patterns about the driver. Further help to make the detection much more accurate for their fatigue/drowsiness and risks in risky behaviours much sooner comes from the added layer of speedometer information.

We have further enhanced the safety quotient by integrating a technology that will alert other cars present in the vicinity that the driver is falling asleep. This is done through light indicators or other signals, which create the alert for other people on the road in cases where there's potential risk.

We may also set up a network among vehicles such that it does not only use hazard lights as an indicator but also pings up on nearby devices of Android Auto or Apple Carplay or other similar systems. This way, we make sure the safety of the driver and take proper care to design a safer and better driving environment for others around.

Lastly, our solution integrates several methods of detection in one system that will deliver much better results than the others. Integrating facial recognition with heart rate and speed data gives us an all-inclusive approach of drowsiness identification that is able to detect it accurately and reliably. We believe that combining technologies will be efficient in overcoming the challenges of driver fatigue by providing an even more effective approach towards safety on the roads.

1.6 METHODS

OpenCV plays a central role in our system since it is the most powerful popular library for computer vision applications. It allows capturing frames from the webcam, thus permitting real-time face detection with landmark extraction and eye tracking. Driver's facial expression may be analyzed based on using techniques like Haar cascades or HOG-based face detection to develop an efficient framework for observing the alertness of the driver and detecting drowsiness *[1][2][3]*.

Thus, Haar cascades become a lightweight and efficient method for object detection within video frames; it scans regions to detect faces and eyes. This is the reason this very useful in real-time because it may provide timely alerts when the driver's eyes are closed for some duration of time, indicating possible fatigue *[4][5]*. In this manner, Dlib enhances this ability by allowing users advanced techniques for facial landmark detection. The pre-trained models can tell features much more precisely like eye positions to calculate EAR or the openness of the mouth for yawning, all of which make the system more accurate *[7][8]*.

The core component of the system is the Eye Aspect Ratio, which takes in computation the distance between the eyelids compared to a threshold. This does give the system a quite precise way of assessing drowsiness with minimal overhead in computation. This non-learning method integrates well with other detection techniques so as to ensure strong performance. For further improvement, an extraction of facial features from video frames is carried out using convolutional neural networks (CNNs). CNNs identify fatigue indicators such as closed eyes or yawning. They are trained on labeled data and thus perform well even under changing light conditions, ensuring consistent and accurate detection *[11][12].*

To capture temporal patterns, the system relies on the design of Long Short-Term Memory (LSTM) networks. LSTMs learn sequences of blinks and yawns that signify the growth of a pattern of fatigue over time and thus increase the reliability of drowsiness detection. Simultaneously, YOLO is an approach toward fast object detection with low latency designed to enhance the real-time capability of its performance in tracking eyes and mouth facial features.

Yet another rich enhancement to the system is through Support Vector Machines (SVMs), which classify binary states, such as "sleepy" versus "awake," working on extracted features like EAR or facial landmarks. This proves very effective when well-curated datasets are used, and the early detection of sleepiness is possible. These advanced methods are built on frameworks like Keras and TensorFlow, which give the versatility to create sophisticated architectures. Even though Keras makes the implementation much easier, TensorFlow provides a facility of development in complex models that can be utilized as per the requirement of the system in detection *[12][18].*

MobileNet was used as a lightweight CNN architecture for the system for efficiency on mobile and edge devices. This allows for real-time drowsiness detection with minimal requirements on the hardware and, therefore, is suitable for usage on platforms like Raspberry Pi or Android systems. These devices are further integrated with frameworks like Flask, which facilitates a smooth flow of communication between the detection system and servers at the backend, thus making sharing of drowsiness data with alert systems or even cloud dashboards a pretty easy affair *[11][18].*

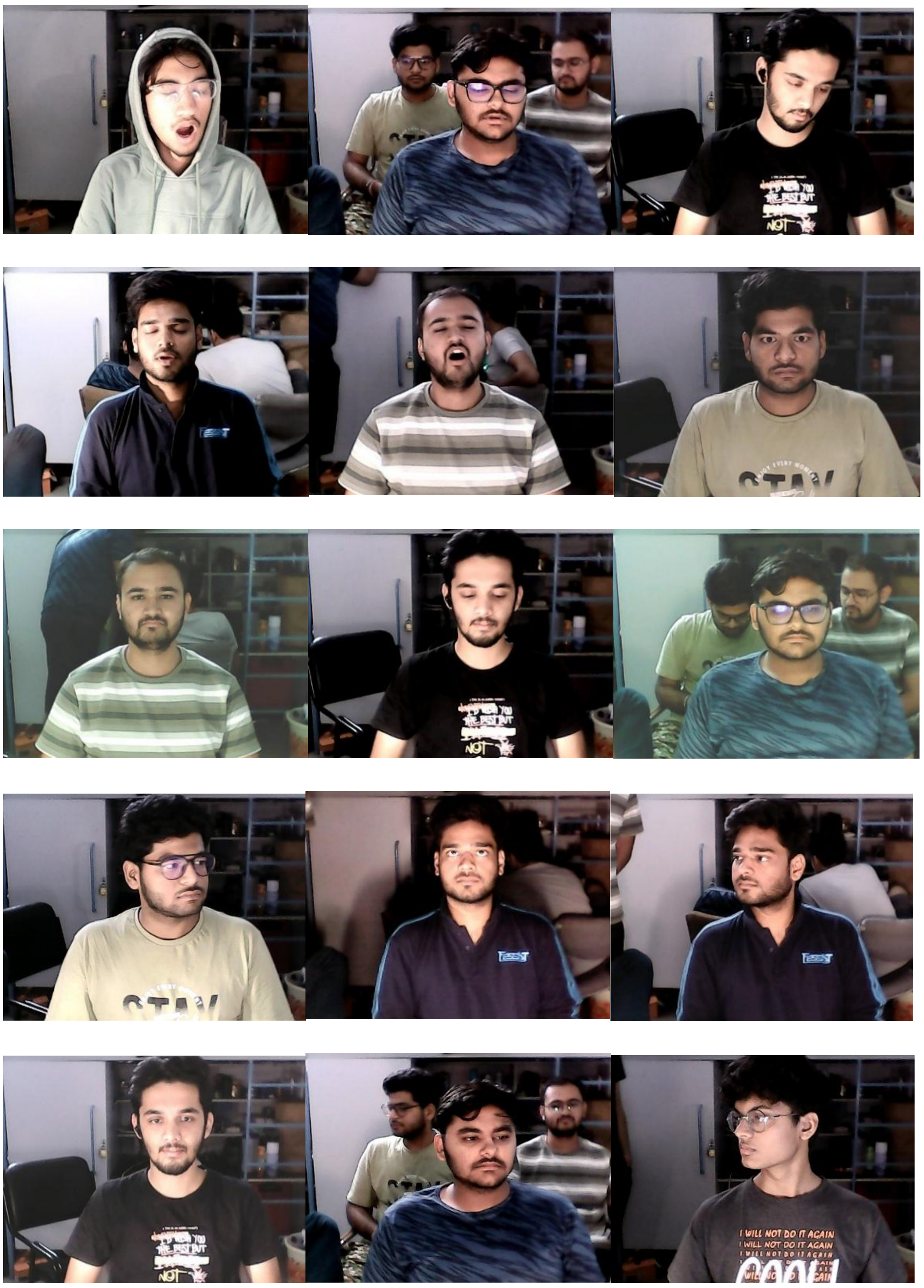
Low latency, real-time transmission of alerts between IoT devices and the central monitoring systems are ensured by the MQTT protocol in overall safety with timely warnings. The primary edge devices are Raspberry Pi that run detection algorithms along with capturing video frames from webcams. The Arduino complements the system through its capability of triggering other external alerts through inputs from the detection module ***[14][16].***

The other features of this system include the detection of blinking of the eyes. The system observes the number of blinks made in a minute by the driver. If the blink rate is decreased noticeably, the system will start to alert the driver. Infrared cameras make the functionality advanced as they are known to work without flaws when it is very dark outside, even when used at night. This automatically ensures that the system works very accurately and effectively in any condition ***[5][7][10].***

To address changing scenarios, the system has a face mask detection that addresses eye movement and blinking rather than the mouth. This way, the drowsiness detection will still be efficient for a masked driver. For the last aspect, the system makes use of Bluetooth connectivity that sends notifications to other devices such as a smartphone or an in-car infotainment system; therefore, timely notification are provided to drivers to take corrective measures which enhance road safety ***[8][12][17].***

1.7 DATASET

Custom Dataset **DATASET-1**



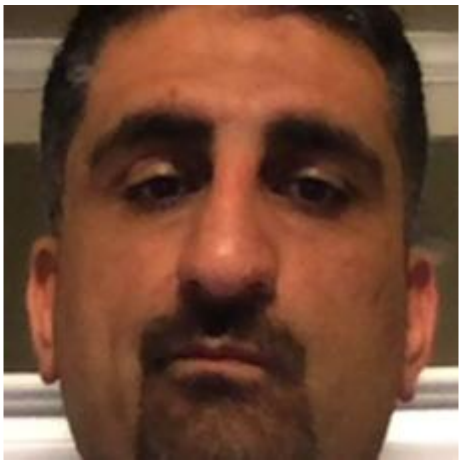
Full Face(KAGGLE) **DATASET-2**

Link: http://mrl.cs.vsb.cz/eyedataset

Drowsy

Not Drowsy:

Drowsy:



Not Drowsy:

II

RELATED WORKS

2.1 FACE DETECTION METHODS

OpenCV is also a crucial tool for face recognition and detection purposes, making use of various advance techniques like the feature of detecting faces through Haar
Cascades and recognizing faces through Histograms of Local Binary Patterns (LBPH). In addition, its Histogram of Oriented Gradients (HOG)
approach solves problems concerning the many orientations in which the pattern will be
detected, thus making OpenCV a very robust real-time application.

Dlib completes the palette of OpenCV tools by using HOG features along with Linear SVM for real-time face detection, gaining much impressive accuracy. It further complements system reliability
by providing a deep learning-based face recognition module that excels in detecting facial
landmarks to ensure robust performance under dynamic scenarios.

The Face_recognition library extends Dlib's backend for even the most simplified yet effective way to perform face recognition. It encapsulates encoding faces, allowing comparisons
between pictures, and simplifies the recognition of individuals in photographs to a great extent with precision and accuracy.

TensorFlow and Keras are the keys to CNN-based models for facial recognition. Implementations like FaceNet, through the use of embedding mappings and learned classifier
strategies, achieve remarkable accuracy and versatility in application.

PyTorch has formed a widely used flexible framework in the design process of tailored CNNs in face detection and recognition. Through such optimization
capabilities, it is increasingly becoming the darling of developers.
Further, the library integration can be extended to include, for instance, DeepFace with deep learning-based face recognition tasks.

YOLO attempts to reinvent how object detection is being done through real-time
efficiency. Variants like YOLOv3, YOLOv4, and YOLOv5 are
particularly tailored for detection purposes when it comes to faces and
features that need to be picked up quickly, so
fast applications are required without sacrificing accuracy.

The Microsoft Azure Face API takes facial recognition to the cloud and offers advanced
capabilities with regard to landmark generation and the precise identification of
coordinates. The benefit for projects is particularly valid and extensive as regards high reliability and scalability in face detection and recognition.

Amazon Rekognition provides face detection extension facilities, for instance, emotion and
age/gender classification by facial analysis. In addition, it can identify a specific person in media content, such as photos of celebrities for actor identification, so it caters
to a wide range of diversified advanced facial recognition needs.

The Google Cloud Vision API excels in face detection with deep insights into facial
emotions. Such an API is of extreme value for applications that require rich recognition
capabilities and therefore yield good data to enrich the experience of users in complex systems.

SimpleCV offers many computer vision algorithms and simplifies access to them for a user-friendly design. It utilizes foundational techniques, such as the Haar Cascades for face detection, and gets one to the core without much coding.

DeepFace is known for its flexible support for multiple backends, which provides unparalleled flexibility in face recognition. The flexibility behind the algorithms' use as per the requirement caters to novice and experienced users, hence offering applications from different projects.

MediaPipe focuses on real-time mesh generation and facial landmark localization, which is a really precious asset for interactive applications. It is actually relatively adaptable and can be easily inserted into most computer vision systems to enhance real-time performance and provide the capabilities for user interaction.

2.2 NON-LEARNING AND LEARNING BASED METHODS

[1] Using long short term memory and convolutional neural networks for driver drowsiness detection

[1] Describes a driver drowsiness detection system based on LSTM and C-LSTM networks based on video frames to identify alertness using eye images. Experimented results for 38 participants, eye movements were recorded using a dual-camera setup along with EEG data classified states to either "alert" or "drowsy."
The accuracy that can be achieved for the regular LSTM model, which focuses on 48 x 48 pixel eye patches, is 82%. In contrast, the accuracy of the C-LSTM model, when its input is taken as 2-D and using a CNN for probing could range between 95% and 97%, depending upon the time window used. The comparisons with the traditional eye-tracking systems relying on SVM and Random Forest classifiers that measure eye gaze and blinking showed that the C-LSTM model surpasses them, reaching an accuracy of 97.87% with a 1 second window, that is above the top accuracy of traditional systems at 85.9%.
The system was trained on an NVIDIA GTX 1070 GPU: 2000 epochs for LSTM and 1000 for C-LSTM. However, C-LSTM is significantly more computationally expensive because it samples at a much higher rate and the performance improvement makes it worthwhile.

[2] Automated Drowsiness Detection For Improved Driving Safety

[2] Describes automated drowsiness detection system for the driver that entails computer vision and machine learning. In contrast to early work, which has used predefined behavioral indicators (blink rate, yawning), these researchers analysed actual facial behaviour during drowsy episodes. Researchers have developed classifiers for the 30 facial actions as described in FACS from a spontaneous expression dataset. Head movements were also monitored through eye tracking and accelerometers.
For driving video game experiments, the system was able to predict episodes of sleep and crash with as much as 96% accuracy within individuals and over 90% across different subjects. It is the highest reported for real-time drowsiness prediction and offers valuable new insights into human behaviour while driving drowsy.
The study used a facial action coding system to determine the principal actions associated with drowsiness. AdaBoost as well as multinomial ridge regression machine learning algorithms were used to predict alertness based on the change in facial expressions. There was robust correlation between specific facial actions and drowsiness states, and expressions should be considered as a variety rather than merely related to ocular behaviours.
Concerning the prediction performance, the classifiers showed great results: around 92% for AdaBoost and 94% for multinomial ridge regression. In all subjects, blink/eye closure was the most predictive action.

[3] Real-Time System for Driver Fatigue Detection Based on a Recurrent Neuronal Network

The authors of *[3]* demonstrate a real-time drowsiness capture system exploiting an RNN, capturing eyes close, yawned, and head tilted actions as facial landmarks. The proposed system seems to be the latest version in this realm of research work; hence it is expected that the current solutions rely heavily on intruding physiological sensors or basic image processing techniques leading to poor user experience in some cases or degrade under real-world conditions. They intend to formulate an anonymous and low-cost method by capturing drowsiness through facial landmarks such as eye closure, yawning, and head tilts.
To bridge this gap, the authors develop a real-time processing scheme of sequences of facial images using 3D CNNs and RNNs in order to accurately detect even subtly manifesting

drowsiness. This model was trained with an accuracy rate of 92 % on the NTHU-DDD dataset as it was demonstrated to benefit significantly from improvement in detection under low light conditions, varied kinds of driving environments, and varied driver profiles.
It points out the practical applicability of building an application with efficient capability that could be managed by minimal hardware by providing an alternative solution particularly at the cost when compared to the more expensive ones where application depends so much on considerable hardware. And most importantly, the system successfully bridges the gap towards making the detection of driver fatigue reliable, scalable, and non-intrusive.

[4] Non-Intrusive Detection of Drowsy Driving Based on Eye Tracking Data

The paper *[4]* under discussion employs eye-tracking data as a non-invasive technique to measure driver drowsiness. While this is, in most cases, the cause of majority vehicular accidents, it remains largely ignored by many, and this needs attention for such tragedies to be avoided. Experiment Results: In this research, the authors made use of 53 subjects in a simulated driving scenario by comparing the eye-tracking data with EEG signals, which was used as a reference for monitoring vigilance
In order to assess the strength of their approach, the team applied classifiers based on Random Forest and Support Vector Machine. Their best RF improved over SVM accuracy range from 88.37% to 91.18%, whereas SVM could improve up to 77.12% to 82.62

[5] Detecting Driver Drowsiness with Multi-Sensor Data Fusion Combined with Machine Learning

[5] Describes A novel system for detecting driver drowsiness using machine learning is presented in the paper. The system used here involves a webcam for facial data acquisition and head movement information through a micro-Doppler radar. The system would aim at monitoring all the signs of drowsiness, like eye blinks, yawns, and head drops, and warn the driver before falling asleep. Researchers bridged the gap that seemed to prevail in previously developed drowsiness detection techniques that were either too intruding or not real-time; they used two sorts of sensors and deep convolutional neural networks in order to reduce the problem and finally detect drowsiness.
The system was experimentally tested in a vehicle and shown to detect drowsy driving conditions with a more than 95% accuracy. The unique feature of the above system is real-time alert and alert triggering mechanism in terms of vibrating the steering wheel and giving messages on the display when it detects a sleeping state. Providing image data from the webcam with the radar signals offers a more accurate assessment of driver alertness than a single-sensor-based system. Such multi-sensor machine learning-based systems have the potential to reduce the occurrence of accidents involving drowsy driving and, thereby make the roads safer.

[6] Driver Drowsiness Detection Using Ensemble Convolutional Neural Networks on YawDD

[6] Describes A driver drowsiness detection system, based on the application of Ensemble Convolutional Neural Networks (CNNs) on the YawDD dataset, is proposed in this paper. The authors had attempted to mitigate improvements in detection accuracy due to failures encountered with standard CNN models for yawning, eye closure, and other manifestations of drowsiness. They found a gap in their performance in cases of inconsistency under varying conditions, like pose variations and occlusion in an individual CNN model.
The authors make use of an ensemble strategy in such a way that the outputs of different CNN models are brought together to get better accuracy. Four CNN models-CNN1, CNN2, CNN3,

and the Ensemble CNN-were made. The best performance comes from the ensemble model with an F1 score of 0.935. This has proved better than other models having CNN models with an F1 score of 0.912 for CNN3. The paper gives an informative account of the successful application of ensemble methods that yield better results in real-world driver drowsiness detection.

[7] A real-time driver drowsiness detection system is aimed to be developed using deep learning techniques for a research paper. Lately, the rate of road accidents due to driver fatigue has risen significantly at times, leading to serious injuries or even death. The architecture used in this method includes CNN, MobileNet, and Single Shot Multibox Detector (SSD). It trains critical indicators of drowsiness such as eye closure and yawning with the help of a dataset that includes open/closed eyes, as well as yawning/no-yawning faces. It processes the real-time video streams for the analysis of driver state and informs him whenever it detects signs of drowsiness.
An important feature of the system is its affordability and efficiency. It can be run on low cost devices like a Raspberry Pi, or even a more basic IP camera; it is thus applicable for real world deployment. The model was learned on about 4,500 images and evaluated using 600 extra images with great accuracy. Though it is very simplified, the system presented above is built to perform in any environment with minimal processing power hence is designed to work in real-time with inexpensive equipment.

[8] This paper is the automated drowsiness detection system for the driver that entails computer vision and machine learning. In contrast to early work, which has used predefined behavioural indicators (blink rate, yawning), these researchers analysed actual facial behaviour during drowsy episodes. Researchers have developed classifiers for the 30 facial actions as described in FACS from a spontaneous expression dataset. Head movements were also monitored through eye tracking and accelerometers.
For driving video game experiments, the system was able to predict episodes of sleep and crash with as much as 96% accuracy within individuals and over 90% across different subjects. It is the highest reported for real-time drowsiness prediction and offers valuable new insights into human behaviour while driving drowsy.
The study used a facial action coding system to determine the principal actions associated with drowsiness. AdaBoost as well as multinomial ridge regression machine learning algorithms were used to predict alertness based on the change in facial expressions. There was robust correlation between specific facial actions and drowsiness states, and expressions should be considered as a variety rather than merely related to ocular behaviours.
Concerning the prediction performance, the classifiers showed great results: around 92% for AdaBoost and 94% for multinomial ridge regression. In all subjects, blink/eye closure was the most predictive action.

[9] Real-Time Driver-Drowsiness Detection System Using Facial Features

[9] Describes A real-time driver drowsiness detection system called DriCare is proposed in this paper, using facial features to monitor driver fatigue. The purpose is developing a non-intrusive system that can determine drowsiness through video images, mainly on the facial expressions of yawning, blinking, and eye closure duration. The authors found gaps in prior systems due to: low accuracy when applied in complex environments and tracking capability. They developed the MC-KCF algorithm, an integration of CNN and KCF to address these problems, and therefore improve facial tracking under varying conditions.
DriCare calculates three important metrics to evaluate the driver's state: blinking frequency, eye closure duration, and yawning. If any of the above metrics has crossed the assumed

threshold, the system generates an alert message to the driver. The experimental results showed that the developed system could predict the status of the driver with an accuracy level of about 92% and over different environmental conditions-like driving in low-light conditions-with good robustness. The authors have identified the shortcomings of traditional systems based on fatigue detection because of facial key-point detection and multi-feature fusion, which therefore presents a practical and scalable solution for real-time monitoring. This research has a great potential in reducing accidents caused by drowsiness, improving road safety, and alerting drivers in time.

[10] HybridFatigue: A Real-time Driver Drowsiness Detection using Hybrid Features and Transfer Learning

[10] Describes HybridFatigue, which is a real-time driver drowsiness detection system using hybrid features and transfer learning. In this, the system utilizes facial landmark information combined with eye movement patterns, enriched by transfer learning techniques. The authors called out the issues with traditional systems in terms of the accuracy of feature extraction and classification. They found that variations in lighting mainly affect these systems, so they employed pre-trained models appended with custom feature extractions for better adaptability. The proposed system therefore focuses on detecting fatigue, using the indicators blinking rate, eye closure duration, and head movement, through deep learning models. Experimentation Demonstrations of high accuracy and real-time performance make HybridFatigue a practical solution in real-world applications. Transfer learning enabled the system to generalize well across different individuals and environments without requiring a deep retraining process. This work contributes towards better driver safety as it offers a very scalable and efficient mechanism for detecting drowsiness that can be readily added to existing driver-assistance systems.

[11] Robust Drowsiness Detection for Vehicle Driver using Deep Convolutional Neural Network.

[11] Describes A more reliable drowsiness detection system for vehicle drivers is reported using deep convolutional neural network, DCNN. Gaps in the previous methods, that suffered low accuracy and hence were computationally expensive and performed poorly under changing lighting conditions, were what the authors aimed to address. This technique utilizes head pose estimation and pupil detection and draws face regions using the frame aggregation strategy to compensate for issues, such as light reflection and shadows.
A cascade of regressors is used for enhancing facial landmark detection. A robust pupil detection is done, overcoming illumination, blur, and reflection artifacts, by applying the DCNN. While using batch normalization at training, it was demonstrated to be more stable and less sensitive to the initialization of its parameters. The experiments were very extensive and showed that the proposed approach outperforms the earlier solutions with an accuracy of 98.97% at a frame rate of 35 fps.
In summary, this paper contributes to the body of knowledge by enhancing real-time driver drowsiness detection through a methodology with high accuracy and efficiency. Such a methodology has the potential to enhance safety on roads by early fatigue detection.

[12] Condition-Adaptive Representation Learning for Driver Drowsiness Detection Using 3D-Deep Convolutional Neural Networks

[12] Proposed a condition-adaptive representation learning framework for detection of driver drowsiness with the help of a 3D deep convolutional neural network is discussed. The four models in the framework are spatio-temporal representation learning, scene condition understanding, feature fusion, and drowsiness detection. The spatio-temporal model captures motion and appearance within the video frames, and the scene condition model identifies the factors in the scene such as the lighting conditions, facial elements composed of head, eyes, and mouth, and whether or not glasses are being worn. Such features have been combined and implemented to render a condition-adaptive representation, thus enhancing the ability of the system to identify drowsiness in a wide range of driving conditions.
It makes use of the NTHU drowsy driver detection video dataset, which experimentally yields the outcome to leave all existing visual analysis-based drowsiness detection methods behind. The approach, through the use of condition-adaptive learning, would gain considerably in the ability to fine-tune its representation with the specific conditions of driving and to enhance more accuracy and robustness across all conditions.

[13] A Survey on State-of-the-Art Drowsiness Detection Techniques

[13] Does an all-rounded review on the approaches to detecting drowsiness in drivers, by grouping them into three: behavioural, vehicular, and physiological parameter-based approaches. The authors draw out methods existing in literature that were based on facial features such as yawning, eye closure, and head movements, as well as physiological signals like heart rate, and the behavioural information provided by vehicles to infer drowsiness. In addition, they review the top supervised machine learning techniques used for classifying drowsiness alongside a comparison of their strengths and weaknesses.
The paper outlines the pros and cons of all the detection methods and gives comparative studies for different models. That is, there is a need to integrate these multimodal techniques to improve the system with better accuracy. It concludes the discussion and presents future research directions along with suggestions to enhance the accuracy of drowsiness detection. It gives valuable suggestions for researchers working with real-time drowsiness detection systems with some relevant frameworks and diagrams.

[14] A Systematic Review on Driver Drowsiness Detection Using Eye Activity Measures.

[14] Emphasis on eye activity measurements, this systematic review investigates driver drowsiness detection (DDD) systems. The review emphasizes how important eye movements including saccadic motions, blink frequency, and eyelid closure are in identifying early indicators of tiredness, which are crucial in averting auto accidents. Using the PRISMA technique, the analysis finds 41 empirical studies that categorize ocular activity metrics into movements of the eyelid and the eyeball. It further divides the technologies utilized for eye monitoring into two categories: active (electrode-based) and passive (video and infrared-based), evaluating each one's effectiveness and intrusiveness in real-time applications.

The review also looks at decision-making algorithms that are used to categorize drowsiness. It finds that convolutional neural networks and other deep learning techniques offer higher classification accuracy (up to 99%). It does, however, highlight certain difficulties, including data heterogeneity, real versus simulated driving circumstances, and inconsistent performance measures amongst research. In order to increase accuracy and early drowsiness detection, the report highlights the necessity of consistent evaluation methodologies and urges future research to concentrate on hybrid models that integrate physiological, behavioural, and vehicle-based techniques.

[15] Intelligent Driver Drowsiness Detection for Traffic Safety Based on Multi CNN Deep Model and Facial Subsampling.

[15] Describes that it is a real-time driver drowsiness detection system based on the integration of computer vision and deep learning. It is interesting that the authors designed an ensemble deep learning model where features are extracted from the eyes of the driver and mouth by utilizing two modules of InceptionV3. These modules perform subsampling on smaller facial regions, significantly reducing computation overhead without sacrificing the detection accuracy.

The system was trained on the NTHU-DDD dataset with a training accuracy of 99.65 and validation accuracy of 98.5, and an overall evaluation accuracy of 97.1. Face detection and segmentation using MTCNN helps the model to work even in varied lighting conditions and with different driver profiles-for example, glasses. The ensemble method incorporates weighted average approaches for joining two InceptionV3 models in order to achieve stronger accuracy for drowsiness detection.

This has highlighted the strength of attention mechanisms in concentrating processing over relevant facial areas such as eyes and mouth, to produce superior results than generic whole face processing models. The result is that the proposed model offers a lightweight, real-time solution for drowsy driving detection clearly outperforming most of the existing state-of-the-art accuracy and computational efficiency.

[16] A Review of Machine Learning Techniques for Driver Drowsiness Detection Based on Behavioural Measures

[16] Describes reviews detection techniques based on machine learning, for spotting driver drowsiness, that exhibit their head movements, yawning, and blinks of the eyes. This paper tries to present the problem of developing detection systems and introduces the techniques involved in the existing approaches, such as SVM, CNN, and HMM. The meta-analysis of 25 research papers clearly pointed out that, although SVM is the most frequently used technique, CNNs proved that they outperformed other techniques in detecting drowsiness.
The paper also discusses revisiting classic algorithms in the recent light of deep learning technologies, and it gives a list of publicly available datasets that have been used for benchmarking. The review would be a valuable source to improve detection systems using machine learning.

[17] Driver Fatigue Detection Based on Eye State Analysis

[17] Offers an eye state-based analysis of a vision-based method for the detection of driver fatigue. The system is designed to be non-intrusive in character, whereby visual cues are analysed without being linked to physical sensors, thus addressing a gap in traditional methods for detecting fatigue. The system uses five phases. They are: face detection, image face segmentation using a mixed skin tone model; eye localization by a crystallization-inspired algorithm; and analysis of the eye states by using pupil height, an eye area, and the width-to-height ratio.
The research shows changes in eye states, transitions from an open to half-open or closed, have a high correlation with driver fatigue. When the driver's eyes remain closed for four consecutive frames or transition between half-open and closed for eight consecutive frames, the system announces drowsiness. A template of the driver's eyes is maintained while taking an initial detection to keep the accuracy in tracking despite lighting or head movement. These experiments were carried out upon real videos in driving by correctly detecting the probability

of drowsiness at a precision rate of 91.16% and issued warnings for the correct real drowsiness instances identified from this video.
The work addresses issues that make it difficult to distinguish very subtle eye movement, and the accuracy in detecting it at high head movements, making it a basis for further improvements in real-time driver fatigue detection systems. Further future work may be on improving the detection of eye flickering, which may give an even more accurate indicator of driver drowsiness. The system does have a great potential for road safety since its messages will alert the drivers correctly at the right time about drowsiness.

[18] Real-Time Driver Drowsiness Detection Based on Eye Movement and Yawning Using Facial Landmarks and Dlib

[18] Presents a real-time driver drowsiness detection system based on the analysis of eye movement and yawning using facial landmarks and Dlib, a machine learning-based toolkit. The study aims to mitigate the number of road accidents caused by driver drowsiness through a non-invasive and cost-effective method. The system continuously monitors the driver's eye state and yawning behavior to detect signs of fatigue, without the need for physical sensors attached to the driver's body.

The proposed system uses behavioral analysis to monitor facial landmarks such as the eyes and mouth. By detecting long periods of eye closure and frequent yawning, the system can determine when the driver is becoming drowsy. It then issues an alert to the driver, helping to prevent accidents caused by sleepiness. The key advantage of this system is its low cost and the minimal computational resources required, making it accessible for wider deployment in various vehicle types.
The authors highlight that the system's use of facial landmark detection, combined with the Dlib toolkit, allows for accurate real-time monitoring under different lighting conditions. This paper contributes to the growing field of computer vision and image processing for driver safety, offering an efficient, non-intrusive solution to a widespread problem.

2.3 PROBLEM RECOGNITION

Study of above mentioned research papers made us realise the following problems and short-comings in different models.

- **Current systems fail to integrate heart rate sensors which is one of the important cue for drowsy state.** ***[3] [7] [15]*** **discuss the advantages of physiological data for better detection.**

- **Although the discovered existing drowsiness detection models fail at low light, this further limits the proper usage of these models in real world applications like night drives. This limitation is covered in papers** ***[2] [6] [13].***

- **The current models do not consider driving speed while detecting the drowsiness, thus neglecting one of the most important cues.** ***[4] [9]*** **highlight the need to integrate real-time data, like vehicle speed, into drowsiness detection systems for better accuracy.**

- **The systems right now are isolated and fail to communicate with nearby cars to alert them and thus poses threat to overall safety, since it only alerts the driver and passengers.**

III PROPOSED SOLUTION

3.1 PROBLEM SETTING

The major challenge about driver and employee drowsiness in hazardous industries is that they easily turn into accidents and injuries, translating to lost productivity. Most of the current detection technologies for drowsiness are reactive. They would detect yawning or head-nodding after it has gripped a driver's exhaustion. The delay makes it challenging to avert such incidents before they happen since it poses serious risks in industrial settings and on the roads *[2] [6] [11].*

However, the present solutions have some limitations:

The physiological signals like heart rate are relatively underutilized.

Heart rate sensor addition with smartwatches and facial recognition can be enhancing accuracy. Heart rate variability is a much stronger signal of fatigue when combined with visual cues. The above combination provides real-time feedback so that the drivers and workers never fall into a sleep in time *[8] [13].*

Failure to function in low light conditions

Training in low light and including infrared and thermal sensors will enable the system to work more efficiently when implemented at night. As such, even when the surrounding is dimly lit, such as while driving during nighttime-the time of day most vulnerable to drowsiness-the system would be reliable.
Not monitoring speed to determine driver's behavior

In addition to this, some data of speed can also be added to provide further details about the status of a driver. The changes of fluctuation and erratic movements mostly indicate drowsiness. With such data of speed along with heart rate and facial data, the system will be able to warn of drowsiness in earlier conditions. And thereby it will be easy to avoid accidents *[9] [10] [14].*

The isolated systems which do not provide any kind of warning to the other vehicles

Technology usage of V2V, or vehicle-to-vehicle communication could advice surrounding drivers of a vehicle if it shows sign of a drowsy driver. This would prompt the action to emergency lights or distress signals to other surrounding drivers of the vehicle who would immediately respond to prevent the accident from happening.
The multi-sensory system here is integrating facial recognition, heart rate monitoring, environmental data, and speed tracking all to ensure that drivers and passengers achieve real-time feedbacks from V2V communication and are safe from accidents caused by fatigue *[1] [8] [16].*

3.2 Framework Overview

The goal of the sleepiness detection system is to improve the safety of drivers by monitoring real-time parameters such as facial feature extraction, heart rate, and vehicle speed. An integrated multi-layered architecture can be used to realise the same. The Data Acquisition Layer captures video of the driver's face through dashboard cameras, while heart rate is read through wearable devices that capture eight to ten samples within a 50 second interval. The driving speed is Retrieved from the speedometer concurrently to normalization the detection logic.

In the Perception Layer, the YOLO model enables real-time facial analysis by detecting such critical markers as eye closure and yawning. Highly developed algorithms determine eye closure that lasts longer than two seconds or someone who frequently yawns as indicators of fatigue. The Feature Extraction and Drowsiness Detection layer combines these insights with heart rate data and flags abnormalities such as heart rates below 60 bpm in conjunction with slow or erratic driving patterns. This would indicate that the driver is probably drowsy. Vehicle speed further enriches this analysis, ensuring a comprehensive approach to assessing driver alertness.

The Alert System makes use of multi-modal warnings, which consist of visual alerts on the dashboard, auditory alarms as well as external signals such as flashing lights that can call the attention of the driver or alert other vehicles in the periphery.. In the System Control Layer, the system is continuously running, analyzing video frames as well as physiological data until manually deactivated or the vehicle has become stationary.

The system exploits the state-of-the-art for real-time facial analysis with a pre-trained YOLO model, video stream with OpenCV, and Python for the detection and alert mechanisms. Valid studies support the reliability of such systems; for example, their precision in eye closure and yawning is more than 95% using YOLO-based models. When combined with heart rate monitoring, these systems have been proven to considerably succeed in preventing sleepiness-related accidents: some models report even higher reliability rates than 97.44 . The introduction of these inventions into automotive systems marks a crucial step in the control of drowsiness-related accidents.

3.3 Algorithm

The algorithm designed for drowsiness detection works by processing video input from a webcam monitoring the driver's face as well as data from the heart rate monitor and speedometer. The algorithm is structured into many steps to detect and alert drowsiness.

The video recording begins, capturing continuous frames of the driver's face. Data from the smartwatch is gathered via the Google Fit API (refer to APPENDIX-1 and APPENDIX-2), alongside speedometer readings taken at five-second intervals over 50 seconds.

Train Model

A drowsiness detection system can be defined as a real-time application used in deep learning and computer vision techniques monitoring facial expressions and detecting signs of drowsiness. Initially, it captures a live video feed through the webcam where each frame has been analyzed by the use of OpenCV's Haarcascade classifier for detecting faces within the frame. This classifier is specifically designed to be fast and efficient for face detection and is thus processed on images in grayscale for finding the appearance of faces. After a face is detected, the region of interest is then cropped out and preprocessed for further analysis by using a number of preprocessing operations. The process comprises the steps of resized images to a fixed size of 224 x 224 pixels, normalized pixel values, and conversion of the image into a deep learning model in the required format.

A system uses a tailored ResNet18 model, designed as a convolutional neural network, focused on feature extraction. The model is finetuned for classification into "wake" and "drowsy" states from facial features. It passes the cropped and preprocessed face image into the model that will output a prediction. When the model detects a drowsy face it increments a counter for counting successive frames showing drowsiness.

Such an algorithm would avoid simplistic dependency on the onset of drowsiness and would declare a person drowsy only when the onset occurs in multiple successive frames. It thus reduces false positives attributed to transient expressions or occlusions. If there are more than a predefined count of consecutive frames which show a drowsy patient, it will raise a pop-up message asking the appropriate action to be taken.

Using the real-time video processing capabilities of OpenCV and the powerful feature extraction of ResNet18, it is therefore going to be efficient and fairly accurate. Besides, this algorithm is very suitable for application in driving assistance, workplace safety, and fatigue monitoring scenarios by always monitoring facial behavior and incorporating a frame-based threshold mechanism.

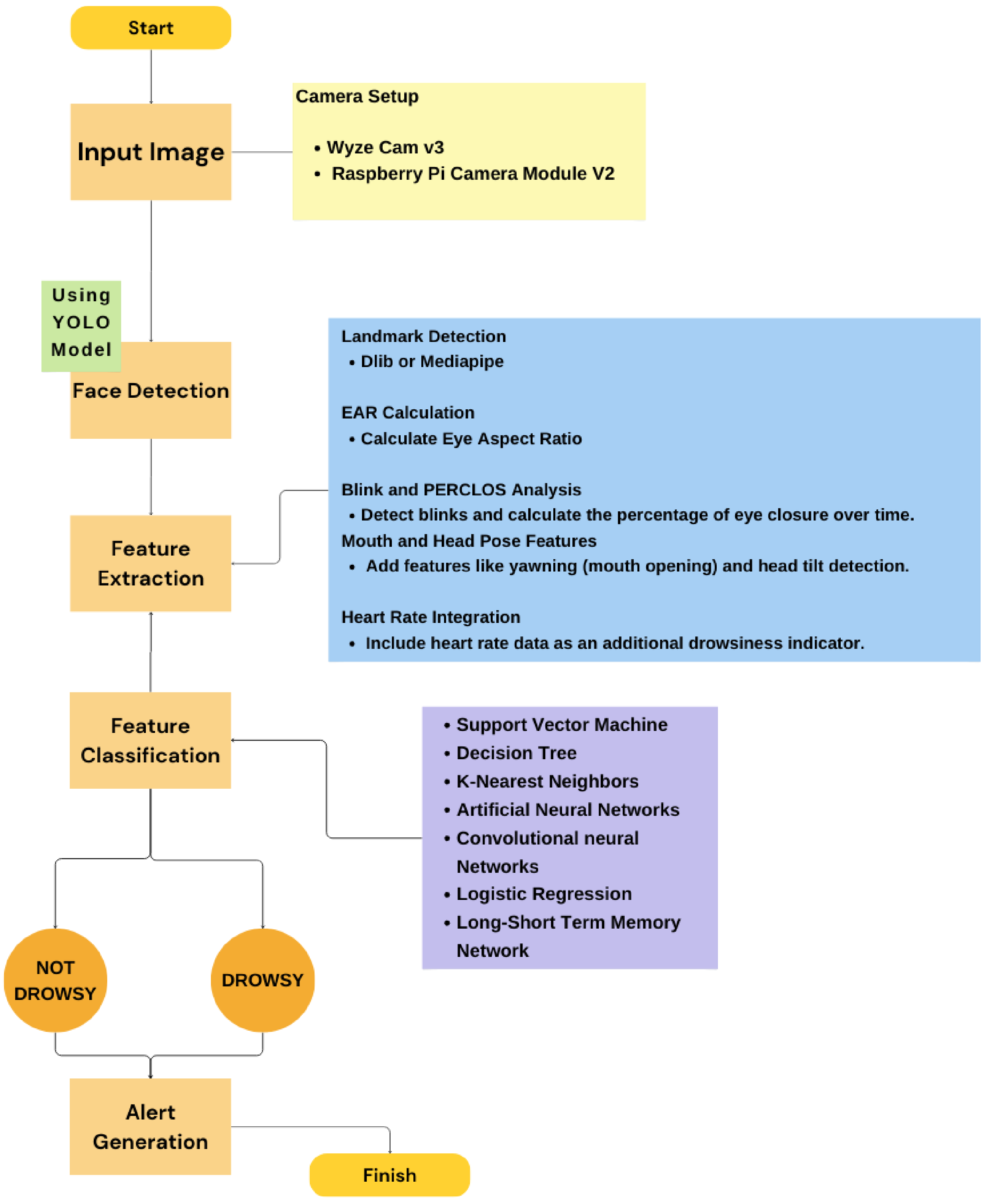
Real Time detection

The real-time drowsiness detection system begins by initializing the needful things in the beginning, such as a pre-trained ResNet18 based deep learning model fine-tuned for achieving binary classification between "awake" and "drowsy" states. Then it loads the weights of the checkpointed, pre-trained model and puts the model in an evaluation mode because it shouldn't update anything during inference. Open the webcam by feeding the live video input through VideoCapture of OpenCV; Initialize a Haarcascade classifier for fast face detection. Over each frame which arrives from the stream by opening the video, convert it into grayscale in order to detect faces, and over each identified face draw a bounding box across the video feed.

The detected face is cropped and pre-processed using a number of transformations on the picture, including making the image 224x224 pixels, normalising against the training data set and converting to a tensor that can be fed into the model. Such face data was fed into the deep learning model, which on this basis predicted whether the person was "sleeping" or "awake." If the response provided by the model indicated "sleeping," the counter increased the number of consecutive frames with drowsiness. On the other hand, the counter is decremented whenever the person is tagged as "awake." To avoid false alarms, the use of the threshold mechanism allows the system to ring the alarm only if the counter has surpassed a certain limit that signifies prolonged drowsiness.

Video feed is constantly streamed with labels ("DROWSY" or "AWAKE") over detected faces and alerts with an apt warning message if necessary. This continues in real time until the user exits by a prescribed keystroke ('q'). Finally, the system free's up the webcam and closes the window showing the display to ensure that there is a proper cleanup of resources. Thus, this algorithm integrates real-time video processing and deep learning-based face detection and classification for drowsiness monitoring robustly and responsively.

3.4 ARCHITECTURE


Start
Input Image
Camera Setup
• Wyze Cam v3
• Raspberry Pi Camera Module V2
Using YOLO Model
Face Detection
Landmark Detection
• Dlib or Mediapipe
EAR Calculation
• Calculate Eye Aspect Ratio
Blink and PERCLOS Analysis
• Detect blinks and calculate the percentage of eye closure over time.
Mouth and Head Pose Features
• Add features like yawning (mouth opening) and head tilt detection.
Heart Rate Integration
• Include heart rate data as an additional drowsiness indicator.
Feature Extraction
Feature Classification
• Support Vector Machine
• Decision Tree
• K-Nearest Neighbors
• Artificial Neural Networks
• Convolutional neural Networks
• Logistic Regression
• Long-Short Term Memory Network
NOT DROWSY
DROWSY
Alert Generation
Finish

IV

EXPERIMENTATION

(DESIGN & IMPLEMENTATION)

4.1 Dataset Description

The dataset for the drowsiness detection project plays a pivotal role in ensuring the model's ability to detect drowsiness with high accuracy in varied real-world scenarios. It combines publicly available labeled data and a custom-built dataset, carefully curated to cover diverse conditions, demographics, and facial orientations. These datasets are used to train, validate, and test a deep learning model, which employs ResNet-18 as its backbone.

Public Dataset

A significant portion of the training data is sourced from a labeled Kaggle dataset, which provides a robust foundation for the model's learning process. The dataset includes images categorized into two classes: **drowsy** and **alert** states. This publicly available dataset ensures broad coverage of variations, such as facial orientations, including direct, tilted, and turned profiles. Environmental factors like varied lighting conditions, complex backgrounds, and common occlusions, such as eyeglasses and masks, are also represented. Additionally, the dataset encompasses a diverse range of individuals, ensuring inclusivity across different ethnicities, genders, and age groups. Each image is assigned a binary label (0 for awake and 1 for drowsy), facilitating straightforward classification by the model.

Custom Dataset

To address the specific challenges of real-world applications and enhance the adaptability of the system, a custom dataset was created. This dataset includes images captured from team members and other contributors to simulate real-life scenarios more accurately. The data collection process focused on scenarios such as low-light environments, daylight glare, and varying facial features under conditions like wearing eyeglasses or masks. These additional images reflect the unique diversity of the local population, providing a significant edge in adapting the model to regional demographics. All images in the custom dataset were labeled using tools like LabelImg, ensuring precision in defining awake and drowsy states.

Diversity and Robustness

The combined dataset is designed to maximize diversity and robustness. It incorporates wide-ranging environmental factors, facial poses, and occlusions to prepare the model for practical deployment. This diversity ensures that the model generalizes effectively, even in less-controlled environments, such as those encountered during live video feeds in vehicles or workplaces.

Application to the Detection Model

The dataset undergoes pre-processing and augmentation techniques, including resizing, rotation, horizontal flipping, and normalization. These transformations enhance the model's ability to recognize patterns in varied scenarios, as implemented in the detection code. During training, the dataset is split into training and validation sets, maintaining a balance between learning and evaluation. This ensures that the model learns to distinguish drowsy from alert states reliably, as seen in both the training and inference stages.

4.2 Implementation Set Up

Hardware and System Configuration

The drowsiness detection system is implemented on a Dell G15 5520 laptop, which provides robust computational capabilities with its 16 GB RAM, 512 GB SSD, and NVIDIA RTX 3050 GPU. These specifications are ideal for handling real-time video streams and performing deep learning model inference efficiently. To capture facial features, a Jio 1080p HD external webcam is employed, delivering high-definition video input. This hardware setup ensures smooth processing of video frames and accurate detection, even under varying lighting conditions or when users wear glasses or masks.

Software Environment

The software stack for this project includes Python as the core programming language, supported by powerful libraries such as PyTorch for building and training the machine learning model, and OpenCV for real-time video processing. Development is carried out using VS Code and Jupyter Notebook, offering flexibility and interactivity during model training and testing. Annotation tools like LabelImg and YOLO Annotator are used to label the datasets, a critical step for training supervised models. GitHub is employed for version control, ensuring the project remains organized and collaborative.

Dataset Preparation

Two datasets form the foundation for training the model. The first is sourced from Kaggle, containing labeled examples of drowsy and awake states, while the second is a custom dataset created using photos of the project team and friends. The custom dataset incorporates diverse conditions, such as variations in lighting, angles, and occlusions like glasses or masks. Images are annotated using LabelImg and YOLO Annotator to identify key facial landmarks, including eyes and mouth, enabling robust training and evaluation.

Model Development

The model architecture is built using a Vision Transformer (ViT) fine-tuned for binary classification (drowsy vs. awake). The ViT leverages its powerful feature extraction capabilities, making it ideal for analyzing subtle changes in facial expressions. The classification head is replaced to adapt the model for this specific task. Training involves the use of the AdamW optimizer and a CosineAnnealingLR scheduler to dynamically adjust the learning rate. Loss is calculated using CrossEntropyLoss, ensuring accurate classification. Data augmentation techniques, such as random cropping and flipping, enhance the model's robustness to real-world scenarios.

Real-Time Processing

For real-time implementation, the system captures video streams using OpenCV and processes frames sequentially. YOLOv5 is used to detect facial landmarks, focusing on regions of interest like the eyes and mouth. These detections are fed into the Vision Transformer for classification. The system continuously monitors Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to identify blinking patterns and yawns, providing heuristic checks for drowsiness. If a drowsy state is detected consistently across multiple frames, an alert mechanism triggers, including audio alarms or visual warnings.

Performance Optimization

The model's performance is optimized through a combination of dataset variability and architectural adjustments. The Kaggle dataset and the custom dataset are alternated during training to improve generalization. Experiments with different models, such as YOLOv5 and MobileNet, help identify the best-performing architecture for deployment. The RTX 3050 GPU is leveraged to process larger batch sizes and speed up training, ensuring the system operates at an optimal frame rate for real-time detection.

nhancements

The system is enhanced with additional features to improve user experience and functionality. Integration with the Google Fit API allows monitoring of heart rate, adding a physiological dimension to drowsiness detection. A speedometer feature checks driving speed at regular intervals, useful in automotive applications. Furthermore, music playback is incorporated to keep users engaged during extended periods of inactivity. These features make the system versatile and adaptable to various scenarios beyond drowsiness detection.

Testing and Deployment

The system is rigorously tested using metrics such as accuracy, precision, recall, and F1-score, ensuring it performs reliably across diverse conditions. Confusion matrices provide insights into the classification results, highlighting areas for improvement. The model is deployed as a standalone Python application for desktop use or converted to a lightweight format using TensorFlow Lite for mobile and IoT devices. Real-world testing involves diverse environments and user behaviors, validating the system's robustness and scalability.

4.3 Training Of Model

The model training process is designed to utilize the computational power of the Dell G15 5520 laptop, featuring 16 GB RAM, a 512 GB SSD, and an NVIDIA RTX 3050 GPU. This system setup ensures efficient handling of large datasets and intensive computations required during the training phase.

The model architecture is based on a pre-trained ResNet-18, which serves as a feature extractor. The fully connected layer of ResNet-18 is modified to include two fully connected layers, a ReLU activation function, and a dropout layer for regularization. This allows the model to specialize in binary classification, distinguishing between drowsy and alert states.

Data preprocessing plays a crucial role in training. Input images are resized to 224x224 dimensions and undergo transformations such as random horizontal flips and rotations for augmentation. This ensures the model learns from diverse conditions, improving generalization. The data is normalized using mean and standard deviation values specific to ImageNet, the dataset on which ResNet-18 was initially trained.

The training dataset is split into an 80-20 ratio, creating separate training and validation sets. Data is loaded in mini-batches using the DataLoader, enabling efficient processing and shuffling of data during training. A batch size of 32 is used to balance memory consumption and computational efficiency.

The loss function employed is CrossEntropyLoss, which is suitable for multi-class classification tasks like this one. The optimizer used is Adam, with an initial learning rate of 0.001, enabling fast convergence. During each epoch, the model undergoes a training phase followed by a validation phase.

In the training phase, the model learns by forward propagating input images, calculating loss, and backpropagating gradients to update weights. Validation is performed after each epoch to evaluate the model's generalization on unseen data. Metrics such as training and validation loss, along with accuracies, are printed for each epoch to track progress.

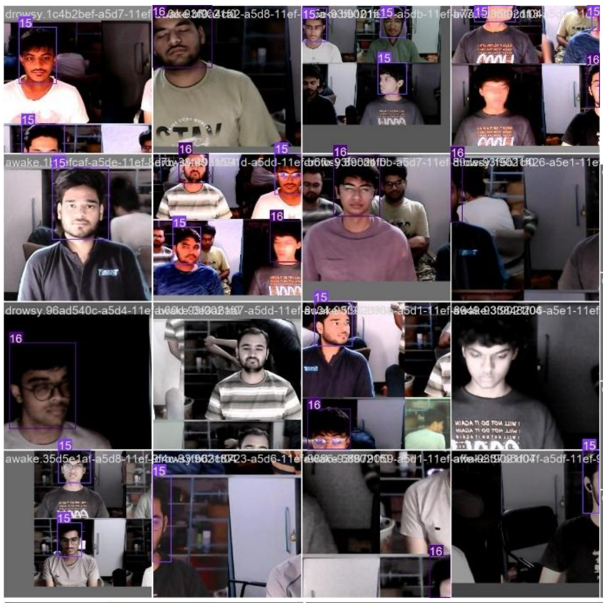
The GPU accelerates this entire process, allowing complex computations for ResNet-18 to be executed efficiently. After 10 epochs, the model's performance stabilizes, and the best version is saved as a PyTorch model file (drowsiness_detection_model.pth). This ensures the trained model can be used later for inference or further fine-tuning.

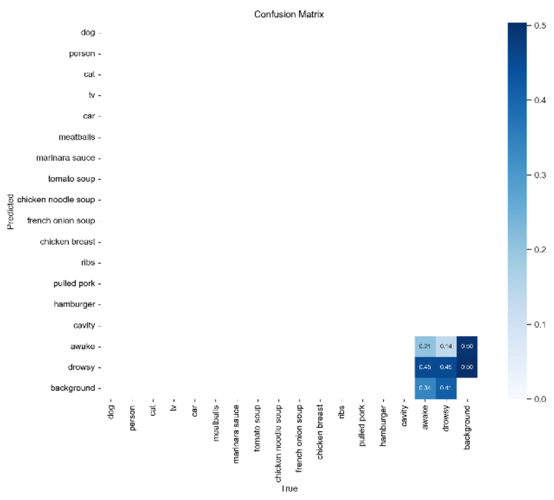
The entire process ensures a robust and well-trained drowsiness detection model, ready for real-world testing and deployment.

4.4 Experiments and analysis

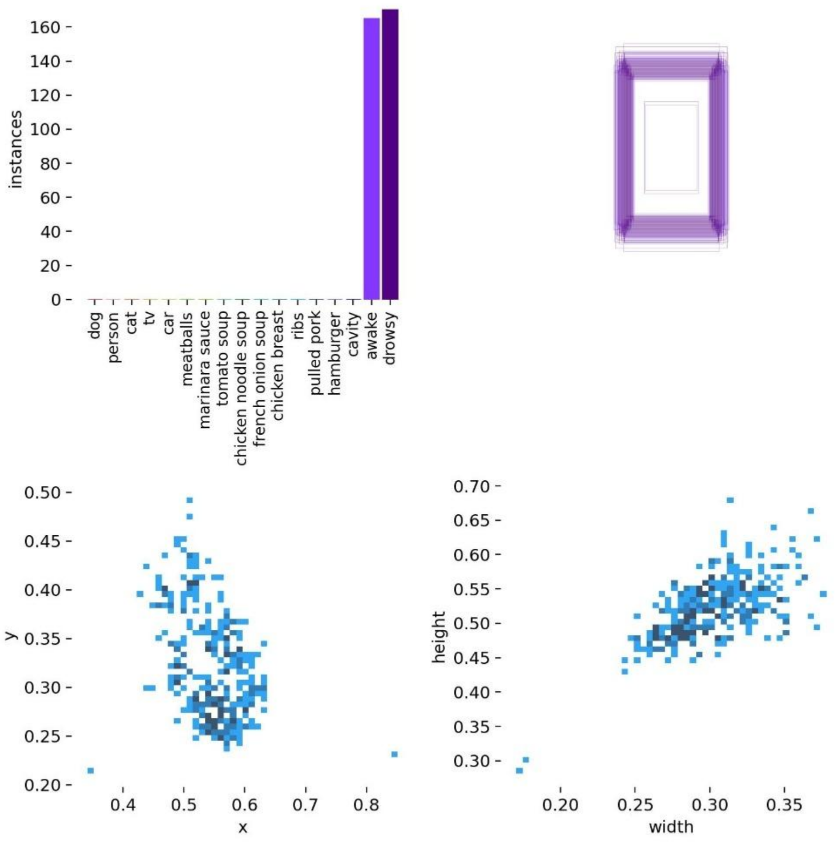
4.4.1 Experiment 1

In this experiment yolo model was used. Also the dataset used to train the model was the custom datatset which had about 500 images. This model did not give the expected results. It had accuravy of only about 70 percent. So we changed the model, code and used different libraries which provided better results as discussed in the next experiment.

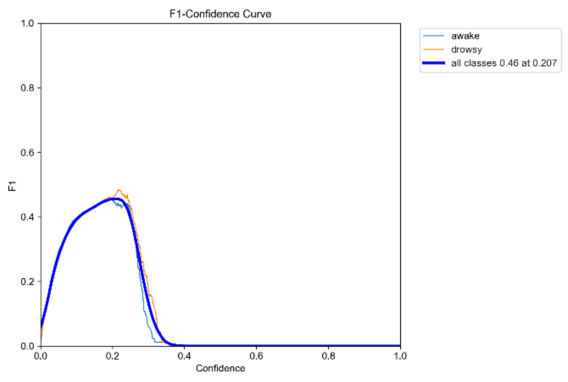




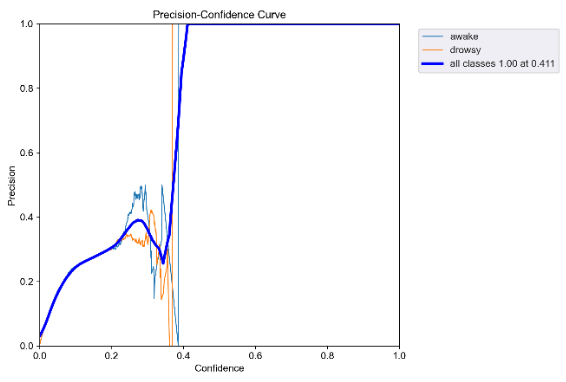
Confusion Matrix
Predicted
True
dog
person
cat
tv
car
meatballs
marinara sauce
tomato soup
chicken noodle soup
french onion soup
chicken breast
ribs
pulled pork
hamburger
cavity
awake
drowsy
background
0.21
0.14
0.00
0.45
0.45
0.00
0.34
0.41
0.5
0.4
0.3
0.2
0.1
0.0



instances
dog
person
cat
tv
car
meatballs
marinara sauce
tomato soup
chicken noodle soup
french onion soup
chicken breast
ribs
pulled pork
hamburger
cavity
awake
drowsy
x
y
width
height



F1-Confidence Curve
F1
Confidence
awake
drowsy
all classes 0.46 at 0.207

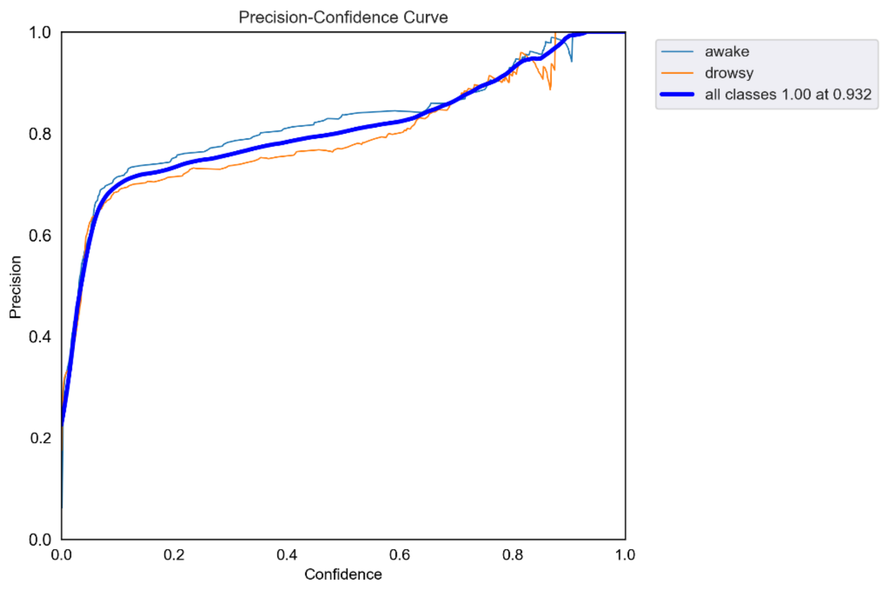


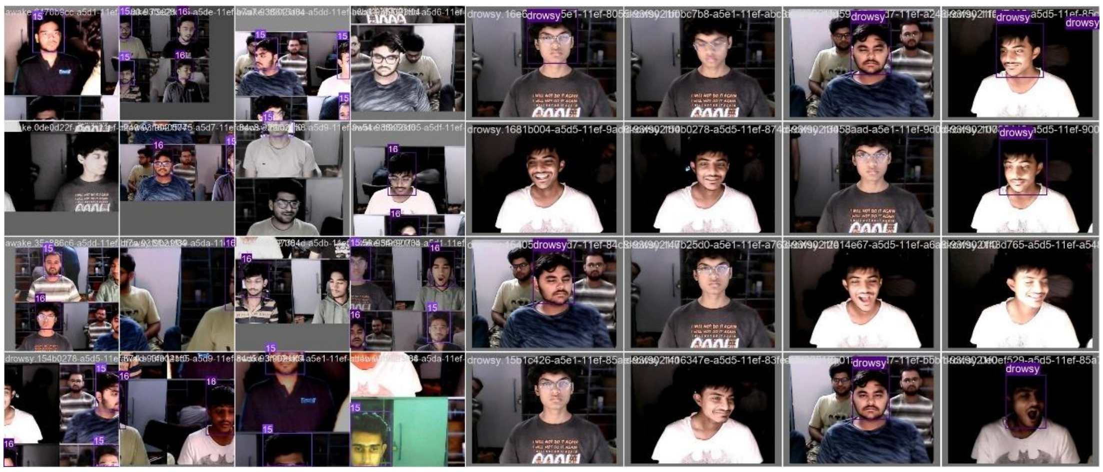
Precision-Confidence Curve
Precision
Confidence
awake
drowsy
all classes 1.00 at 0.411

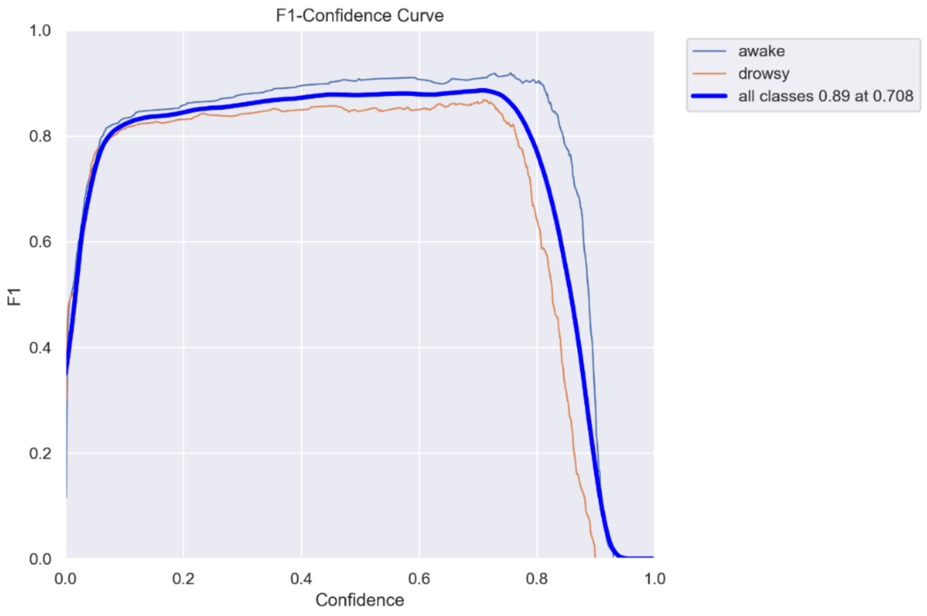
4.4.2 Experiment 2

This Experiment was simlar to that of experiment 1 but the model was changed. Also we changed the machine from Nvidia GTX 1650 to Nvidia RTX 3050. The updated code was:

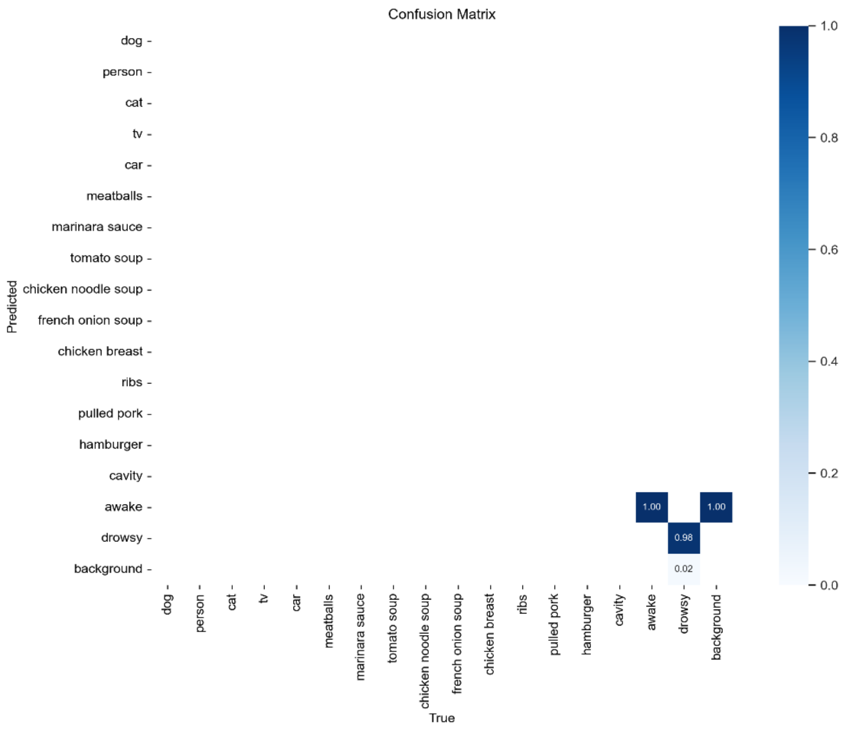
```
import torch
from matplotlib import pyplot as plt
import numpy as np
import cv2
import uuid #unique identifier
import os
import time
cap=cv2.VideoCapture(0)
for label in labels:
    print('Collecting images for'.format(labels))
    time.sleep(5)
    for img_num in range(number_imgs):
        print('Collecting images for {}, image
number{}'.format(label,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)
        if cv2.waitKey(10) & 0xFF==ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
 #THEN A DATASET.YML FILE WAS CREATED AND THE MODEL WAS PUT TO TRAINED
!cd "C:/Users/Samarth Khandelwal/yolov5"  && python train.py --img 7000 --
batch 70 --epochs 5 --data dataset.yml --weights yolov5s.pt --workers 2
```







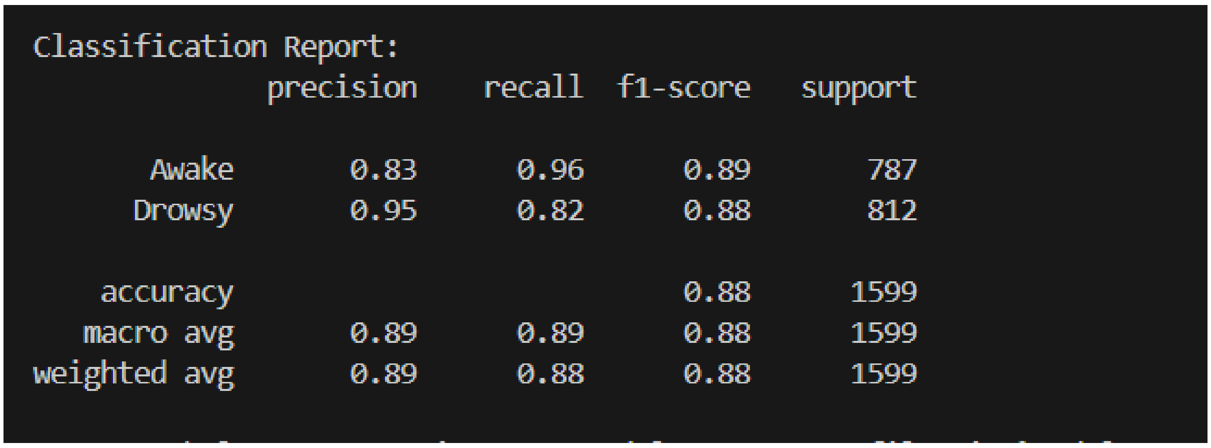
F1-Confidence Curve
F1
Confidence
awake
drowsy
all classes 0.89 at 0.708



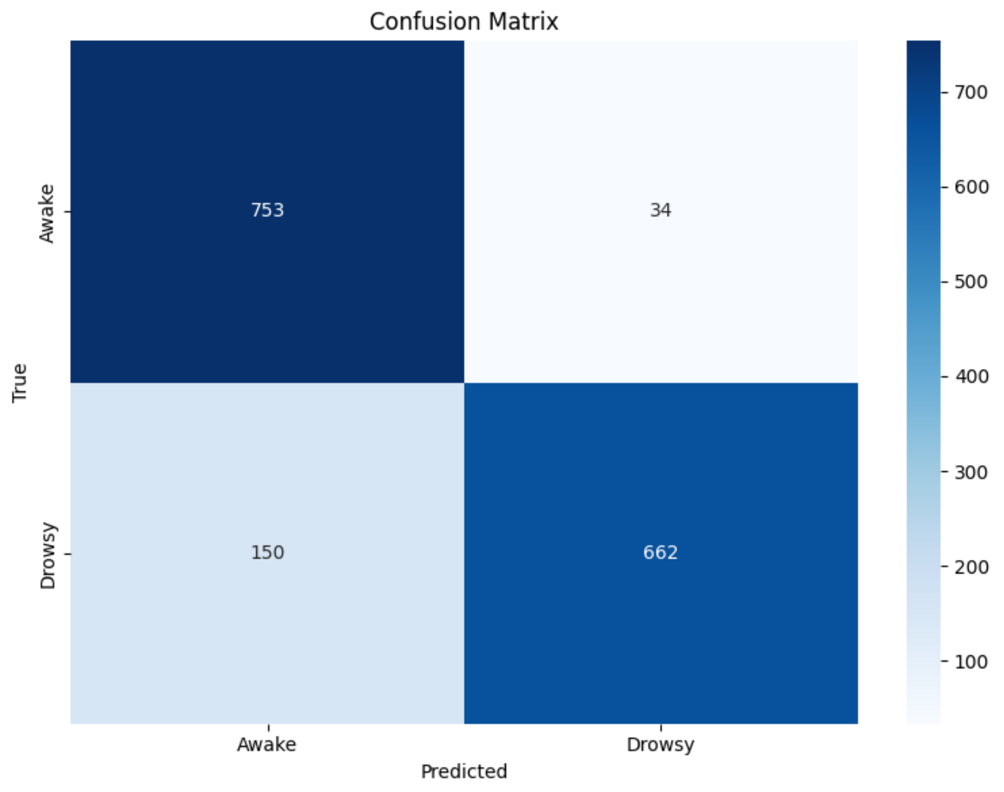
Confusion Matrix
Predicted
True
dog
person
cat
tv
car
meatballs
marinara sauce
tomato soup
chicken noodle soup
french onion soup
chicken breast
ribs
pulled pork
hamburger
cavity
awake
drowsy
background
1.00
0.98
0.02

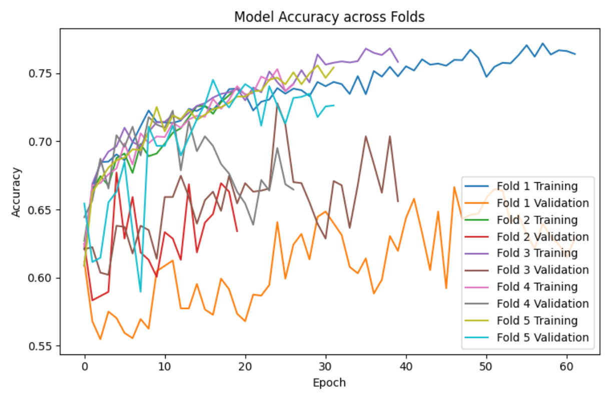
4.4.3 Experiment 3

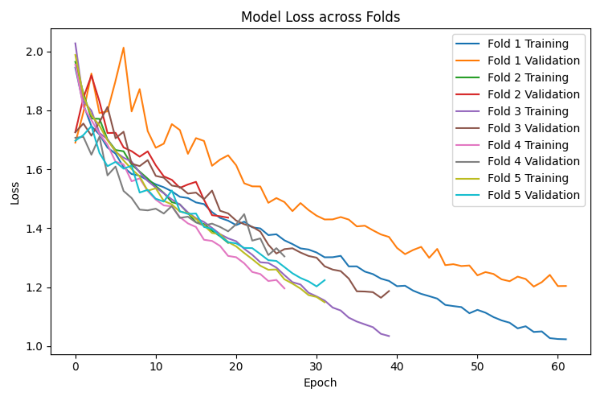
In this experiment we shifted our model an yolo to tenserflow which enhanced the accuracy and we achieved accuracy of about 88%.

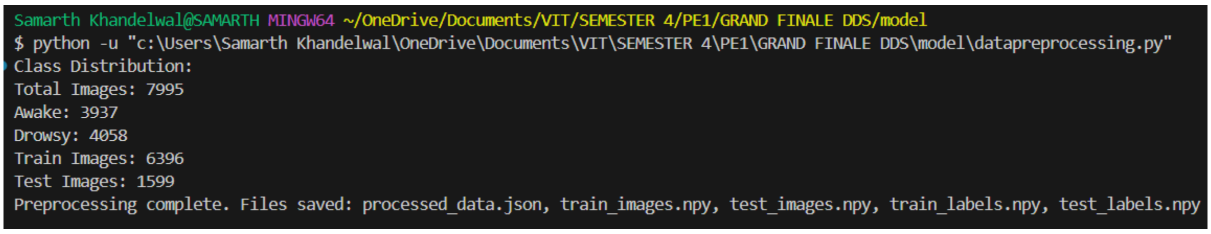


```
Classification Report:
              precision    recall  f1-score   support

       Awake       0.83      0.96      0.89       787
      Drowsy       0.95      0.82      0.88       812

    accuracy                           0.88      1599
   macro avg       0.89      0.89      0.88      1599
weighted avg       0.89      0.88      0.88      1599
```

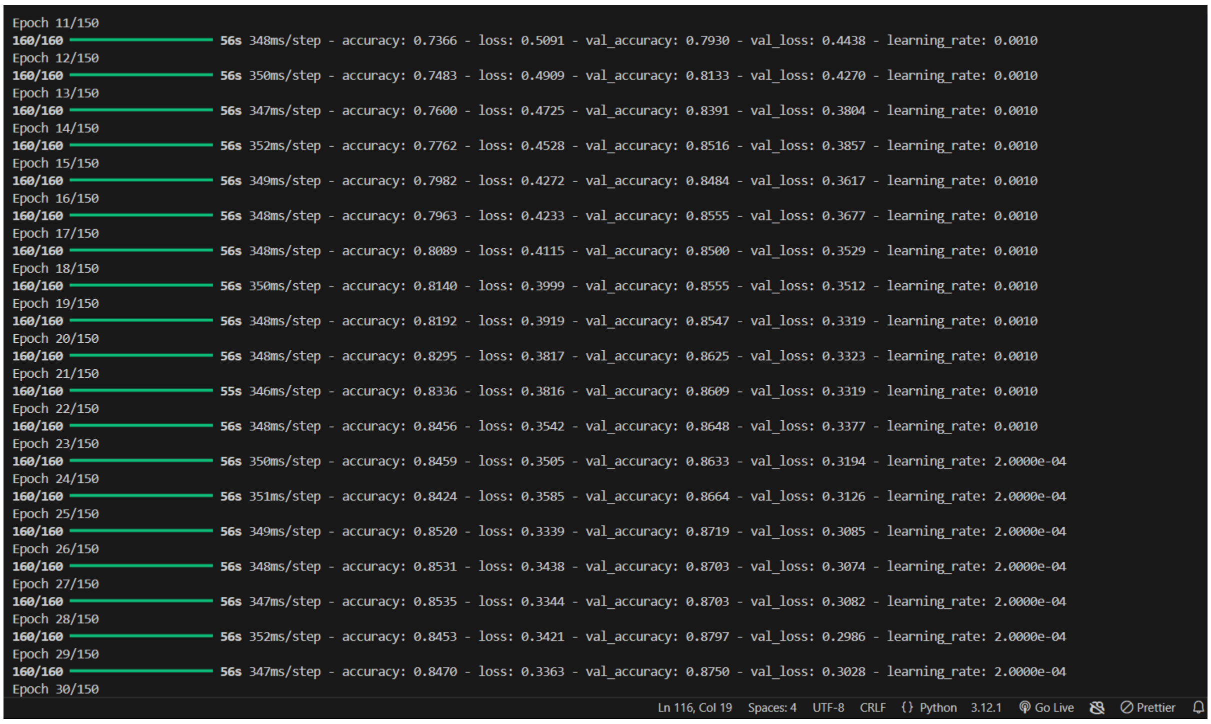








```
Samarth Khandelwal@SAMARTH MINGW64 ~/OneDrive/Documents/VIT/SEMESTER 4/PE1/GRAND FINALE DDS/model
$ python -u "c:\Users\Samarth Khandelwal\OneDrive\Documents\VIT\SEMESTER 4\PE1\GRAND FINALE DDS\model\datapreprocessing.py"
Class Distribution:
Total Images: 7995
Awake: 3937
Drowsy: 4058
Train Images: 6396
Test Images: 1599
Preprocessing complete. Files saved: processed_data.json, train_images.npy, test_images.npy, train_labels.npy, test_labels.npy
```



```
Epoch 11/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.7366 - loss: 0.5091 - val_accuracy: 0.7930 - val_loss: 0.4438 - learning_rate: 0.0010
Epoch 12/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 350ms/step - accuracy: 0.7483 - loss: 0.4909 - val_accuracy: 0.8133 - val_loss: 0.4270 - learning_rate: 0.0010
Epoch 13/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 347ms/step - accuracy: 0.7600 - loss: 0.4725 - val_accuracy: 0.8391 - val_loss: 0.3804 - learning_rate: 0.0010
Epoch 14/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 352ms/step - accuracy: 0.7762 - loss: 0.4528 - val_accuracy: 0.8516 - val_loss: 0.3857 - learning_rate: 0.0010
Epoch 15/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 349ms/step - accuracy: 0.7982 - loss: 0.4272 - val_accuracy: 0.8484 - val_loss: 0.3617 - learning_rate: 0.0010
Epoch 16/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.7963 - loss: 0.4233 - val_accuracy: 0.8555 - val_loss: 0.3677 - learning_rate: 0.0010
Epoch 17/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.8089 - loss: 0.4115 - val_accuracy: 0.8500 - val_loss: 0.3529 - learning_rate: 0.0010
Epoch 18/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 350ms/step - accuracy: 0.8140 - loss: 0.3999 - val_accuracy: 0.8555 - val_loss: 0.3512 - learning_rate: 0.0010
Epoch 19/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.8192 - loss: 0.3919 - val_accuracy: 0.8547 - val_loss: 0.3319 - learning_rate: 0.0010
Epoch 20/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.8295 - loss: 0.3817 - val_accuracy: 0.8625 - val_loss: 0.3323 - learning_rate: 0.0010
Epoch 21/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 55s 346ms/step - accuracy: 0.8336 - loss: 0.3816 - val_accuracy: 0.8609 - val_loss: 0.3319 - learning_rate: 0.0010
Epoch 22/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.8456 - loss: 0.3542 - val_accuracy: 0.8648 - val_loss: 0.3377 - learning_rate: 0.0010
Epoch 23/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 350ms/step - accuracy: 0.8459 - loss: 0.3505 - val_accuracy: 0.8633 - val_loss: 0.3194 - learning_rate: 2.0000e-04
Epoch 24/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 351ms/step - accuracy: 0.8424 - loss: 0.3585 - val_accuracy: 0.8664 - val_loss: 0.3126 - learning_rate: 2.0000e-04
Epoch 25/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 349ms/step - accuracy: 0.8520 - loss: 0.3339 - val_accuracy: 0.8719 - val_loss: 0.3085 - learning_rate: 2.0000e-04
Epoch 26/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 348ms/step - accuracy: 0.8531 - loss: 0.3438 - val_accuracy: 0.8703 - val_loss: 0.3074 - learning_rate: 2.0000e-04
Epoch 27/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 347ms/step - accuracy: 0.8535 - loss: 0.3344 - val_accuracy: 0.8703 - val_loss: 0.3082 - learning_rate: 2.0000e-04
Epoch 28/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 352ms/step - accuracy: 0.8453 - loss: 0.3421 - val_accuracy: 0.8797 - val_loss: 0.2986 - learning_rate: 2.0000e-04
Epoch 29/150
160/160 ━━━━━━━━━━━━━━━━━━━━ 56s 347ms/step - accuracy: 0.8470 - loss: 0.3363 - val_accuracy: 0.8750 - val_loss: 0.3028 - learning_rate: 2.0000e-04
Epoch 30/150
```

CODE:

```
DATA PRE PROCESSING
import os
import json
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split

# Define paths
images_folder = r"C:\Users\Samarth Khandelwal\OneDrive\Documents\VIT\SEMESTER
4\PE1\GRAND FINALE DDS\model\test\images"
labels_folder = r"C:\Users\Samarth Khandelwal\OneDrive\Documents\VIT\SEMESTER
4\PE1\GRAND FINALE DDS\model\test\labels"

def load_labels_and_images(images_folder, labels_folder):
    """
    Load images and their corresponding labels from separate folders.

    Args:
        images_folder (str): Path to the folder containing image files
        labels_folder (str): Path to the folder containing label files

    Returns:
        tuple: Lists of image paths and corresponding labels
    """
    image_paths = []
    image_labels = []

    # Iterate through image files
    for image_file in os.listdir(images_folder):
        if image_file.endswith((".jpg", ".png", ".jpeg")):
            # Construct corresponding label file path
            label_file = os.path.join(labels_folder,
image_file.replace('.jpg', '.txt').replace('.png', '.txt').replace('.jpeg',
'.txt'))

            # Check if label file exists
            if os.path.exists(label_file):
                # Read label file
                with open(label_file, 'r') as f:
                    label_content = f.read().strip().split()

                # First digit determines the class (0 = awake, 1 = drowsy)
                if label_content:
                    label = int(label_content[0])

                    # Add to lists
                    image_paths.append(os.path.join(images_folder,
image_file))
                    image_labels.append(label)

    return image_paths, image_labels

# Load images and labels
image_paths, image_labels = load_labels_and_images(images_folder,
labels_folder)

# Split dataset into train and test sets
train_images, test_images, train_labels, test_labels = train_test_split(
    image_paths, image_labels, test_size=0.2, random_state=42,
stratify=image_labels
)

# Optional: Print class distribution
class_distribution = {
    "Total Images": len(image_paths),
    "Awake": image_labels.count(0),
    "Drowsy": image_labels.count(1),
    "Train Images": len(train_images),
    "Test Images": len(test_images)
}
print("Class Distribution:")
```

```python
for key, value in class_distribution.items():
    print(f"{key}: {value}")

# Save the processed paths and labels for later use
data = {
    "train_images": train_images,
    "train_labels": train_labels,
    "test_images": test_images,
    "test_labels": test_labels
}

# Save as JSON file
with open('processed_data.json', 'w') as f:
    json.dump(data, f, indent=4)

# Optional: Prepare images for model training
def prepare_images(image_paths, target_size=(224, 224)):
    """
    Load and preprocess images.

    Args:
        image_paths (list): List of paths to image files
        target_size (tuple): Desired image size for model input

    Returns:
        numpy.ndarray: Preprocessed image array
    """
    images = []
    for path in image_paths:
        # Open image
        img = Image.open(path)

        # Resize image
        img = img.resize(target_size)

        # Convert to numpy array and normalize
        img_array = np.array(img) / 255.0

        images.append(img_array)

    return np.array(images)

# Optional: Prepare training and testing image arrays
train_image_array = prepare_images(train_images)
test_image_array = prepare_images(test_images)

# Save preprocessed image arrays (optional)
np.save('train_images.npy', train_image_array)
np.save('test_images.npy', test_image_array)
np.save('train_labels.npy', np.array(train_labels))
np.save('test_labels.npy', np.array(test_labels))

print("Preprocessing complete. Files saved: processed_data.json,
train_images.npy, test_images.npy, train_labels.npy, test_labels.npy")
```

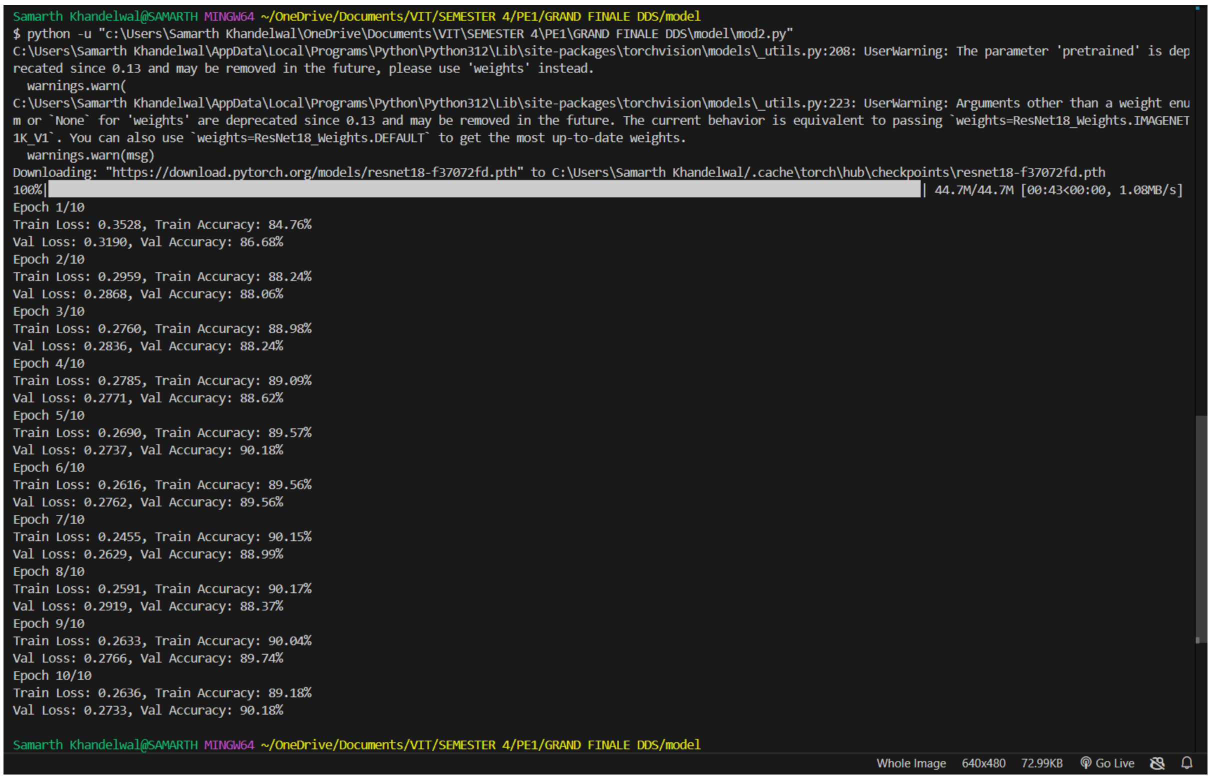
DATASET LOADER

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,
ModelCheckpoint
from tensorflow.keras.applications import MobileNetV2
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold

# Set random seeds for reproducibility
tf.random.set_seed(42)
np.random.seed(42)

# Advanced Data Augmentation
def create_advanced_data_augmentation():
    return tf.keras.Sequential([
        tf.keras.layers.RandomFlip("horizontal_and_vertical"),
        tf.keras.layers.RandomRotation(0.2),
        tf.keras.layers.RandomZoom(0.2),
        tf.keras.layers.RandomContrast(0.2),
        tf.keras.layers.GaussianNoise(0.1)
    ])

# Enhanced CNN Model with Transfer Learning
def build_enhanced_model(input_shape=(224, 224, 3), num_classes=2):
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )
    base_model.trainable = False

    model = Sequential([
        create_advanced_data_augmentation(),
        base_model,
        tf.keras.layers.GlobalAveragePooling2D(),
        Dense(512, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
        BatchNormalization(),
        Dropout(0.5),
        Dense(256, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
        BatchNormalization(),
        Dropout(0.4),
        Dense(num_classes, activation='softmax')
    ])

    # Use a fixed learning rate instead of a schedule
    model.compile(
        optimizer=Adam(learning_rate=0.0001),  # Fixed learning rate
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

# K-Fold Cross Validation Training
def train_with_cross_validation(X, y, num_folds=5):
    skf = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)

    fold_scores = []
    histories = []

    for fold, (train_index, val_index) in enumerate(skf.split(X, y), 1):
        print(f"\nTraining Fold {fold}")

        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        # Reset model for each fold
        model = build_enhanced_model()

        # Checkpoint to save best model
        checkpoint = ModelCheckpoint(
            f'best_model_fold_{fold}.keras',
            save_best_only=True,
            monitor='val_accuracy'
        )

        # Learning Rate Reduction
        reduce_lr = ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=5,
            min_lr=0.000001
        )

        # Early Stopping
        early_stopping = EarlyStopping(
            monitor='val_accuracy',
            patience=15,
            restore_best_weights=True
        )

        # Train Model
        history = model.fit(
            X_train,
            y_train,
            validation_data=(X_val, y_val),
            epochs=300,
            batch_size=32,
            callbacks=[reduce_lr, early_stopping, checkpoint]
        )

        # Evaluate Model
        val_loss, val_accuracy = model.evaluate(X_val, y_val)
        fold_scores.append(val_accuracy)
        histories.append(history)

        print(f"Fold {fold} Validation Accuracy: {val_accuracy * 100:.2f}%")

    # Print Average Performance
    print(f"\nAverage Cross-Validation Accuracy: {np.mean(fold_scores) *
100:.2f}%")

    return histories

# Visualization Functions
def plot_cross_validation_results(histories):
    plt.figure(figsize=(15, 5))

    # Accuracy Plots
    plt.subplot(1, 2, 1)
    for i, history in enumerate(histories, 1):
        plt.plot(history.history['accuracy'], label=f'Fold {i} Training')
        plt.plot(history.history['val_accuracy'], label=f'Fold {i}
Validation')
    plt.title('Model Accuracy across Folds')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss Plots
    plt.subplot(1, 2, 2)
    for i, history in enumerate(histories, 1):
        plt.plot(history.history['loss'], label=f'Fold {i} Training')
        plt.plot(history.history['val_loss'], label=f'Fold {i} Validation')
    plt.title('Model Loss across Folds')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.savefig('cross_validation_results.png')
    plt.close()

def main():
    # Load preprocessed data
    train_images = np.load('train_images.npy')
    train_labels = np.load('train_labels.npy')

    # Perform Cross-Validation Training
    histories = train_with_cross_validation(train_images, train_labels)

    # Visualize Cross-Validation Results
    plot_cross_validation_results(histories)

if __name__ == "__main__":
    main()

REAL TIME

import cv2
import numpy as np
from tensorflow.keras.models import load_model

# Load the pre-trained model
model = load_model('drowsiness_detection_model.h5')

# Webcam setup
cap = cv2.VideoCapture(0)  # 0 for the default webcam
input_size = (224, 224)  # Assuming your model expects 224x224 input

while True:
    # Capture frame from webcam
    ret, frame = cap.read()

    # Pre-process the frame
    frame_resized = cv2.resize(frame, input_size)
    frame_normalized = frame_resized / 255.0
    frame_expanded = np.expand_dims(frame_normalized, axis=0)

    # Make prediction
    prediction = model.predict(frame_expanded)
    drowsy_probability = prediction[0, 1]

    # Determine drowsiness state
    if drowsy_probability > 0.5:
        drowsiness_state = "Drowsy"
    else:
        drowsiness_state = "Awake"

    # Display the results
    cv2.putText(frame, drowsiness_state, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 0, 255), 2)
    cv2.imshow('Drowsiness Detection', frame)

    # Press 'q' to exit the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()
```

4.4.4 Experiment 4

This is our final model which gave accuracy of 90%+. This will be further trained with more ephocs and more comprehnesive dataset. Currently only 10 epochs were done and the dataset was in well lit environment.

The Drowsiness Detection Model employs several libraries and techniques to perform real-time drowsiness detection from facial images. The core library used is PyTorch, which provides the necessary tools to build and train the neural network. The model leverages ResNet-18, a pre-trained deep learning architecture known for its ability to effectively learn image features using Convolutional Neural Networks (CNNs). ResNet-18 is modified by replacing the final fully connected layer to adapt it to a binary classification task: awake (0) and drowsy (1). To enhance the accuracy of the model, cross-entropy loss is utilized as the loss function, which is suitable for classification problems. For real-time face detection, OpenCV is used with Haar Cascade Classifiers to identify faces in video streams. Images are processed using Torchvision for transformations like resizing and normalization, preparing them for the model. Additionally, PIL is used to handle image input and output. The model's ability to track drowsiness is supported by analyzing consecutive frames, where prolonged drowsiness is flagged, alerting the user. This combination of deep learning, image processing, and real-time video analysis enables the system to effectively detect drowsiness.



```
Samarth Khandelwal@SAMARTH MINGW64 ~/OneDrive/Documents/VIT/SEMESTER 4/PE1/GRAND FINALE DDS/model
$ python -u "c:\Users\Samarth Khandelwal\OneDrive\Documents\VIT\SEMESTER 4\PE1\GRAND FINALE DDS\model\mod2.py"
C:\Users\Samarth Khandelwal\AppData\Local\Programs\Python\Python312\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is dep
recated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\Samarth Khandelwal\AppData\Local\Programs\Python\Python312\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enu
m or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=ResNet18_Weights.IMAGENET
1K_V1`. You can also use `weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to C:\Users\Samarth Khandelwal/.cache\torch\hub\checkpoints\resnet18-f37072fd.pth
100%|████████████████████████████████████████████████████████████████████████████| 44.7M/44.7M [00:43<00:00, 1.08MB/s]
Epoch 1/10
Train Loss: 0.3528, Train Accuracy: 84.76%
Val Loss: 0.3190, Val Accuracy: 86.68%
Epoch 2/10
Train Loss: 0.2959, Train Accuracy: 88.24%
Val Loss: 0.2868, Val Accuracy: 88.06%
Epoch 3/10
Train Loss: 0.2760, Train Accuracy: 88.98%
Val Loss: 0.2836, Val Accuracy: 88.24%
Epoch 4/10
Train Loss: 0.2785, Train Accuracy: 89.09%
Val Loss: 0.2771, Val Accuracy: 88.62%
Epoch 5/10
Train Loss: 0.2690, Train Accuracy: 89.57%
Val Loss: 0.2737, Val Accuracy: 90.18%
Epoch 6/10
Train Loss: 0.2616, Train Accuracy: 89.56%
Val Loss: 0.2762, Val Accuracy: 89.56%
Epoch 7/10
Train Loss: 0.2455, Train Accuracy: 90.15%
Val Loss: 0.2629, Val Accuracy: 88.99%
Epoch 8/10
Train Loss: 0.2591, Train Accuracy: 90.17%
Val Loss: 0.2919, Val Accuracy: 88.37%
Epoch 9/10
Train Loss: 0.2633, Train Accuracy: 90.04%
Val Loss: 0.2766, Val Accuracy: 89.74%
Epoch 10/10
Train Loss: 0.2636, Train Accuracy: 89.18%
Val Loss: 0.2733, Val Accuracy: 90.18%

Samarth Khandelwal@SAMARTH MINGW64 ~/OneDrive/Documents/VIT/SEMESTER 4/PE1/GRAND FINALE DDS/model
```

TRAINING

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, models
from PIL import Image
```

```
class DrowsinessDataset(Dataset):
    def __init__(self, image_dir, label_dir, transform=None):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.transform = transform
        self.images = []
        self.labels = []

        # Process images and labels
        for img_name in os.listdir(image_dir):
            if img_name.endswith(('.jpg', '.png', '.jpeg')):
                # Get corresponding label file
                label_path = os.path.join(label_dir, img_name.rsplit('.',
1)[0] + '.txt')

                if os.path.exists(label_path):
                    with open(label_path, 'r') as f:
                        label_line = f.readline().strip().split()
                        if label_line:
                            # First digit indicates drowsiness state
                            label = int(label_line[0])
                            self.images.append(os.path.join(image_dir,
img_name))
                            self.labels.append(label)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path = self.images[idx]
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, self.labels[idx]

class DrowsinessDetectionModel(nn.Module):
    def __init__(self, num_classes=2):
        super().__init__()
        # Use pre-trained ResNet as feature extractor
        self.resnet = models.resnet18(pretrained=True)

        # Modify last fully connected layer
        num_features = self.resnet.fc.in_features
        self.resnet.fc = nn.Sequential(
            nn.Linear(num_features, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
```

```python
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        return self.resnet(x)

def train_model(model, train_loader, val_loader, criterion, optimizer,
num_epochs=10):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    for epoch in range(num_epochs):
        model.train()
        train_loss, correct_train = 0.0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            correct_train += (predicted == labels).sum().item()

        # Validation
        model.eval()
        val_loss, correct_val = 0.0, 0

        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                correct_val += (predicted == labels).sum().item()

        print(f"Epoch {epoch+1}/{num_epochs}")
        print(f"Train Loss: {train_loss/len(train_loader):.4f}, Train
Accuracy: {100 * correct_train/len(train_loader.dataset):.2f}%")
        print(f"Val Loss: {val_loss/len(val_loader):.4f}, Val Accuracy: {100 *
correct_val/len(val_loader.dataset):.2f}%")

def main():
    # Data transformations
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
    ])

    # Paths for images and labels
    image_dir = r'C:\Users\Samarth Khandelwal\OneDrive\Documents\VIT\SEMESTER
4\PE1\GRAND FINALE DDS\model\test\images'
    label_dir = r'C:\Users\Samarth Khandelwal\OneDrive\Documents\VIT\SEMESTER
4\PE1\GRAND FINALE DDS\model\test\labels'

    # Create datasets
    full_dataset = DrowsinessDataset(image_dir, label_dir,
transform=transform)

    # Split dataset
    train_size = int(0.8 * len(full_dataset))
    val_size = len(full_dataset) - train_size
    train_dataset, val_dataset = torch.utils.data.random_split(full_dataset,
[train_size, val_size])

    # Create data loaders
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

    # Initialize model
    model = DrowsinessDetectionModel()

    # Loss and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Train the model
    train_model(model, train_loader, val_loader, criterion, optimizer)

    # Save the model
    torch.save(model.state_dict(), 'drowsiness_detection_model.pth')

if __name__ == '__main__':
    main()
```

Real Time Detection

```python
import torch
import torch.nn as nn
import torchvision.transforms as transforms
import cv2
import numpy as np
from torchvision import models

class DrowsinessDetectionModel(nn.Module):
    def __init__(self, num_classes=2):
        super().__init__()
        self.resnet = models.resnet18(pretrained=False)
        num_features = self.resnet.fc.in_features
        self.resnet.fc = nn.Sequential(
            nn.Linear(num_features, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        return self.resnet(x)

def detect_drowsiness():
    # Load pre-trained model
    model = DrowsinessDetectionModel()
    model.load_state_dict(torch.load('drowsiness_detection_model.pth'))
    model.eval()

    # Transformations
    transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
    ])

    # Face detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    # Video capture
    cap = cv2.VideoCapture(0)

    # Drowsiness tracking
    drowsy_frames = 0
    max_drowsy_frames = 30  # Adjust based on desired sensitivity

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Convert to grayscale for face detection
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces
        faces = face_cascade.detectMultiScale(gray, 1.1, 4)

        for (x, y, w, h) in faces:
            # Draw rectangle around face
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

            # Crop face
            face_img = frame[y:y+h, x:x+w]

            # Prepare for model
            face_tensor = transform(face_img)
            face_tensor = face_tensor.unsqueeze(0)  # Add batch dimension

            # Predict
            with torch.no_grad():
                outputs = model(face_tensor)
                _, predicted = torch.max(outputs.data, 1)

                # 0 = awake, 1 = drowsy
                if predicted.item() == 1:
                    drowsy_frames += 1
                    cv2.putText(frame, 'DROWSY', (x, y-10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
                else:
                    drowsy_frames = max(0, drowsy_frames - 1)
                    cv2.putText(frame, 'AWAKE', (x, y-10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

        # Alert for prolonged drowsiness
        if drowsy_frames > max_drowsy_frames:
            cv2.putText(frame, 'DROWSINESS ALERT!', (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 255), 4)

        # Display
        cv2.imshow('Drowsiness Detection', frame)

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

    # Cleanup
    cap.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    detect_drowsiness()
```

V RESULT AND DISCUSSION

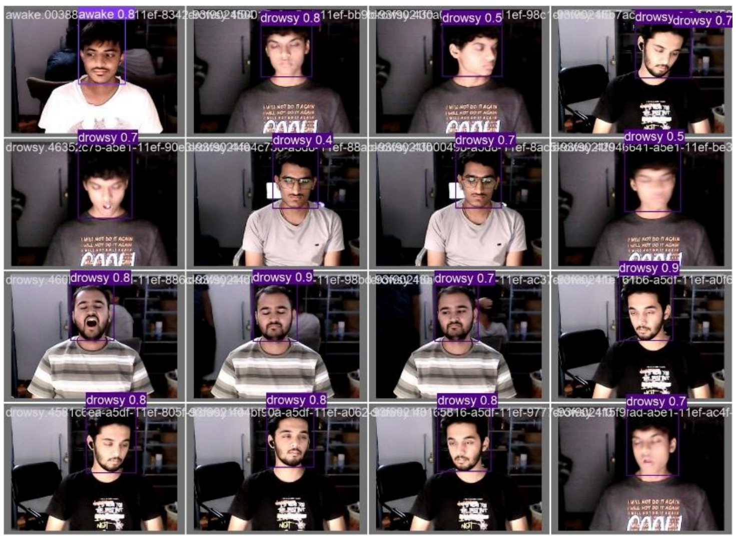
5.1 Dataset 1 : Experiment 1 & 2

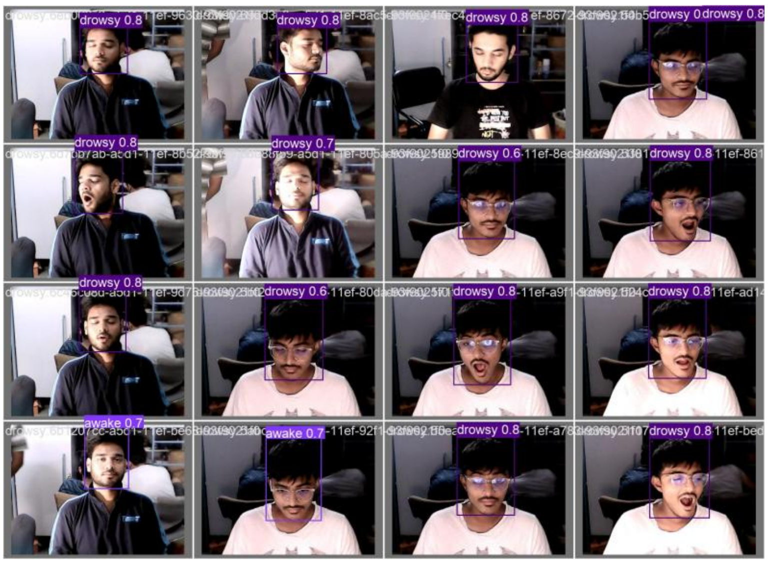
The curve plots F1 score as a function of the model's confidence over the two classes "awake" and "drowsy." The thick blue line denotes the global performance over all classes, which reaches a maximum F1 score of 0.89 when the threshold over the confidence is set to 0.708.
Main observations:
Awake Class Performance: The F1 score for the "awake" class (blue line) is consistently higher than that of the "drowsy" class across the confidence range. This suggests that the model is more accurate in detecting when a person is awake compared to when they are drowsy.

Drowsy Class Performance: The "drowsy" class (orange line) has a lower F1 score, indicating that the model struggles more with detecting drowsiness. This could be due to the model having fewer distinguishing features for drowsy states or fewer samples during training for this class.

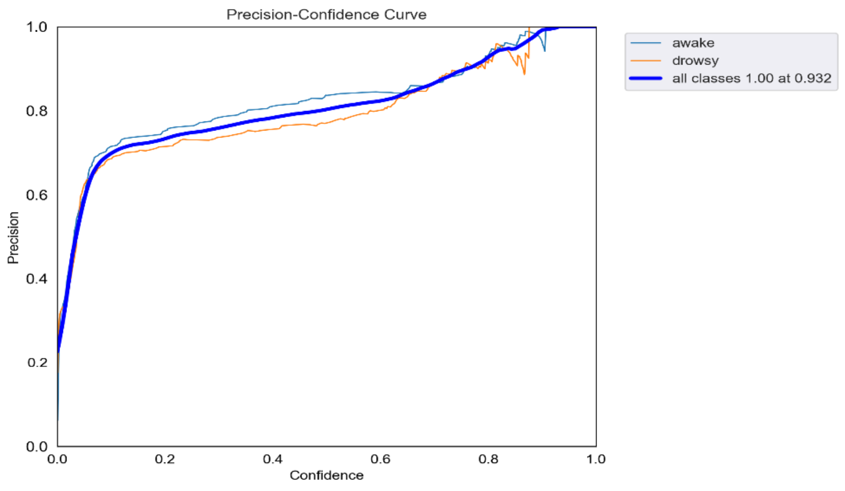
Confidence Threshold: As the confidence level increases, the F1 score initially rises and peaks around 0.8 for both classes, after which it sharply drops. This shows that the model performs best with moderate confidence thresholds and starts misclassifying instances when the confidence is pushed too high.

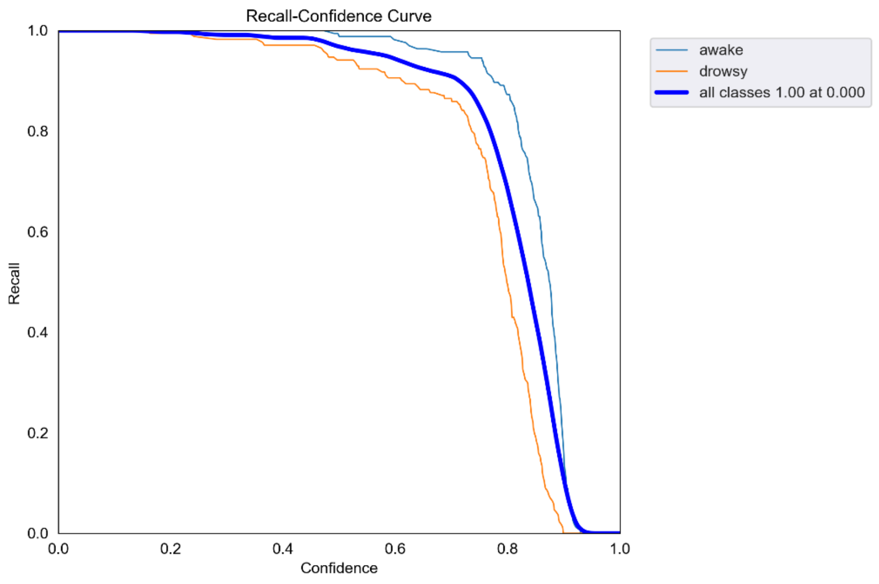
Batch Prediction :

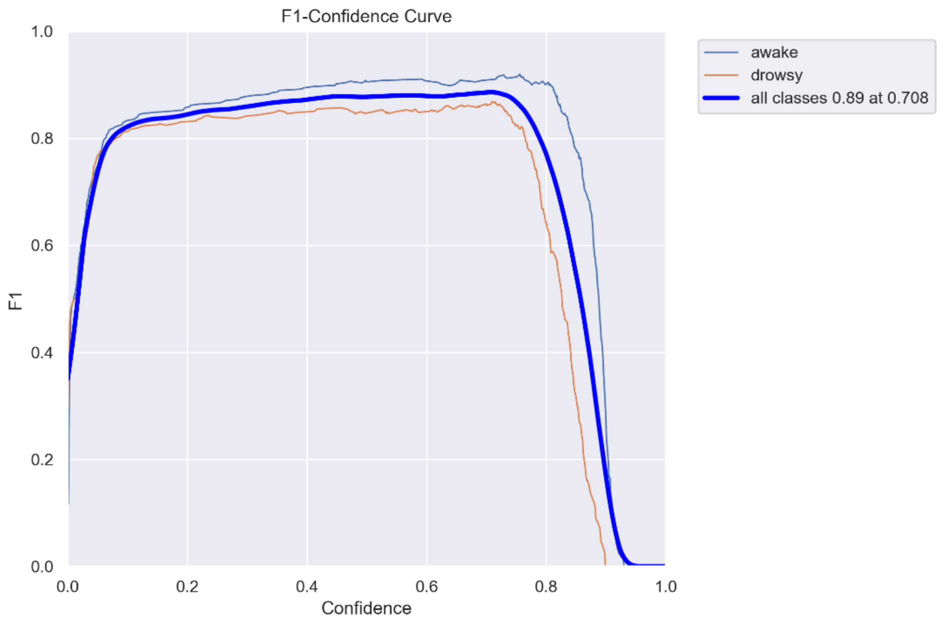




Result Visualization :







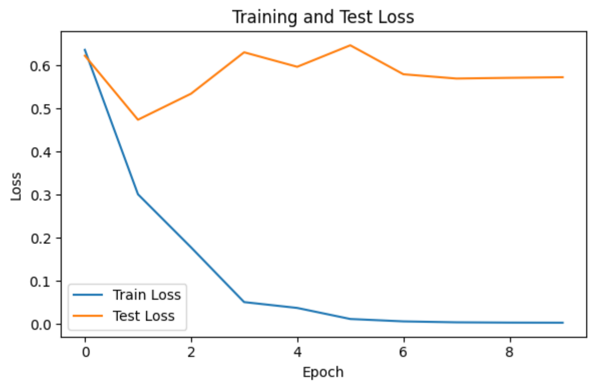
5.2 Dataset 2 : Experiment 3 & 4

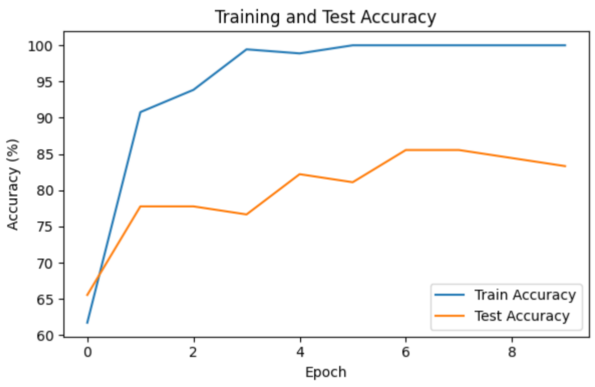
Training and Test Loss:

The training loss decreases rapidly, nearing zero by the 5th epoch, indicating that the model is fitting the training data well. However, the test loss decreases initially but starts increasing after the 2nd epoch, suggesting overfitting as the model performs better on the training set than on the test set.

Training and Test Accuracy:

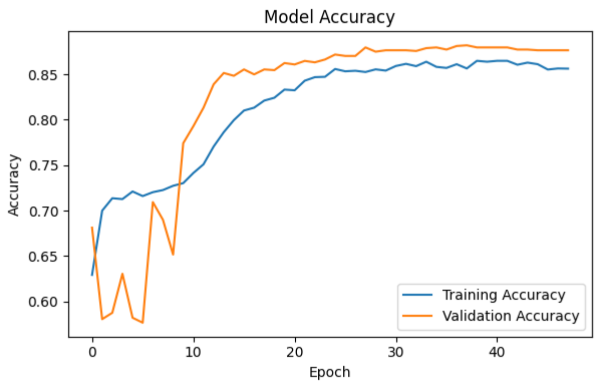
Training accuracy climbs sharply, reaching almost 100% by the 4th epoch, while test accuracy plateaus around 75-80%. This further confirms overfitting, as the model generalizes poorly to unseen data.

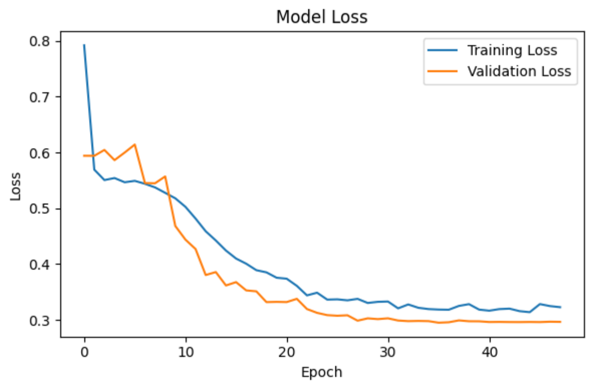




Analysis of Training and Validation Performance

The graphs display the training and validation performance metrics for the implemented drowsiness detection system over 50 epochs. The accuracy plot shows steady improvement in both training and validation accuracy, with the validation accuracy stabilizing at around 0.85, indicating strong model generalization. However, the training accuracy surpasses the validation accuracy, reflecting consistent learning without significant overfitting. In the loss plot, the training loss steadily decreases, while the validation loss stabilizes around epoch 20 and maintains a low value, confirming effective model optimization. These results suggest that the model is well-trained and balanced for predicting drowsiness accurately across both the training and validation datasets.





APPENDIX-1

This authorisation file should be in same folder as of fetch data.

```python
import os
import google.auth.transport.requests
from google_auth_oauthlib.flow import InstalledAppFlow

SCOPES = ['https://www.googleapis.com/auth/fitness.heart_rate.read']
CLIENT_SECRETS_FILE = "D:\Sarthak\ProjectExhibitio\client_secret_496433399505-
i9t2dcedrv2d4gf5emvh7ksa8l9lsq1t.apps.googleusercontent.com.json"

def authenticate_google_fit():
    creds = None
    if os.path.exists('token.json'):
        creds = google.auth.load_credentials_from_file('token.json')[0]

    if not creds or not creds.valid:
        flow = InstalledAppFlow.from_client_secrets_file(CLIENT_SECRETS_FILE,
SCOPES)
        creds = flow.run_local_server(port=8080)

        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    return creds
```

APPENDIX-2

This is Google Fit API which which uses auth token genereted by code in APPENDIX-1.

```python
import time
from googleapiclient.discovery import build
from auth import authenticate_google_fit
from datetime import datetime, timedelta

def get_heart_rate_data(creds, start_time, end_time):
    service = build('fitness', 'v1', credentials=creds)

    # Data source ID for heart rate data
    data_source =
'derived:com.google.heart_rate.bpm:com.google.android.gms:merge_heart_rate_bpm
'

    # Convert start_time and end_time to nanoseconds
    start_ns = int(start_time.timestamp() * 1_000_000_000)
    end_ns = int(end_time.timestamp() * 1_000_000_000)

    dataset = f"{start_ns}-{end_ns}"

    try:
        # Fetch the heart rate data
        result = service.users().dataSources().datasets().get(
            userId='me',
            dataSourceId=data_source,
            datasetId=dataset
        ).execute()

        # Process and display the heart rate data points
        if result.get('point'):
            for point in result['point']:
                start_time_ms = int(point['startTimeNanos']) / 1_000_000
                end_time_ms = int(point['endTimeNanos']) / 1_000_000
                heart_rate = point['value'][0]['fpVal']
                start_time_str = datetime.fromtimestamp(start_time_ms /
1000).strftime('%Y-%m-%d %H:%M:%S')
                end_time_str = datetime.fromtimestamp(end_time_ms /
1000).strftime('%Y-%m-%d %H:%M:%S')
                print(f"Heart Rate: {heart_rate} bpm | From: {start_time_str}
To: {end_time_str}")
        else:
            print(f"No heart rate data found between {start_time} and
{end_time}")

    except Exception as e:
        print(f"An error occurred: {e}")
```

```python
def fetch_real_time_data(creds, interval_seconds=60):
    print("Starting real-time heart rate data fetching...\n")

    while True:
        # Get current time as end_time
        end_time = datetime.now()

        # Set start_time to a few seconds before the current time for real-
time data
        start_time = end_time - timedelta(seconds=interval_seconds)

        # Fetch heart rate data for the interval
        get_heart_rate_data(creds, start_time, end_time)

        # Wait for the interval duration before fetching the data again
        time.sleep(interval_seconds)

if __name__ == '__main__':
    creds = authenticate_google_fit()  # Authenticate and get credentials

    # Fetch real-time data every 60 seconds (1 minute)
    fetch_real_time_data(creds, interval_seconds=3600)
```

References

[1] *Quddus, Azhar & Shahidi Zandi, Ali & Prest, Laura & Comeau, Felix. (2021). Using long short term memory and convolutional neural networks for driver drowsiness detection. Accident Analysis & Prevention. 156. 106107. 10.1016/j.aap.2021.106107. https://www.researchgate.net/publication/350801877_Using_long_short_term_memory_and_convolutional_neural_networks_for_driver_drowsiness_detection*

[2] *Esra Vural ,Müjdat Çetin , Aytül Erçil , Gwen Littlewort , Marian Bartlett , Javier Movellan. 1 November 2008 ,Ford Otosan. https://core.ac.uk/outputs/11740526/?utm_source=pdf&utm_medium=banner&utm_campaign=pdf-decoration-v1*

[3] *Ed-Doughmi, Y., Idrissi, N., & Hbali, Y. (2020). "Real-Time System for Driver Fatigue Detection Based on a Recurrent Neuronal Network." Journal of Imaging, 6(3), 8.*

[4] *Zandi, A. S., Quddus, A., Prest, L., & Comeau, F. J. E. (2019). "Non-Intrusive Detection of Driver Drowsiness Using Eye Tracking Data." Transportation Research Record, 2673(9), 10.1177/0361198119847985.*

[5] *Kulhandjian, H. (2021). "Detecting Driver Drowsiness with Multi-Sensor Data Fusion Combined with Machine Learning." Mineta Transportation Institute.*

[6] *Salman, R. M., Rashid, M., Roy, R., Ahsan, M. M., & Siddique, Z. (2021). "Driver Drowsiness Detection Using Ensemble Convolutional Neural Networks on YawDD." arXiv. https://arxiv.org/abs/2112.10298.*

[7] *Md. Tanvir Ahammed Dipu ,Hossain, Syeda & Arafat, Yeasir & Rafiq, Fatama. (2021). Real-time Driver Drowsiness Detection using Deep Learning. International Journal of Advanced Computer Science and Applications. 12. 844-850. 10.14569/IJACSA.2021.0120794.' https://www.researchgate.net/publication/353620881_Real-time_Driver_Drowsiness_Detection_using_Deep_Learning*

[8] *Ch, Venkata & Reddy, U. Srinivasulu & KishoreKolli, Venkata. (2019). Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State. Revue d'Intelligence Artificielle. 33. 461-466. 10.18280/ria.330609. https://www.researchgate.net/publication/338251837_Deep_CNN_A_Machine_Learning_Approach_for_Driver_Drowsiness_Detection_Based_on_Eye_State*

[9] *Deng, W., & Wu, R. (2019). "Real-Time Driver-Drowsiness Detection System Using Facial Features." IEEE Access, 7, 118727-118738. https://doi.org/10.1109/ACCESS.2019.2936663*

[10] *Abbas, Q., Hussain, S., & Shad, S. A. (2020). "HybridFatigue: A Real-time Driver Drowsiness Detection using Hybrid Features and Transfer Learning." ResearchGate.*

[11] *Saif, A. F. M., & Mahayuddin, Z. R. (2020). "Robust Drowsiness Detection for Vehicle Driver using Deep Convolutional Neural Network." International Journal of Advanced Computer Science and Applications, 11(10). https://doi.org/10.14569/IJACSA.2020.0111043*

[12] *Yu, J., Park, S., Lee, S., & Jeon, M. (2018). "Condition-Adaptive Representation Learning for Driver Drowsiness Detection Using 3D-Deep Convolutional Neural Networks." IEEE International Conference on Intelligent Transportation Systems (ITSC), 8580568. https://doi.org/10.1109/ITSC.2018.8580568*

[13] *Ramzan, M., Khan, H. U., Awan, S. M., Ismail, A., Ilyas, M., & Mahmood, A. (2019). "A Survey on State-of-the-Art Drowsiness Detection Techniques." IEEE Access, 7, 61904-61919. https://doi.org/10.1109/ACCESS.2019.8704263*

[14] *Kolus, A. (2024). "A Systematic Review on Driver Drowsiness Detection Using Eye Activity Measures." IEEE Access, 12. https://doi.org/10.1109/ACCESS.2024.3424654*

[15] *Intelligent Driver Drowsiness Detection for Traffic Safety Based on Multi CNN Deep Model and Facial Subsampling Ahmed, M., Masood, S., Ahmad, M., & El-Latif, A. A. (2021). "Intelligent Driver Drowsiness Detection for Traffic Safety Based on Multi CNN Deep Model and Facial Subsampling." IEEE Transactions on Intelligent Transportation Systems.*

[16]

Citation*: Ngxande, M., Tapamo, J. R., & Burke, M. (2017). "A Review of Machine Learning Techniques for Driver Drowsiness Detection Based on Behavioral Measures." IEEE 46th Annual Conference of the Industrial Electronics Society (IECON), 8261140. https://doi.org/10.1109/IECON.2017.8261140*

[17] *Du, Y., Ma, P., Su, X., & Zhang, Y. (2008). "Driver Fatigue Detection Based on Eye State Analysis." Proceedings of the 11th Joint Conference on Information Sciences, Atlantis Press.*

[18]

Citation*: Al-Madani, A. M., Gaikwad, A. T., Mahale, V., Ahmed, Z. A. T., & Shareef, A. A. A. (2021). "Real-Time Driver Drowsiness Detection Based on Eye Movement and Yawning Using Facial Landmarks and Dlib." IEEE International Conference on Data Science and Information Technology (DSIT), 9457005. https://doi.org/10.1109/DSIT.2021.9457005*