Drowsiness Detection In Automobile Drivers

Priyanka Akula, Priya Varahan, Pooja Manjunatha and Nandini S Nair

Department of Data Analytics, San Jose State University

FA23: DATA-270, Section 11, Data Analytics Process

Dr. Eduardo Chan

Dec 9, 2023

Abstract

Driver fatigue is a significant contributing factor to automobile accidents globally. It typically arises from fatigue or insufficient sleep. In order to address this issue, it is necessary to implement proactive measures to alert drivers when they are excessively fatigued to operate a vehicle. This research proposes an effective approach to identify fatigued drivers by employing deep learning techniques. The detection system utilizes a Night Time Yawning Microsleep Eyeblink Distraction video dataset. This dataset is then transformed into a frame level dataset consisting of three distinct components: yawning, sleepiness accompanied by microsleep episodes, and degrees of alertness. Prior research employing Residual Network models had an 85% success rate in detecting sleepiness. However, this paper aims to enhance these findings by examining other models, including the Visual Geometry Group19, Viola Jones, You Only Look Once version 3, and Convolutional Neural Networks, with the objective of improving the accuracy of detection. The suggested models are assessed using gathered data and provide a superior accuracy of 97% for the Visual Geometry Group19 in comparison to the conventional Residual Network model. The ultimate outcomes of these models are crucial for enhancing and developing more sophisticated systems capable of promptly detecting driver fatigue, hence reducing the incidence of drowsy driving-related accidents.

Keywords: driver drowsiness, deep learning, degrees of alertness, and residual network.

Drowsiness Detection In Automobile Drivers

Introduction

Automobiles play a major role in our daily life. On a daily basis, people travel to work, school, etc. Individuals can now go to different locations with greater convenience and efficiency because of this technology, yet there is a drowsiness in drivers. Drivers who are drowsy run the possibility of fatal events including accidents, injuries, and in the worst circumstances, death. This may take place particularly at night. However, sleep can strike a motorist at any time, with disastrous or lethal results. Numerous factors can contribute to drowsiness such as poor routines for eating and sleeping, medications, diet, alcohol, lack of sleep, aging, stress, and boredom when driving. There is a problem that requires a solution. Around the globe, countless lives will be saved if this issue is resolved.

The National Safety Council (NSC) reports that intoxicated driving causes 100,000 collisions and 1,550 fatalities annually (Rivelli, 2022). The purpose of this research article is to apply deep learning and machine learning methods to solve this challenge. The models developed should be extremely effective at anticipating drowsiness. It may be too late to detect drowsiness a few seconds later because the accident might have already occurred. Therefore, utilization of four prominent algorithms and evaluating each model's prediction precision and comparative analysis of the same is done. You Only Look Once version3 (YOLOv3) , Visual Geometry Group19 (VGG19), Convolutional Neural Network (CNN), and Viola Jones are the algorithms that are employed.

A neural network technique called YOLOv3 is used to detect objects. An image is divided into a grid by this deep learning algorithm, which then forecasts bounding boxes. There are class probabilities in these grids. YOLOv3 is renowned for its accuracy and real-time

performance. YOLOv3 is renowned for its ability to identify multiple objects and scale. It looks for indicators of driver sleepiness by sifting through photos and videos. Closed eyes and other facial features are the cues. In order to use this method for training, the photos must be annotated. For this YOLOv3 is used.

An object identification deep learning architecture is the CNN. It has a reputation for precise object detection. When it comes to finding patterns in the image dataset, CNN excels. In order to examine minute details inside the image and video collection, this model closely examines photos and videos. It can identify indicators of fatigue by capturing the head posture and additional face traits. The network's deep layers enable it to identify intricate patterns that ultimately reveal the driver's level of tiredness.

Image characteristics are extracted by VGG19. To learn visual features, deep convolutional layers in the network are taken advantage of. In this algorithm, transfer learning is being used. The dataset selected for this project can be used with a pre-trained VGG19. Based on the information learned during training, the network will identify patterns of tiredness in the facial characteristics with help of a labeled dataset.

An algorithm for machine learning object detection is called Viola-Jones. The other term for it is the Haar Cascade Classifier. It has object detection capabilities. On features, it creates rectangular patterns. This algorithm also requires a labeled dataset. Classifiers are present in it, and they are arranged in a cascaded structure. Only the drowsy samples are kept in this model's final step, while all of the alert samples are eliminated. It is used to categorize an eye's condition as open or closed.

The proposed models seek to use the supervised-learning algorithms to generate a quick and effective solution to stop traffic accidents and fatalities brought on by tired drivers.

Pre-processing the data, applying the algorithms, training the model and evaluation of the models' accuracy to select the best one are the stages involved. To improve the algorithm's ability to distinguish between different people's levels of drowsiness, the dataset includes images of people with a wide range of facial features, such as mustaches, glasses, and varying eye sizes. The elements of the system design, from the initial stages of data collecting to the final documentation, are covered in the following sections of the research paper. Our goal is to develop a solution for driver drowsiness. Choosing the optimal algorithm can help reduce the dangers associated with driving while drowsy thereby improving road safety. The prediction accuracy of the algorithms is compared in this research article.

Project Requirements

Functional Requirements

Reliable machine learning algorithms can be developed by utilizing the data from Night-Time Yawning-Microsleep-Eyeblink-driver Distraction (NITYMED) dataset. The four algorithms that need to be used in order to predict the algorithm with the highest accuracy are YOLOv3, VGG19, CNN and Viola Jones. The picture data set is processed, stored, and analyzed using python in our local system. Large amounts of unstructured raw image data will be initially stored in the local system.The algorithms are implemented using the open-source machine learning framework TensorFlow.The algorithm is assessed by computing accuracy, F1 score. The model that can forecast drowsiness the best will have the highest accuracy. Lastly, Tableau is used to create a dashboard for visualization needs.

AI Powered Requirements

The AI-powered solution requires the utilization of machine learning models such as YOLOv3, VGG19, CNN, and Viola Jones for tasks comprising face detection, eye state classification, yawning detection, and head pose assessment.To be more specific, use of the Viola-Jones algorithm for face detection and recognition.In optimal illumination circumstances, this algorithm obtains a 95% accuracy rate. Even if the video or picture collection contains more than one face, the project is still able to identify the driver's face. VGG19, CNN, are two options for classifying eye states. When the eye's condition is open or closed an accuracy of up to 85.7% is achieved using the models(Chandra et al., 2021). Given their high level of efficiency, these systems can identify microsleep with a high accuracy when an eye closes for longer than 1.5 seconds.

To identify yawning tendencies in drivers, YOLOv3 is employed. 95.9 % accuracy is achieved by this algorithm (Soe et al., 2022). In addition, the CNN algorithm is used to evaluate head posture. Most of the time, the system detects situations in which the driver's head tilts because of fatigue.

If the idea is applied to cars, real-time data analysis with minimal latency and maximum accuracy can be achieved by employing an algorithm. This will benefit society as a whole. On a diversified dataset that guarantees representation from various ethnicities, age groups, etc., model training and ongoing evaluation are also necessary. Data gathering and driver alerting in situations where drowsiness is identified can be accomplished with hardware systems currently found in cars, such as cameras and sensors. In the event of false alarms, drivers can manage the feedback mechanisms in this approach, which are quite significant. However, the system can use

these feedbacks to build on prior knowledge. These artificial intelligence needs are unique to the selected machine learning algorithms that, when implemented, will produce quantifiable results.

Data Requirement

NITYMED dataset is used. Additionally, there are about 130 movies in the dataset that show drivers in actual driving cars at night. The vehicle had the hardware installed. The categories in the movie that have attempted to cover the data of 21 drivers with diverse descriptive features include microsleep and yawning. To identify the drowsiness in this film, it must first be transformed to frames, processed, and then examined utilizing algorithms. In contrast, the NITYMED dataset has about 15,000 student points, and is accessible to the general public. Data on 37 distinct individuals which includes 33 women and 4 males with a range of descriptive attributes are included in this image data set. Some of the descriptive attributes include male or female, does the person wear glasses or not, eye state of driver is closed or open, reflections are small, large, or nonexistent, light conditions is the lighting good or bad during the image capture process and sensor ids of the images are categorized based on three distinct sensors that capture them, the models are used to evaluate this image collection. The best accurate model for drowsiness detection is predicted at the end of the model training and evaluation process.

Project Deliverables

Deliverables comprises important phases of the project, from project proposal to the final report. The following are the list of project deliverables which is mentioned in Table1.

Project Proposal

Different issue statements and scenarios were discussed in order to identify the best problem to solve and strategy to use. The focus was on sleepy driving, which causes traffic

accidents and fatalities. The intricacy of drowsiness made it difficult to correctly diagnose and treat. In order to build a deep learning model that would predict driver tiredness with accuracy and reliability, the strategy called for using historical drowsiness data. The process was directed by the Cross Industry Standard Process for Data Mining (CRISP-DM) methodology for manageable iterations or sprints, which were applied to each phase to facilitate the shift to an Agile methodology. Four different deep learning models were created for the study with the goal of evaluating past data and producing accurate predictions.

Introduction

The proposed research problem, together with its background, goals, importance, and necessity, are all included in the introduction document. Along with details on the data that is used, anticipated contributions, and real-world applications of the research, the approaches employed to address the problem are provided. A study of the prior studies' literature, a comparison of the various models, techniques, and conclusions, and related research in the field are covered.

Project Management Tool And Work Breakdown Structure

Project management software Jira was used to efficiently distribute and manage all of the activities. Each step was documented. The project milestones, component deliverables, and work packages are listed in the Work Breakdown Structure (WBS).

Effort Estimates, Pert, And Gantt Chart Creation

The paper provides a list of the effort that went into each of the research project deliverables. In addition, a Gantt chart and a PERT (Program Evaluation and Review Technique) chart is included. The Gantt chart shows the tasks, length, status, and individuals in charge of

each deliverable for the project. The PERT chart presents the individual tasks, dependencies, and milestones in a systematic manner.

Data And Project Management Plan

The methods for gathering data, the development procedures that is involved, the organization, software, and hardware requirements of the project, a work breakdown structure, a Gantt chart that shows the tasks, deadlines, resources required, and deliverables for each team member, and a PERT chart that shows the specific tasks and dependencies is included in this document.

Data Collection And Data Exploration

In order to help address proposed research concerns the data collection and exploration document offered a variety of methods for obtaining and compiling data from various sources. It included details about the location of the data, how it was gathered, how long it took to acquire, and who was in charge of gathering it. It also offered a detailed explanation of every possible insight that might be gleaned from the data.

Data Engineering

This document has detailed steps involved in data collection, pre-processing, data transformation, data preparation, and data reduction.

Abstract

The paper gives a summary of the topics under study, the specific research question, the problem statement that is being addressed, and some background data on earlier studies.

Project Presentation

PowerPoint presentation gives an overview of the problem and how the research is being used to solve it.

Individual Research Paper On Model Development

Document contains details on the deep learning model that each member of the team built on their own. Each team member has a paper with different deep learning models highlighted in it.

Group Project Report

The project report would include an explanation of the objectives of the project, a description of the methods used to achieve those objectives, a discussion of the difficulties encountered, and the project's outcomes. It would give a summary of the subject and present the research results.

Table 1

Project Deliverables And Timeline

Deliverable	Description	Due Date
Project Proposal	Propose Paper Detection For Drowsiness Detection	**09/21/2023**
Project Breakdown	Creation Of WbS In Jira	**09/30/2023**
Project Plan	Creation Of Tasks,Subtask And Task Assignment, Gantt Chart And Pert Chart	**10/06/2023**
Data Collection	Collection Of The Data	**09/23/2023**
Prototype	Model Implementation	**10/21/2023**
Testing	Evaluation Of The Models	**10/26/2023**
Final Model	Comparisons Of Model And Identification Of The Best Model	**11/05/2023**
Project Report	Documentation of the project	**11/20/2023**

Note. Table Listing The Project Deliverables, Their Purpose And Dates.

Technology And Solutions Survey

Identifying driver drowsiness is critical to preventing accidents. Over time, a plethora of technologies and solutions have emerged to address this problem. This analysis delves into the latest advancements in drowsiness detection technologies, methodologies, and software and hardware solutions, along with the drawbacks of employing models, YOLOv3 ,VGG19, CNN , and Viola Jones.

Convolutional Neural Network

CNNs are very effective at detecting drowsiness because of how well they can process image data. The human face exhibits telltale signs of exhaustion, such as drooping eyelids, prolonged blinks, and a hunched head. CNNs are trained on copious amounts of tagged visual data, enabling them to pick up on these minute variations. After being trained, they are able to determine a driver's level of sleepiness from live facial photos. CNNs are powerful because they can automatically learn and adjust spatial feature hierarchies from images without requiring human feature selection.

Ali et al. (2021) employed Dlib in conjunction with a CNN to perform feature extraction and dataset categorization. This method successfully detects driver fatigue, which contributes to the reduction of collisions. To detect if the driver is dozing off or has their eyes closed, the system makes use of a CNN model that has been trained beforehand. The driver is deemed drowsy if the results of both CNN and Dlib are integrated, which measures the degree of drowsiness.

The usage of a combination of techniques, including OpenCV, Viola Jones, and CNN, was investigated (S et al., 2022). For accuracy, the study concentrated on elements like illumination and camera distance. Here, CNN uses a 244 x 244 input image to evaluate camera

data and identify open and closed eyes. The design makes use of the Adam optimizer-trained Softmax in the final integrating layer, Rectified Linear Unit (ReLU) activation in the convolutional layers, and average pooling. The study suggested a real-time data gathering system, which is becoming more and more practical given that contemporary cars come with integrated sensors and alerts.

VGG19

Bajaj et al. (2021) conducted research on the identification of essential face traits to assess fatigue levels and detect driver drowsiness. These particular facial landmarks are essential for identifying sleepiness. CNNs with the VGG19 architecture were used in the study, ensuring fast recognition of sleepiness signs and effective computation. To show these patterns, the system produces graphical representations. The size of the input image is set to (224, 224, 3). The architecture consists of a series of max pooling and convolutional layers, each with a different size filter. To improve the model's discerning capacity, an additional attention mechanism could be included to highlight finer details.

Nevertheless, controlling several parameters and convolutional layers becomes difficult when employing high-complexity approaches. One major problem that persists is overfitting, especially when dealing with smaller datasets. The use of regularization techniques like dropout or weight decay is being explored as a way to lessen this. Using VGG19 models that have already been trained and adjusting them for certain tasks or domains could significantly lessen the need for large datasets and processing capacity. This method is intended to facilitate effective application and training in a variety of contexts.

YOLOv3

Luo (2023) used the YOLOv3 algorithm in a study to detect driver tiredness. They employed the cameras in automobiles for real-time surveillance to construct a training and testing dataset containing labeled real-world instances, with the goal of preventing accidents brought on by sleepy drivers. To reduce the burden on the graphics processing unit (GPU), a pre-trained model was used rather than creating a new one. The public dataset had pictures of two bus drivers with labels that indicated whether they were "awake" or "drowsy." The tool utilized for labeling was a target-detection dataset annotation tool called LabelIng. Every image had a text file with its label attached. The confusion matrix used to calculate the results showed that it was hard to tell one category from another. Reduced confidence results in larger model predictions, according to the recall-confidence curve. The purpose of the article was to recognize all data categories with high accuracy, recognizing that one major restriction in huge datasets is Macalisang the manual tagging of photographs.

Real-time sleepiness detection was investigated by Macalisang et al. (2021), alerting drivers when they seem drowsy. Participants in the study who had hearing difficulties reported that 92% of them woke up at 450 Hz and 57% at 3100 Hz. They also observed that people can become sleepy or awakened by vibrations in their automobile seats. They separated the two sorts of their system: vehicle-oriented and driver-oriented. It used sensors to track steering wheel movement, pedal pressure, and vehicle speed in order to identify signs of tiredness. The driver-oriented system included facial feature detection. They used the pre-trained YOLOv3 algorithm to train a Kaggle dataset. Compared to RetinaNet, this model labeled with LabelImg showed superior accuracy and was 3.8 times faster. The model's results include a 100%

mean-average precision (mAP), 97.3609% training accuracy, 98.5993% validation accuracy, and 100% final test accuracy. Every time the automobile sensed tiredness, an alarm went off.

Viola Jones

The goal of the research was to create a system that could identify drivers who were sleepy. The webcam on a car is used by the system to take pictures. An alarm is set off when these images match the system's specified "awake" and "drowsy" eye states. To detect eyes, the Viola-Jones method is used. The true positive rate (sensitivity), true negative rate (specificity), false positive rate, false negative rate, precision, recall, F1 score, and the Area Under the Receiver Operating Characteristic Curve (AUROC) are some of the measures used to evaluate the correctness of the model. The technology is made to distinguish between a single blink and prolonged eye closure in order to prevent false alarms. Alerts are generated via image processing techniques such as edge and peak detection. Because this approach is simple to incorporate into current automotive systems, the authors support it. The technology showed a high accuracy rate of almost 95% (Kasodu et al., 2022) after evaluation. Even while this non-invasive method is already very precise, it might be improved even further, particularly under different lighting circumstances.

Literature Survey Of Existing Research

This section addresses some of the major studies that address the issue of drowsiness detection.To fully grasp the complexities of this undertaking, reading through a number of research articles helps. After doing independent study, a decision to evaluate the algorithms' forecast accuracy was made.

Keeping students' attention in the classroom is a task. Lost students can cause the teacher to lose focus, which can ultimately result in the loss of control over the class. This is a crucial

component of what students are learning. Therefore, this paper uses computer vision and machine learning approaches to identify student behavior in the classroom and improve learning.

The CNN and YOLOv3 algorithms are employed. In order to determine if a student is paying attention in class, these algorithms can examine their eye movements and facial expressions. It can identify a student's level of focus, level of fatigue, and emotional state.

The FER (2013) dataset served as the model's training set. The model is assembled using the cross-entropy loss function, accuracy measurements, and Adam optimizer. A confusion matrix is employed in the model's assessment. According to Afsar et al. (2023), the accuracy of the suggested system is 93.12%.Driving while fatigued can be dangerous and increase the risk of significant accidents and injuries. In this research, a system for detecting driver drowsiness using YOLOv3 and Jetson Nano is proposed. A beep alerts the motorist to the impending danger.The hardware that is mounted on the car's dashboard is what gathers the data. The Jetson Nano recognizes the driver's mouth and eyes. If it is determined that the driver is sleepy, the buzzer will ring. Through simulations, the accuracy of this model is verified. The dataset may be found on int.net. It has two thousand photos in it. The three image classifications are: closed eyes, open eyes, and yawning mouth. The labeled dataset exists.

The dataset is divided into datasets for testing, validation, and training. In order for the data to be input into the YOLOv algorithm, it must first undergo preprocessing. The dataset has been enhanced by 25%. Next, this dataset is fed into YOLOv3. Although YOLOv3 offers a large amount of computer power, its precision is not that great. Increasing the dataset and changing the hyperparameters will yield high accuracy.In order to identify the most accurate model, hyperparameters are varied during the training phase. This covers the F-1 score, recall, accuracy, precision, and false positives and true positives.

Next, the Jetson Nano is equipped with the optimal model. Epoch 150 has the highest accuracy. In every model, the recall value of the yawning and closed eyes class is higher than that of the usual class. Then, in a chamber with lighting conditions akin to an automobile, this model is tested. The model's confidence values for open eyes, near yawning, and 0.72, 0.82, and 0.75, respectively, are given (Lazuardi et al., 2023). After that, hardware is used to implement this concept. The detecting method is quick even though the FPS shooting is slow. They intend to incorporate methods that will raise fps values in their upcoming development.

The goal of the research was to create a system that can examine facial expressions and eye movements to identify sleepiness in drivers of motor vehicles. When the system senses that a driver is drowsy it sounds an alert and applies the brakes, which keeps the car from colliding. Data is obtained via the car's mounted webcam. An alarm is triggered by this data when it is compared to photographs of both open and sleepy eyes stored in the system. For eye detection, the Viola-Jones algorithm is employed. In order to minimize false alerts, the system is built to distinguish between a single blink and extended eye closure. Edge detection, grayscale conversion, and noise filtering are accomplished by the use of image processing techniques, such as the Sobel operator. Metrics like the true positive rate (sensitivity), true negative rate (specificity), false positive rate, false negative rate, precision, recall, F1 score, and Area Under the Receiver Operating Characteristic Curve (AUROC) are used in the study to gauge how accurate the model is. The Hough Transform Algorithm is also described, which helps determine the condition of the eyes based on the separation between the eyebrows and eyelashes by linking edge points to create edge lines. The combination of algorithms that strike a balance between computing efficiency and accuracy is what makes the suggested system effective (Sur et al., 2015).

The study suggests a novel approach for a real-time sleepiness monitoring system that uses a camera to track drivers' facial expressions and measure how often they blink and yawn. This method combines the modeling techniques of structured Support Vector Machine (SVM) and Supervised Descent Method (SDM). The structured SVM is trained using Yaw datasets in order to optimize yawn detection. This model incorporates the Viola-Jones algorithm for real-time face detection, which uses SDM face alignment techniques to locate facial landmarks. These aspects of the face are used to detect yawning and blinking rate. Standard accuracy criteria such as precision, recall, and F1-score are used to evaluate the efficacy of this technique, which yields an overall accuracy rate of 81% (Tyagi et al., 2022).

The study fills a vacuum in the driver monitoring systems sector by using the Viola-Jones algorithm to recognize drivers' faces and eyes. For pattern localization, the Haar cascade technique is also used. This system's primary function is to monitor drivers' eyes and sound an alert if it detects extended eye closure. The Eye Aspect Ratio (EAR) measures the distances between the eyes in both the horizontal and vertical directions to determine if the eyes are open or closed and to diagnose drowsiness. A blink is indicated by a rapid drop in the EAR value followed by an increase. The model predicts drowsiness based on the EAR value, particularly when the EAR value is consistent and drops below a predetermined threshold. Additionally, the system has a virtual assistant built to communicate with drivers and support the preservation of attentiveness (Neshov & Manolova, 2017).

The study fills a vacuum in the literature on driver monitoring systems. A proposed method may manually handle mouth blockages and upper body movements. The idea is to use camera video to train a neuromorphic algorithm that can recognize yawns in drivers and track whether or not they are wearing seatbelts. Data from the Prophesee event camera is combined with

recordings from the dashboard and rear view cameras from the Public YawDD collection for research purposes. While Recurrent Neural Networks (RNN) with long short-term memory (LSTM) cells are utilized for seatbelt status detection, CNN are used for yawn detection. Artificially created event data is used to train both models. Accuracy is assessed using the seatbelt state detection dataset, and F1 score, precision, and recall are used to gauge the yawn dataset's performance. The seatbelt state identification results demonstrate a strong F1 score of 0.96 on the YawDD dataset and 0.99 on a real-event dataset. To broaden the scope of the research, next work will require collecting more real-world data and gaining access to more public statistics for seatbelt status and yawning. According to Kielty et al. (2023), these models might be incorporated into driver monitoring systems' embedded hardware.

The goal of the research is to use artificial intelligence and Internet of Things technology to develop a dependable, real-time system for tracking driver drowsiness, with an emphasis on ocular aspect ratio. The MRL Eye Dataset and a custom dataset are combined to create the dataset. In order to train a deep learning model to analyze eye aspect ratios, the MRL dataset contains pictures of human eyes in both open and closed conditions. Images of drivers at different phases of tiredness can be found in the custom dataset. Using both custom datasets and the MRL eye for training, a CNN architecture built in Keras is employed. The model's evaluation metric is the Area Under the Receiver Operating Characteristic (AUROC) curve, and it has an amazing Area Under the Receiver Operating Characteristic (AUROC) of 97.88% (Soman et al., 2023). Potential future improvements could entail combining the model with other gadgets, like as fitness trackers, to provide a thorough physiological status analysis of the driver .

Ali et al. (2021) combined method achieves an accuracy rate of about 98%. Even though the CNN and Dlib algorithms use different approaches, the combined result works incredibly well. It

is noteworthy that, according to the technical paper, the trained dataset attains an average accuracy rate of 98%, whereas the non-trained dataset delivers an accuracy rate of 95%. Comparing this performance to another study by Dua et al. (2020), the suggested approach demonstrates an accuracy of 83%.

S et al. (2022b) study incorporates a number of algorithms, such as CNN, Viola Jones, and Opencv. There is evidence that the model's training and validation losses are parallel, proving there was no overfitting. The CNN performance of the model for each circumstance is good but not exceptional, since the other model accuracies are higher than 90%. There was just a 10% difference in accuracy between yawning and identifying eye problems in a study looking at indicators of tiredness. System accuracy falls below 70% when the camera is positioned too close to the driver, especially at a distance of 0.25 meters. Lighting affects performance, with daytime viewing yielding higher outcomes than nighttime viewing. While yawning detection retains 70% accuracy, eye condition detection in the dark only reaches 40%. The absence of a night mode on the camera is the reason for the reduced accuracy at night. On the other hand, the system reliably and consistently detects indicators of sleepiness with over 80% accuracy under specific settings. It is advised by the authors to position the camera at least 0.5 meters away from the driver for best results.

A sophisticated method to detect drivers who are at risk of falling asleep at the wheel was unveiled by Bajaj et al. (2021) at the 7th International Conference on Advanced Computing and Communication Systems (ICACCS). This system's main goal is to address the serious problem of driver fatigue-related accidents. CNN and computer vision techniques are used in the suggested system, which is smoothly incorporated into an Android application for effective and user-friendly implementation. Fatigue detection using facial feature analysis eliminates the need

for extra sensors on the driver by analyzing the length of eye blinks and closures. As part of the system architecture, real-time video streams are recorded, saved in cloud-based infrastructure, and then examined using CNN models such as VGG19.

Although this methodology shows potential, the system under investigation has a number of drawbacks that need to be carefully considered. The use of cloud architecture for the analysis and storage of video streams presents a number of challenges, including the potential for latency and connectivity problems, particularly in places with inadequate network infrastructure. Furthermore, because different people have a wide range of facial expressions, the system's reliance on facial feature analysis may make it challenging to effectively detect tiredness across a diverse driver demography. Furthermore, variations in illumination and camera angles are examples of environmental factors that may have an impact on the system's functionality. These elements could have an impact on how accurate the drowsiness detection algorithm is.

Technology And Solution Survey

The intricate interplay of multiple factors impacting driver attentiveness makes it difficult to identify tiredness in cars. Due to their potential to forecast drowsiness, machine learning (ML) models—especially those that include data sources like physiological signals, facial expressions, vehicle dynamics, and environmental conditions have gained a lot of attention recently.

Numerous investigations have resulted in significant breakthroughs in complex monitoring systems. Using a machine learning framework that combined YOLOv3 with CNN, Afsar et al. (2023) showed a noteworthy accuracy rate of 93.12% in tracking students' emotions and attention in a classroom. Lazuardi et al. (2023) developed a system for identifying driver tiredness in cars using the Jetson Nano platform and the YOLOv3 algorithm, demonstrating good performance in accurate drowsiness detection.

There have been major advances in the field of sleepiness detection from earlier research. Sur et al.(2015) employed the Viola-Jones algorithm and the Sobel operator to identify tiredness in drivers of automobiles. In order to obtain an accuracy rate of 81%, Tyagi et al. (2022) devised a real-time driver drowsiness detection system that integrated SDM and SVM algorithms. Neshov and Manolova (2017) improved the system with a virtual assistant for more attentiveness by using the Viola-Jones algorithm and Eye Aspect Ratio (EAR) for driver sleepiness detection. When Ali et al. (2021) coupled the Dlib and CNN techniques, they were able to detect driver drowsiness with an accuracy rate of almost 98%.

Recent work has focused on exploring innovative combinations of technology and algorithms. Kielty et al. (2023) demonstrated notable accuracy in their system for yawning detection and seatbelt compliance, which uses neuromorphic algorithms, CNN, and RNN. Using CNN and the MRL eye dataset, Soman et al. (2023) successfully classified the levels of driver tiredness, yielding an AUROC score of 97.88%. S et al. (2022) used OpenCV, Viola-Jones, and CNN approaches to successfully detect indicators of tiredness, emphasizing concurrent training and validation losses. In their 2020 study, Balam et al. (2021) concentrated on an electroencephalography (EEG)-based drowsiness detection system, utilizing statistical tests and Hjorth criteria for evaluation.

Data Management Plan

Data Collection Approaches

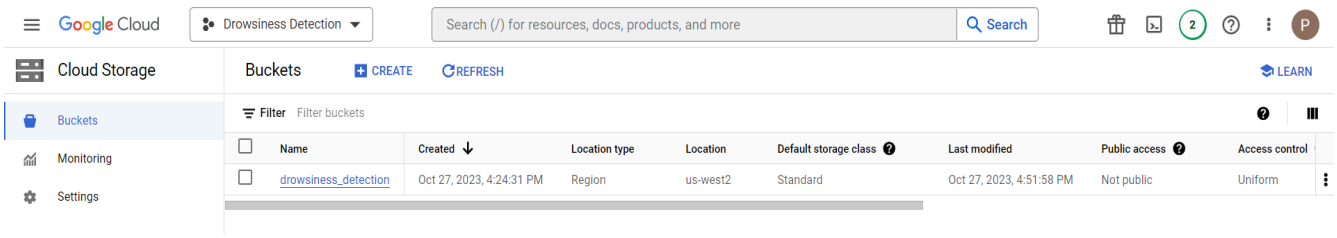
Night Time Yawning Microsleep Eyeblink Driver Distraction (NITYMED) dataset was used for the research and was utilized from the research institute Embedded System Research and Application (ESDA) lab website.This dataset, which includes 130 films, highlights the importance of identifying tiredness in nighttime driving conditions by showing actual drivers operating their vehicles. A camera mounted on the dashboard of an automobile was used to record the videos. Eleven male and ten female drivers who are actively participating in the study are among the participants, they differ in terms of things like hair color, whether or not they have facial hair, and whether or not they wear spectacles. There are two separate groups into which the videos are divided. The drivers were seen yawning three times in total, lasting around 15 to 25 seconds in each of the 107 films that were examined. In the films that were watched, drivers converse, look around them, and occasionally take brief, two-minute naps between scenes. There are 21 recordings in all that show this occurrence. Every movie in the collection has an mp4 encoded frame rate of 25 frames per second. The dataset comes in two resolution options: High-Definition Television (HDTV) 720p, which is 1280 pixels wide and 1080 pixels high. Its total size is around 4.6 gigabytes. The dataset that is offered can be useful in comparing and contrasting models and algorithms for the goal of identifying fatigue during the night. This dataset can also be used to assess other face, mouth, and eye tracking applications. With the exception of a few scenes where the internal automobile lights are dimmed, the movie lighting is mostly natural. The purpose of this change is to simulate the lighting that is normally found on an avenue. For safety reasons, it is important to note that most of the films were taken on a route with little traffic and dark lighting.

Data Management and Storage

Working with data requires managing it since a model's performance is directly impacted by the quality of the data. Data transformation techniques, such as data cleansing, were required to address issues such as missing numbers or incorrect measurements that may have resulted from faulty equipment. The data was first preprocessed and then collected using Python tools. To examine the data and obtain fresh ideas, an exploratory data analysis was then carried out. Next, the dataset was divided into testing and training subsets. Features from both of these databases have been engineered. Data handling and storage was done using the Google Cloud Platform (GCP). The ideal storage class for the bucket will vary depending on factors like the frequency of access and the speed at which data must be restored. The structure and content of the data remained unchanged since it was stored in the container in the original format from which it was extracted. Figure 1 depicts how Google Cloud Platform (GCP) buckets store data.

Figure 1

Google Cloud Platform (GCP) Bucket

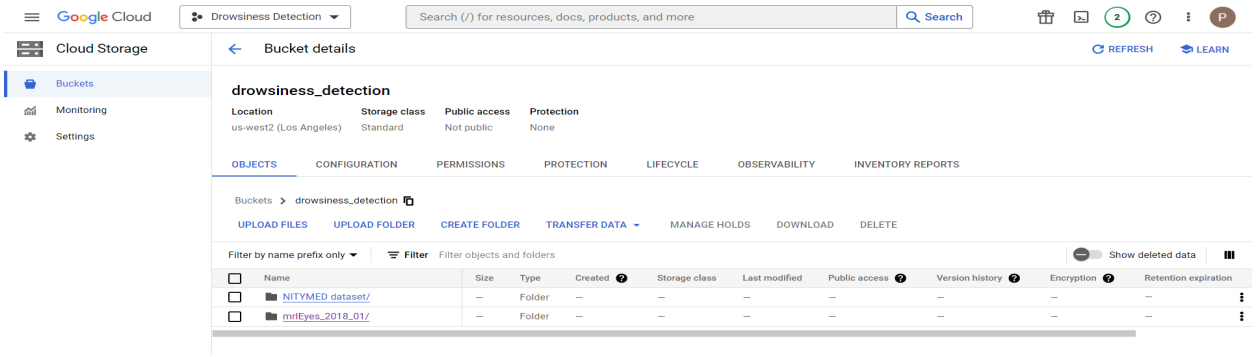


Note. This Shows The Image Of Google Cloud Platform (GCP) Bucket

The bucket is the root directory and was established for the drowsiness detection project. Other folders were built within this bucket to hold other datasets. The NITYMED dataset is the name of the first directory in the paradigm.A total of 130 movies showing drivers operating real cars at night are included in the datasets. A camera mounted on the dashboard of an automobile was used to record the videos. The root directory of GCP is displayed in Figure 2.

Figure 2

Google Cloud Platform (GCP) Root Directory

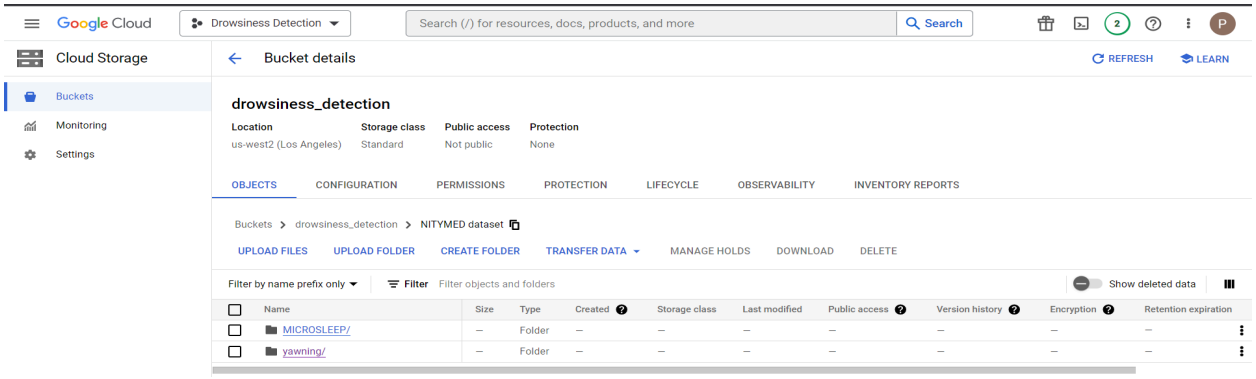


Note. This Shows The Image Of Google Cloud Platform (Gcp) Root Directory

The video files in the first folder of the NITYMED dataset were divided into two subfolders: yawning and micro sleepy. NITYMED directory on the GCP is displayed in Figure 3.

Figure 3

NITYMED *Directory In GCP*

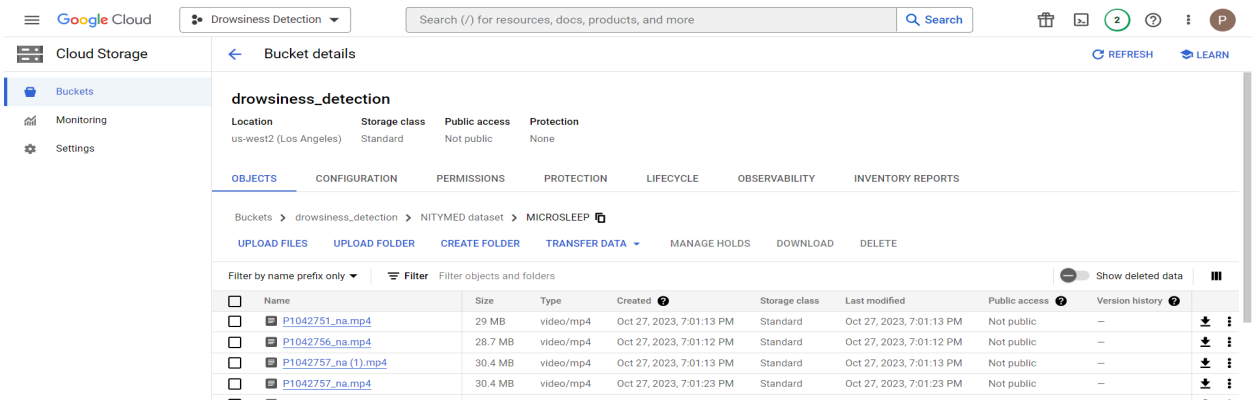


Note. This Shows The Image Of NITYMED Directory In GCP

Within the microsleep folder, the drivers converse, assess their environment visually, and go into brief, two-minute microsleep episodes. There are twenty-one recordings that document this event.The Google Cloud Platform (GCP) microsleep folder is displayed in Figure 4.

Figure 4

Google Cloud Platform (GCP) Microsleep Folder

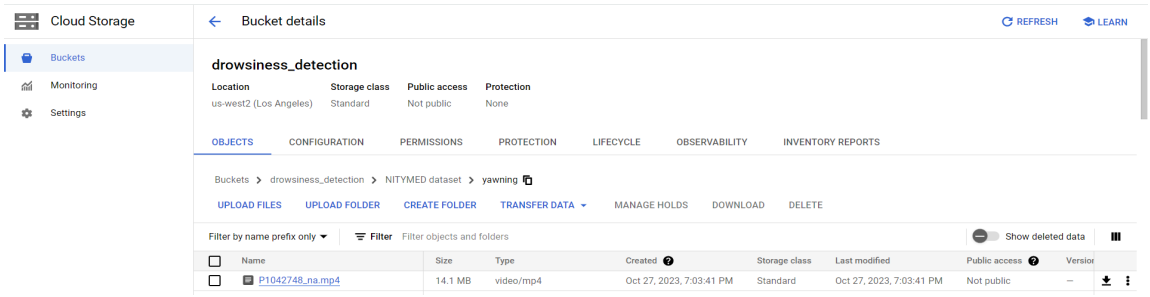


Note. This Shows The Image Of Google Cloud Platform (GCP) Microsleep Folder

There are 107 analyzed movies in the yawning folder. Three instances of yawning were noted in the drivers, with an average yawn lasting between fifteen and twenty-five seconds.The Google Cloud Platform (GCP) yawning folder is displayed in Figure 5.

Figure 5

Google Cloud Platform (GCP) Yawning Folder



Note. This Shows The Image Of Google Cloud Platform (GCP) Yawning Folder

Table 2 lists the folder and its naming convention and the purpose.Table 3 provides a thorough summary of all the files found in the specified bucket's folders, along with important information like file types and naming conventions.

Table 2

Folder And Its Naming Convention And Purpose

Name	Purpose
NITYMED dataset	Storing video files

Note. This Table Shows The Folder And Its Naming Convention And Purpose

Table 3

Data Files And Its Naming Convention

File	Naming Convention	Folder Name	File Format
Microsleep and yawning	METR_LA_<source>_<v_num>	Night Time Yawning Microsleep Eyeblink Driver Distraction(NITYMED)	MP4

Note. This Table Shows The Data Files And Its Naming Convention

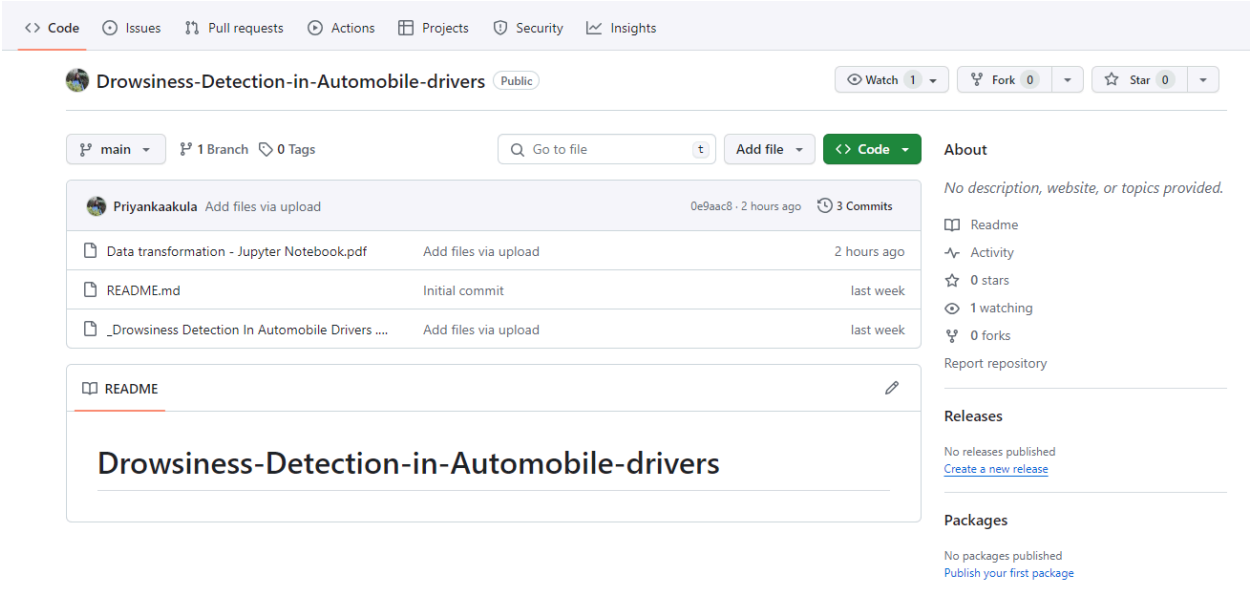
Data Usage Mechanism

Four different deep learning models will be developed as part of the project: Viola-Jones, You Look Only Once Version 3 (YOLOv3), Convolutional Neural Network (CNN), and Visual Geometry Group 19 (VGG19).These algorithms were used to evaluate the gathered information and generate accurate predictions meant to reduce traffic accidents and fatalities brought on by fatigued drivers. Documentation must be produced to guarantee that other readers and users have a complete knowledge of the data and the procedures used to obtain it. This guide will be written and stored in a GitHub repository as a ReadMe file. The complete process of gathering data from several sources, including the particular format and organization of the data, was fully documented. Together with the code implementation, it was also to describe the planned uses of the data and include details on the libraries used for data preprocessing. The documentation will

also include links to pertinent references on this subject. Future project updates will define the archiving and retention policies that apply to the files that have been uploaded and the buckets that have been created on the Google Cloud Platform (GCP). The GitHub repository will be open to the public to enable further research and exploration of this topic. In the process of implementing a Data Management Plan (DMP), all team members share an equal amount of tasks. Projects employ a range of storage systems because the volume of data needed varies and must be scaled up or down based on project requirements. The project's data, codes, and required documentation are uploaded to a private GitHub repository because only a fraction of the data is used. GitHub repositories will also be used by the project as its version control mechanism. The other team members are added as collaborators, and one team member is named as the repository's owner. The path for the GitHub repository can be seen in Figure 6.

Figure 6

Github Repository



Note. The Image Shows The Github Repository

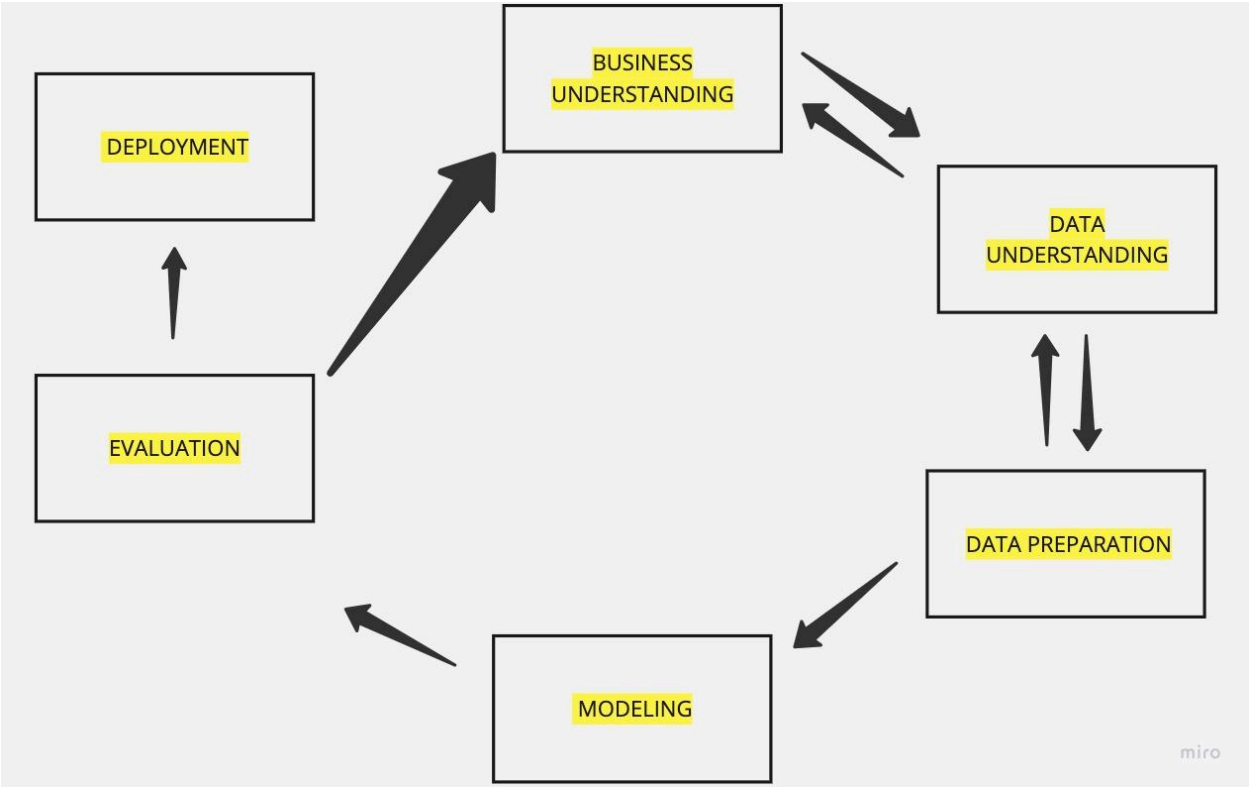
Project Development Methodology

Cross-Industry Standard Process For Data Mining

The abbreviation for the Cross-Industry Standard Process for Data Mining is CRISP-DM.This project is being developed using the CRISP-DM approach. This approach offers a methodical and adaptable way to complete the research. The process is divided into six stages: deployment, evaluation, modeling, data preparation, business understanding, and data understanding. The CRISP-DM methodology is shown in Figure 7

Figure 7

CRISP-DM Methodology



Note. This Shows The Image Of Crisp-Dm Methodology

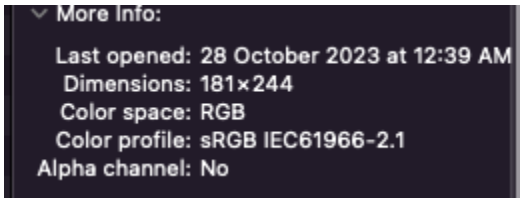
Business Understanding. This phase is the first step of the six phases of the CRISP-DM methodology.It entails identifying the project's goals, evaluating the environment, deciding on

the data mining aim, and creating a project strategy. The goal is to reduce the amount of fatalities and collisions that result from drowsy drivers on the road. The purpose of this project is to develop a detection model that can identify driver fatigue in order to stop these kinds of accidents. The plan entails specifying the objectives, due dates, and distribution of resources. The information was gathered from multiple sources. The model's accuracy and false alerts are among the project's hazards. This stage defines the problem of sleepy driving, establishes milestones, and creates a map for applying algorithms, laying the groundwork for solving it.

Data Understanding. Understanding the dataset and the crucial characteristics needed by the algorithms used to identify sleepiness is crucial at this stage. Initiating data collection, data description, data exploration, and data quality verification were all included in this step. The NITYMED dataset was gathered and contains information on 11 men and 10 women with different haircuts, beards, and eyeglasses. The frame-level collection contains 130 movies and 53331 pictures. The movies have footage of yawning, microsleeping, and attentive drivers. This dataset contains movies, whereas the frame-level dataset contains both images and videos. Figure 8 shows the image information of the dataset.

Figure 8

Information Of An Image In The NITYMED Dataset



Note. Information Of An Image In The *NITYMED* Dataset

Data preparation. Frames from video files were retrieved using the OpenCV library. These frames were stored in a directory as pictures. Algorithms such as YOLOv3 operate on

images with the same resolution. Each picture was resized to fit that specific version of You Only Look Once (YOLO). Median filtering or Gaussian blur was used to lessen the noise in the image. The duplicates were eliminated based on their confidence scores in order to get rid of the overlapping bounding boxes.The images needed were scaled for the Viola-Jones algorithm so that they fit the Viola-Jones classifier. Both the labels and the input photos were augmented. CNN architecture was utilized for facial feature identification and eye alignment. By making sure that the most confident detections and the non-overlapping detections were not eliminated and confidence scores were utilized to eliminate duplicates.

Modeling. Choosing modeling methodologies, creating test designs, creating models, and evaluating the models were all part of this process. Prediction was done using the Viola Jones, YOLOv3, CNN, and VGG19 algorithms. A training set, a testing set, and a validation set comprise the NITYMED dataset. 80:10:10 was the ratio of training to validation to testing.The mentioned predictive algorithms were employed and the objective of this study is to identify the most accurate algorithm for sleepiness detection in drivers. The training set was used to apply and train the algorithms. After training, the parameters were adjusted to achieve the best fit for the training dataset while minimizing error.The model was assessed using the test dataset. An assessment tool was the performance matrix. The model's accuracy, precision, recall, loss, and F-1 scores were used to assess it.

Evaluation. Assessing the outcomes was the next stage following the creation and assessment of the models. These models underwent a thorough examination to determine their efficacy based on insights, patterns, and accuracy. The effectiveness was evaluated using the following metrics: recall, accuracy, precision, and F-1 score. These measurements guaranteed that the models satisfy the needs of the project.

Deployment. The CRISP-DM technique ends with this step. This entails deploying the models in order to sustain them throughout time. The deployment environment had been determined, and real-time integration while maintaining attention to the schedule and objectives. An integration plan was developed and the best GCP deployment environment was chosen. Following its deployment, key performance indicators (KPIs) were used to track the models performance. Following deployment, a final report was created that included the project's goals and different stages.

Project Organization Plan

Work Breakdown Structure

The Work Breakdown Structure (WBS) divides large projects into manageable chunks. It serves as a roadmap for the team through each stage of data mining, from conceptualizing the business to actually executing it.The WBS aids in identifying any potential problems that might develop throughout the project. The CRISP-DM method was used in this project. The first step was to determine which CRISP-DM phases were important.The business understanding, data understanding, data preparation, modeling, assessment, and deployment are the primary steps in the WBS process. Each subphase was further broken down into smaller tasks that were assigned to team members in order to guarantee the project's completion.

The first stage of the Work Breakdown Structure (WBS) is Business Understanding, which establishes the project's scope. This entails figuring out the problem description and the project aim. Following the study of research articles, four deep learning models were chosen for the undertaking.

The NITYMED dataset was selected for the study.This dataset includes 130 real-world videos of drivers operating a vehicle at night, when it's critical to detect tiredness. Python data

exploration is used to clean up and eliminate outliers, missing values, and other anomalies from the raw data. Prior to the following phase, which was mostly concerned with data preparation, the quality of the data is being verified.

When working with complicated datasets, data preparation was an essential stage in the analysis and modeling process. One of the crucial jobs in the data preparation stage was feature extraction for sleepiness detection. This entails extracting pertinent characteristics from the dataset in order to assess a person's level of sleepiness. Data wrangling comes next after these features are extracted. Python packages were utilized to modify the datasets because they contained images and video. The video datasets were transformed into frames throughout this process. The prediction was based on the patterns found in the features that have been detected, which will help with the precise identification of drowsiness in people.

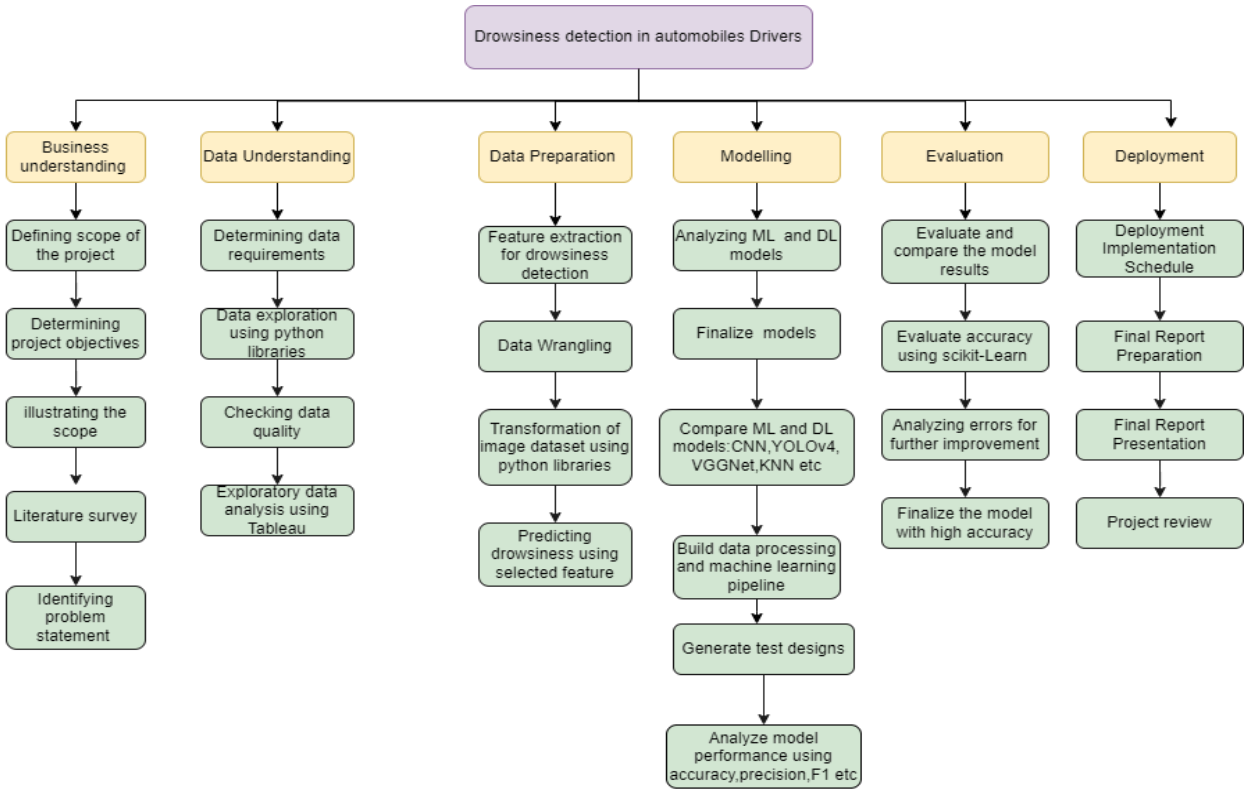
Various deep learning models were analyzed during the project's modeling phase. Examining the four models' performances will be the goal, these models were selected following research conducted during the project's initial stage: Visual Geometry Group 19 (VGG19), You Only Look Once version 3 (YOLOv3), Viola Jones, and Convolutional Neural Network(CNN). The goal at this point was to assess each model's accuracy in identifying the target variables, after which significant adjustments were made to increase accuracy. The evaluation measures employed for the models created in the subsequent phase were F1 score, accuracy, precision, recall, and loss.

Using Scikit-Learn, models were compared and their accuracy assessed during the evaluation phase. In order to continue improving, this also entails identifying mistakes and fixing them. The model with the best accuracy was chosen once the performance of each model was

evaluated. The selected models were implemented and observed. The project's work breakdown structure, which is based on the CRISP-DM methodology, is displayed in Figure 9.

Figure 9

Work Breakdown Structure (WBS)



Note. This Shows the Image of Work Breakdown Structure (WBS)

Project Resource Requirements And Plan

The resources needed to complete a project and their allocation are covered in this section. To ensure that the project will be finished on schedule and to centralize the budget, this was an essential component of project management.

Hardware Requirements

Hardware requirements is the study that pertains to the Drowsiness Detection in Drivers project, which uses technology. Throughout the research, the team members utilized MacBook,

Dell computer systems. Table 4 below lists the CPU, GPU, RAM, and storage capacity of the computer system.

Table 4

Hardware Requirements

System	System Name	CPU	GPU	RAM
MacBook	MacBook Pro	Apple M2 Pro chip 12-core CPU with 8 performance cores and 4 efficiency cores	48- or 64-core GPU with up to 8192 ALUs and 21 TFLOPs of FP32 performance.	16GB of RAM as standard, but you can upgrade to 32GB
Dell	Inspiron 15	Intel i7 13th Gen 4.5 Ghz	NVIDIA 3050Ti	64GB

Note. This Table Shows The Hardware Requirements

Tools and Licenses

Numerous programming languages and tools are needed for the creation of a sleepiness detection model. Verifying whether licenses are necessary for each tool used in the project is crucial. The list of tools being utilized for the project is displayed in Table 5 .

Table 5

Tools And Licenses For The Project

Tools	Purpose	License
Jupyter Notebook	For Python Code	Open Source
JIRA	Project management tool	Student
Draw.io	WBS,Pert chart,Figures	Student

Tools And Licenses For The Project

Tools	Purpose	License
Microsoft Office 360	Word,Excel	Student
Google Docs	Documentation	Free
Google Cloud Platform (GCP)	Data Storage	Student
Github	Managing projects	Free
Google Colab	For Python Code	Free
Zoom	Project meetings	Free

Note. This Table Shows The Tools And Licenses For The Project

Project Resources Cost And Justification

This includes the entire budget required for the project's effective completion as well as the rationale behind it. The remaining services that were utilized are shown in Table 6 , and the hardware equipment for the research.

Table 6

Project Resources Cost And Justification

Resources	Justification	Duration (Months)	Cost
Google Cloud Platform	For deployment and ETL	3	$0
Jira Software	Creation of Gantt Chart, WBS and Project Management	4	$0

Note. This Table Shows The Project Resources Cost And Justification

Project Schedule

Project management tools such as the Gantt chart and PERT chart are used to depict the project schedule.

Gantt Chart

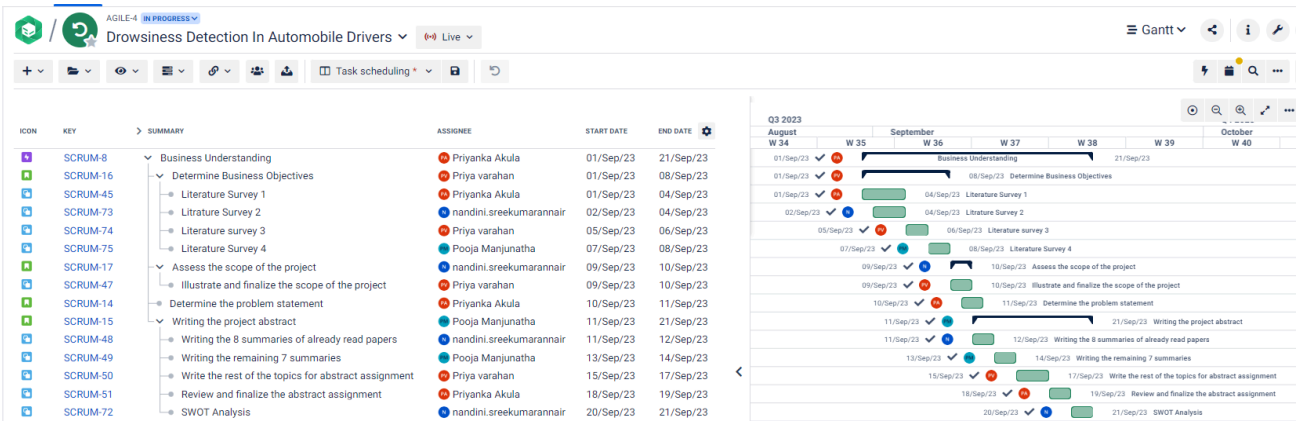
A Gantt chart is a visual aid for project management that shows the deadlines for each job and subtasks within the project. In the project, the Gantt chart is created using Jira Software, and the subjects of CRISP DM and Exploratory Data Analysis (EDA) are used to define the epics.

The epics are broken down into tasks, which are then broken down into smaller jobs. Subtasks are split up so that each one takes one to two days to complete. Once more, one worker is allocated to each subtask. The dependencies between jobs are also depicted in the chart. By using this representation, risk can be reduced while also enhancing communication, scope definition, and time management.

Business Understanding. As part of the CRISP-DM framework, the project's early stages were centered on identifying its objectives. There were other assignments and smaller jobs in this phase, which was called Business Understanding. Participants were given the subtask Literature Survey as part of the six-day Understanding Business Objectives main task. The one-day job Assess the Scope of the Project, which included the subtask Illustrate and Finalize the Scope of the Project, was another essential component. Concurrently, after evaluating the project's scope, the task of Determining the Problem Statement was started in order to determine the problem statement. Writing the Project Abstract was the last assignment for this phase. It was divided into five smaller projects and finished in nine days. Every work and subtask in the Business Understanding phase had precise deadlines and assignees listed.For every job and subtask in Business Understanding, deadlines and task assignees are shown in Figure 10.

Figure 10

Business Understanding Phase

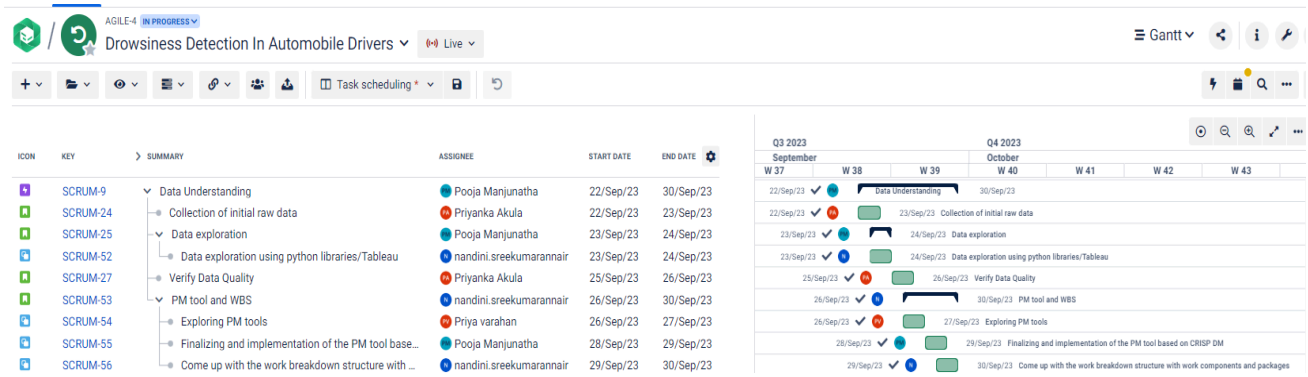


Note. This Shows The Image Of Business Understanding Phase

Data Understanding.A critical initial stage in the CRISP DM process was determining the project's objective. Several tasks and subtasks were part of this phase, which was called Business Understanding. A subtask called Literature Survey was given to each participant as part of the six-day Understanding Business Objectives task. The next assignment was to assess the project's scope, which took one day and included the subtask of illustrating and finalizing the project's scope. The work of determining the problem statement was started once the project's scope had been assessed in order to ascertain the problem statement. Writing the Project Abstract, which was broken down into five smaller assignments and finished over the course of nine days, was the phase's last assignment. Each Business Understanding task and its assignees are listed in detail in a Figure 11 along with their deadlines.

Figure 11

Data Understanding Phase

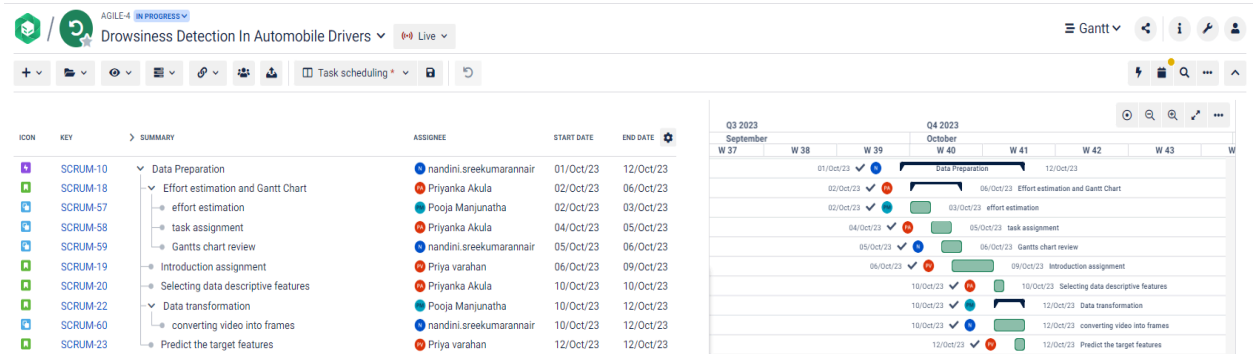


Note. This Shows The Image Of Data Understanding Phase

Data Preparation. The third CRISP DM step, data preparation, was an important part of the project. Effort estimation and Gantt charts, two essential components of the epic Data Preparation, were the first tasks to be completed. The subtasks of task assignment, Gantt chart review, and effort estimation were assigned to specific personnel. These small jobs took two days to do each. One of the team members was given the Introduction Assignment. Data Transformation was another important task that took three days to complete and entailed transforming video data into frames. Another team member was tasked with this assignment, which required one day of work. Figure 12 provides a clear outline of the epic Data Preparation tasks, subtasks, accompanying deadlines, and assignees.Figure A1 shows the detailed Gantt Chart For Business Understanding To Data Transformation.

Figure 12

Data Preparation Phase

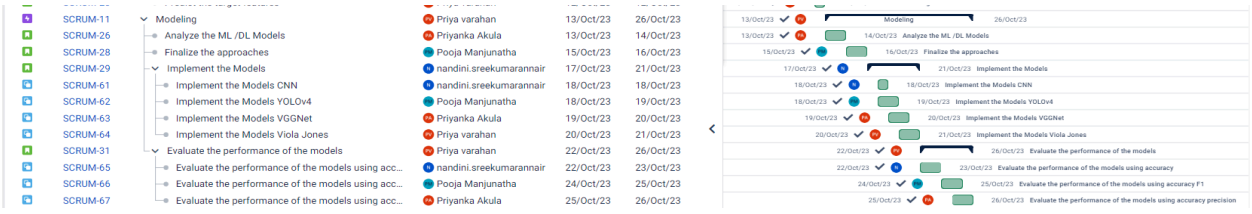


Note. This Shows The Image Of Data Preparation Phase

Modeling. Modeling, the fourth and most important step in the CRISP DM process, took about ten days to complete. The four primary responsibilities of this stage were to study the DL Models, complete the DL Approaches, apply the Model, and evaluate Performance. Analyzing the DL Models required evaluating the models' applicability in forecasting drivers' fatigue levels based on the given information. I finished this assignment in a single day. Comparably, the work of finalizing the DL approaches needed data models to be finished before the project could be carried out, and this was likewise completed in a single day. The model was then collaboratively implemented and evaluated over the course of eight days by all of the participants. Figure 13 provides exact documentation of these tasks, showing their development and conclusion.

Figure 13

Modeling Phase

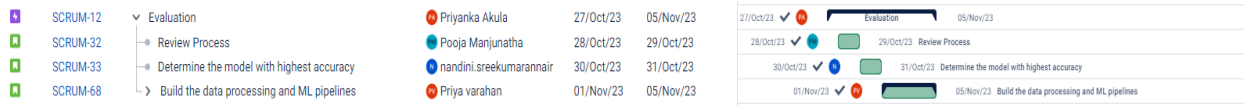


Note. This Shows The Image Of Modeling Phase

Evaluation. The assessment phase of project management took place over the course of several days. A six-day program was part of this phase, and among the tasks were creating data processing and machine learning pipelines, identifying the model with the highest accuracy, and reviewing the process. In a single day, all of the tasks Review Process and Determine the Model with the Highest Accuracy were finished. Figure 14 depicts the specifics of these duties, their development, and their conclusion.

Figure 14

Evaluation Phase

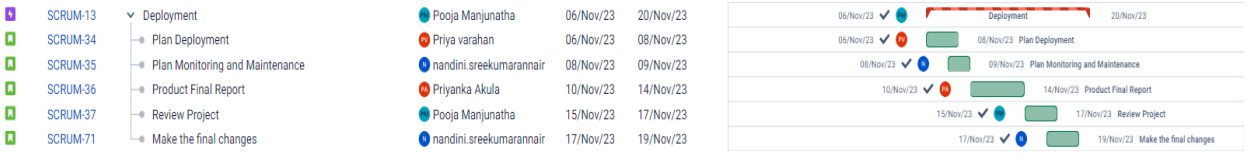


Note. This Shows The Image Of Evaluation Phase

Deployment .The epic Deployment lasted 11 days and concluded the project. This phase featured Plan Monitoring and Maintenance and Plan Deployment, which began three days in. It was important to make final changes to dependencies, timetables, and statuses. These activities were crucial to the project's operational stability and real-world readiness. The dependencies and status of these tasks is shown in Figure 15. Figure A2 shows the detailed Gantt chart for Modeling to Deployment.

Figure 15

Deployment Phase



Note. This shows the image of Deployment Phase

Pert Chart

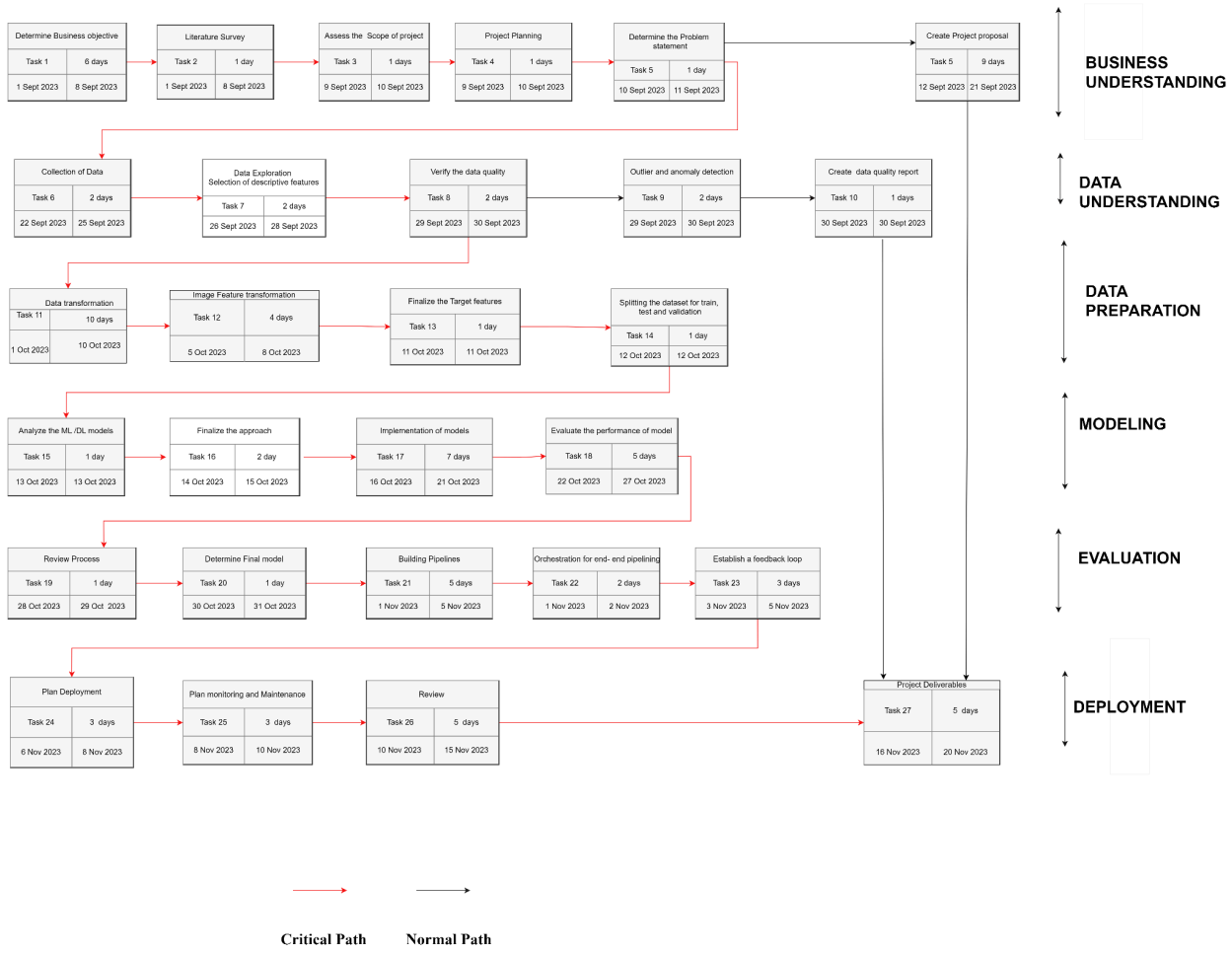
Gantt charts and other project development tools serve a variety of visualization purposes.Program Evaluation Review Technique(PERT) charts, or program evaluation review techniques, are a type of network diagram where the linkages and dependencies between the activities are highlighted. The sequence of tasks timeline aids in identifying the project's key path. For this project, creating the PERT chart entails breaking down the tasks, determining dependencies, calculating time, and determining the critical route.

Critical path. By determining the tasks that must be accomplished in the order shown in Figure in order for the project to be successfully finished, the PERT chart is completed. The PERT chart has been categorized into six stages, including business knowledge, data preparation, deployment, modeling, assessment, and evaluation. The project claims that the CRISP-DM approach is being applied.

A critical path, usually red in diagrams, represents hierarchically separate tasks necessary for project completion. Understanding company objectives and identifying the problem statement are critical at the business understanding stage of a project. While vital, project proposal creation is not critical because the project can progress without it. In the data comprehension stage, outlier and anomaly detection and data quality reports are useful but not crucial to project success. This stage requires raw data collection, descriptive feature selection, and data quality verification. Instead, a normal path, usually black, has tasks with no dependencies. These tasks' delays do not influence other jobs or subtasks or the project's completion. Figure 16 shows the Pert chart for the project.

Figure 16

Pert Chart



Note. This Shows The Image Of Pert Chart For The Project

Data Engineering

Data Process

Many and different attempts have been made using deep learning to detect instances of tiredness in drivers of automobiles using data from the Night Time Yawning Microsleep Eyeblink Distraction platform (Petrellis et al., 2021).

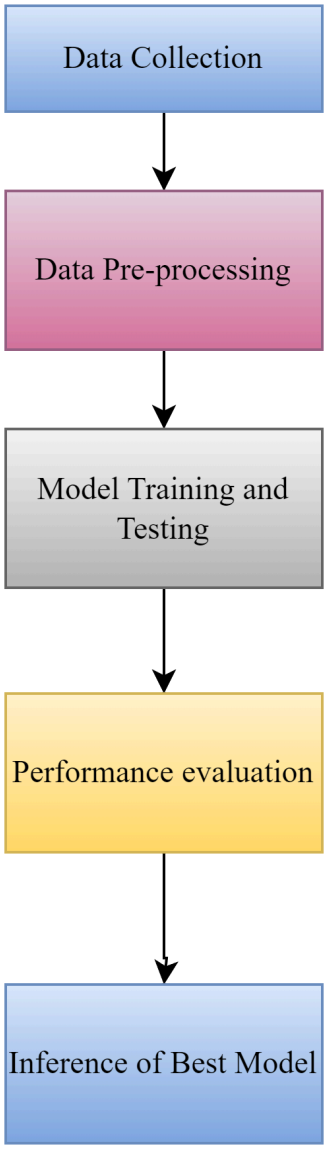
Figure 17 illustrates the process flow, which delineates each step of the sleepiness detection project on photographs utilizing several deep-learning models. The principal prerequisite for employing deep learning models in data analysis is the availability of a substantial quantity of data. The information came from a lab dedicated to media research and an Electronics Systems Development and Application (ESDA) lab. The final compilation includes 130 footage of actual drivers operating a vehicle at night, when it is crucial to recognize signs of fatigue. The videos were recorded by the dashboard-mounted camera. The drivers, ten women and eleven males, have a variety of facial hair styles, beards, and eyeglasses. The videos are split into two categories: microsleep drivers, which talk, look around, and sleep for two minutes (21 movies), and drivers who yawn three times every fifteen to twenty-five seconds (107 videos).

A gigabit of sample data, namely 53,333 photos related to yawning, microsleep, and alert, is generated in order to finish the project. The annotations provided for photos are stored in JavaScript Object Notation (JSON) files.

The next step is pre-processing of the data, wherein Exploratory Data Analysis (EDA) is performed. Data anomalies, such as outliers, inconsistent data, noisy data, and other comparable irregularities, are found and dealt with using this method. The dataset is cleaned and transformed following Exploratory Data Analysis (EDA) in order to guarantee that the model is provided with accurate and dependable data.

Figure 17

Workflow Diagram



Note. Workflow Of Drowsiness Detection And Image Classification Procedures

Numerous data problems are handled throughout the data cleaning process, including noisy, inconsistent, missing, and incomplete data. Errors that happened during the Application programming interface (API) data collecting phase or during the aggregation of data from several sources may be the cause of these problems.

The acquired images from the films show differences in shape and resolution, which could affect how well the model works. As such, scaling becomes very important during the pre-processing phase. To improve overall performance and speed up model training, the dataset is shrunk to the ideal size. Picture transformation is the next important step in the pre-processing of data. Here, methods including data smoothing, normalization, regularization, and dimension reduction are used to improve the model's performance as discussed by (Hemalatha & Karthick, n.d.). To reduce noise in photos, data smoothing techniques like averaging, median filtering, bilateral filtering, and Gaussian filtering are frequently used. However, the range of pixel intensity values can be adjusted using the normalization technique. In image analysis, data regularization techniques are used to reduce the problem of overfitting. These strategies cover a variety of techniques, such as dropout and data augmentation. When the number of images that need to be processed increases, the processing speed and system efficiency are affected. For this reason, it is best to reduce the image size.

The next important step is to prepare the data for model generation when pre-processing chores like data transformation and cleaning are finished. It is standard procedure to reserve 20% of the available data for model testing and 80% of the data for training in order to get the best possible model performance.

Once more, the dataset is divided into two parts: the first is set aside for training, while the second is set aside for validation. This division occurs in a 3:1 ratio. As a result, the final dataset is divided into three different subsets for analysis and assessment: training, validation, and testing. The values are distributed as follows: 80%, 10%, and 10%.

Data Collection

The Night Time Yawning Microsleep Eyeblink Driver Distraction dataset was provided by the research institute Electronics Systems Development and Application lab. Since the creator has granted permission for its use, the dataset is publicly available and available to the entire team. This dataset's main goal is to pinpoint instances of drowsiness in these particular situations. A camera mounted on the dashboard of a car was used to record the videos. The sample for the study is made up of 10 female and 11 male drivers, all of whom have noticeable characteristics such as wearing eyeglasses, having facial hair, and having hair color. Two main categories are used to group the videos. It was found that the drivers exhibited yawning behavior three times during the course of the analysis on a sample of 107 films. Every time a yawn occurred, it lasted anywhere from 15 to 25 seconds. The films that are being examined show drivers conversing, scanning their environment, and going through brief, two-minute microsleep episodes. A collection of 21 recordings in the dataset has revealed this pattern. The movies in the collection have been compressed into mp4 files, which have a 5 frames per second frame rate. It's also critical to recognize that there are no audio components. A comprehensive approach to data collection is shown in Figure 18. To sum up, the dataset that is provided here is a useful tool for analyzing and forecasting driver fatigue.

Figure 18

Data Collection Plan

Data Collection Plan							
Project number:	Group 1	Project title:	Drowsiness Detection In Automobile D	Project leader:	Priyanka Akula	Date:	11/03/2023
Description of the data collection							
	Data was taken from ESDA lab and updated in 2021. This dataset contains mp4 car dash cam videos from different genders driving at night. These videos show yawning and microsleep. For dataset collection, we downloaded full-version videos of both categories and stored them in GCP and our local to transform them for training and testing models.						
What will be done with the data once it has been collected?							
	yes	Identify how well the process is meeting customer needs		yes	Analyze a problem, or identify the causes of variation		
	yes	Obtain exploratory view of the process		yes	Test a hypothesis about the process output		
	yes	Evaluate the measurement system		yes	Test a hypothesis about the effects of one or more inputs		
	yes	Check the stability of the process (is it in control?)		yes	**Control a process input or monitor a process output**		
	yes	Conduct a capability study			**Other reason…**		
Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)							
			1	**2**	**3**		
What?	**Variable title**		Microsleep	Yawning	Drowsiness detection		
	Input (X) or output (Y) var		X	X			
	Unit of measurement		resolution	resolution			
	Data type		mp4	mp4			
	Collection method		Manual	Manual			
	If manual		The NITYMED dataset is downloaded from the original path by manual from the ESDA lab. This information is open to the public on the website and can be downloaded by filling out a request form. A set of 130 movies was collected of real drivers moving their cars at night, when detecting drowsiness is more important. There are 4.12 GB of movie files on it, which were collected over two days.				
MSA	Gauge/instrument		N/A	N/A	N/A		
	Location		N/A	N/A	N/A		
	Gauge calibrated?		N/A	N/A	N/A		
	Measurement system checke		N/A	N/A	N/A		
	Precision (R&R) adequate?		N/A	N/A	N/A		
	Accuracy adequate?		N/A	N/A	N/A		
Historical data	Historical data exist?		N/A	N/A	N/A		
	Source of historical data		N/A	N/A	N/A		
	Historical data representative		N/A	N/A	N/A		
	Mean and Standar deviation		N/A	N/A	N/A		
	Upper and lower specificatio		N/A	N/A	N/A		
	Target		N/A	N/A	N/A		
Sampling	Minimum sample size (MSS)		N/A	N/A	N/A		
	Sampling frequency		N/A	N/A	N/A		
	Sub-grouping needed?		N/A	N/A	N/A		
	Sub-group size		N/A	N/A	N/A		
	Stratification needed?		N/A	N/A	N/A		
Who?	**Data collector**		Priyanka Akula	Priyanka Akula	Priyanka Akula		
	Operational definition exist? Data collector trained? Resources available for data collector?		There is no operational definition for the gathered to make sure it is consistent, and it doesn't need to be documented as meta data. To understand the facts, you don't need to know anything about the domain.				
When?	**Start date**		26-Sep	26-Sep	26-Sep		
	Due date		28-Sep	28-Sep	28-Sep		
	Duration (in days)		2	2	2		

Note. The Names Of The Attributes, Data Sources, And Project Implementation Dates For The Data Collecting Plan Are Shown In The Table Above.

Dataset Samples. The videos that make up Night Time Yawning Microsleep Eyeblink Driver Distraction dataset are gathered here. The microsleep directory and the yawning directory

are the two subdirectories that were previously created out of the frames directory. The MP4 file format is used to encode each and every video. Figure 19 shows a screenshot of a video file from the Microsleep Directory.As seen in Figure 20 , the directory has a total of 23 video files demonstrated in Figure B1.

Figure 19

Sample Of Raw Dataset From The Microsleep Directory

Note. The Figure From The Raw Data, Which Is A Video File From A Microsleep Directory.

Figure 20

Number Of Video Files In The Microsleep Directory

```
# Print the count of video files
print(f"Number of video files in the microsleep directory: {len(video_files)}")
```

```
Number of video files in the microsleep directory: 23
```

Note. The Number Of Videos In The Microsleep Directory Is Displayed.

After analyzing the "Microsleep" directory inside the Night Time Yawning Microsleep Eyeblink Driver Distraction dataset, some significant conclusions regarding the film's temporal

properties have been drawn. The properties provide a wide range of information, including the title, frame count, resolution, and duration of the video. Figure 21 shows the sample properties and is demonstrated in Figure B2.

Figure 21

Microsleep Directory Sample

```
Video: P1042792_na.mp4
Frame Count: 3135
Frame Rate: 25
Resolution: 1920x1080
Duration: 125.4 seconds
Video: P1042757_na.mp4
Frame Count: 3640
Frame Rate: 25
Resolution: 1920x1080
Duration: 145.6 seconds
Video: P1043068_na (1).mp4
Frame Count: 3665
Frame Rate: 25
```

Note. A Thorough Sample Of Microsleep Data Is Displayed.

P1043068A_na(1) has the highest frame count out of 3655.Among 3055 MP4, P1043155 has the lowest rating as shown in Figure 22 and is demonstrated in Figure B3.

Figure 22

Highest and Lowest Frame Count in the Microsleep Directory

```
# Print the video with the highest and lowest frame counts
print(f"Video with the highest frame count: {video_with_highest_frame_count}")
print(f"Highest frame count: {highest_frame_count}")
print(f"Video with the lowest frame count: {video_with_lowest_frame_count}")
print(f"Lowest frame count: {lowest_frame_count}")
```

```
Video with the highest frame count: P1043068_na (1).mp4
Highest frame count: 3665
Video with the lowest frame count: P1043115_na.mp4
Lowest frame count: 3055
```

Note. Frame Count Details

The resolution of each movie is 1920 x 1080, matching the resolution of the video files found in the microsleep subfolder demonstrated in Figure B4. The video files in this directory are smaller than the ones in the microsleep directory. The video files in this directory have an average length of 130.16 seconds as shown in Figure 23 and is demonstrated in Figure B5.

Figure 23

Highest, Shortest And Average Duration Of Video Files In The Microsleep Directory

```
Highest duration: 146.60 seconds
Shortest duration: 122.20 seconds
Average duration: 130.16 seconds
```

Note. The Above Figure Gives The Length Of The Video Files.

Figure 24 shows a sample of the properties of videos present in the yawning directory demonstrated in Figure B6 and Figure 25 shows a screenshot of a video file from the same directory. This directory contains 95 files as shown in Figure 26 which is demonstrated in Figure B7. The number of frames in videos varies. With 418 frames, P1043120_na.mp4 has the fewest, and with 1490 frames, P1042778_na.mp4 has the most as shown in Figure 27 demonstrated in B8.

Figure 24

Yawning Directory Sample

```
Video: P1043090_na.mp4
Frame Count: 830
Frame Rate: 25
Resolution: 1920x1080
Duration: 33.2 seconds
Video: P1042773_na.mp4
Frame Count: 990
Frame Rate: 25
Resolution: 1920x1080
Duration: 39.6 seconds
```

Note. Properties Of Videos From A Video File in Yawning Directory.

Figure 25

Sample Frame From The Yawning Directory

Note. Screenshot From The Raw Data, Which Is A Video File Of Yawning Directory.

Figure 26

Number Of Video Files In The Yawning Directory

```
print(f"Number of video files in the yawning directory: {len(video_files)}")
```

```
Number of video files in the yawning directory: 95
```

*Note.*The Count Of Videos In The Yawning Directory.

Figure 27

Number Of Frames In The Yawning Directory

```
Video with the highest frame count: P1042778_na.mp4
Highest frame count: 1490
Video with the lowest frame count: P1043120_na.mp4
Lowest frame count: 418
```

Note. The Maximum And Lowest Frame Counts In The Yawning Directory.

The resolution of each movie is 1920 x 1080, matching the resolution of the video files found in the microsleep subfolder demonstrated in Figure B9. The video files in this directory are smaller than the ones in the microsleep directory. As shown in Figure 28, the video files in the

directory have an average duration of 31.12 seconds, with the longest being 59.60 seconds and the smallest being 16.72 seconds demonstrated in Figure B10.

Figure 28

Highest, Shortest And Average Duration Of Video Files In The Yawning Directory

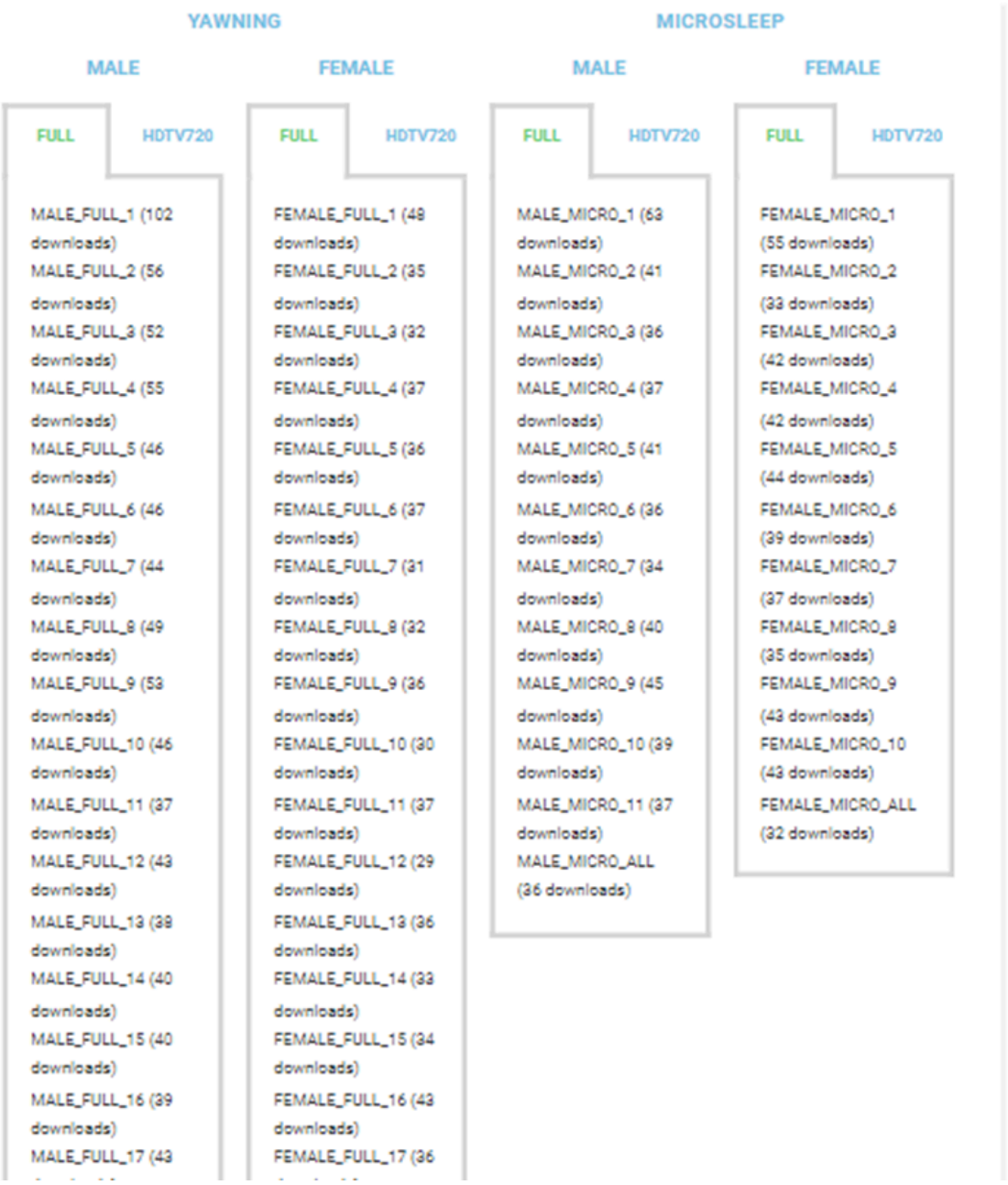
```
Highest duration: 59.60 seconds
Shortest duration: 16.72 seconds
Average duration: 31.12 seconds
```

*Note.*The Length Of The Yawning Video Files.

Synthetic Data Samples. The process of creating fake data using computer modeling and algorithms so that the final product has the same statistical properties as the original dataset is known as "synthetic data production." This procedure is crucial to the production of the data. There are several justifications for creating synthetic data, including privacy protection, cost reduction, increasing the size of the dataset, and enabling edge case analysis. The study for the research is done using the 130-video Night Time Yawning Microsleep Eyeblink Driver Distraction dataset. Two separate groups are created from the footage. The behavior of the drivers was seen to include occurrences of yawning in a sample of 107 movies, and symptoms of microsleep in 21 video samples. The Night Time Yawning Microsleep Eyeblink Driver Distraction video dataset is represented visually in Figure 29.

Figure 29

Night Time Yawning Microsleep Eyeblink Driver Distraction Video Dataset



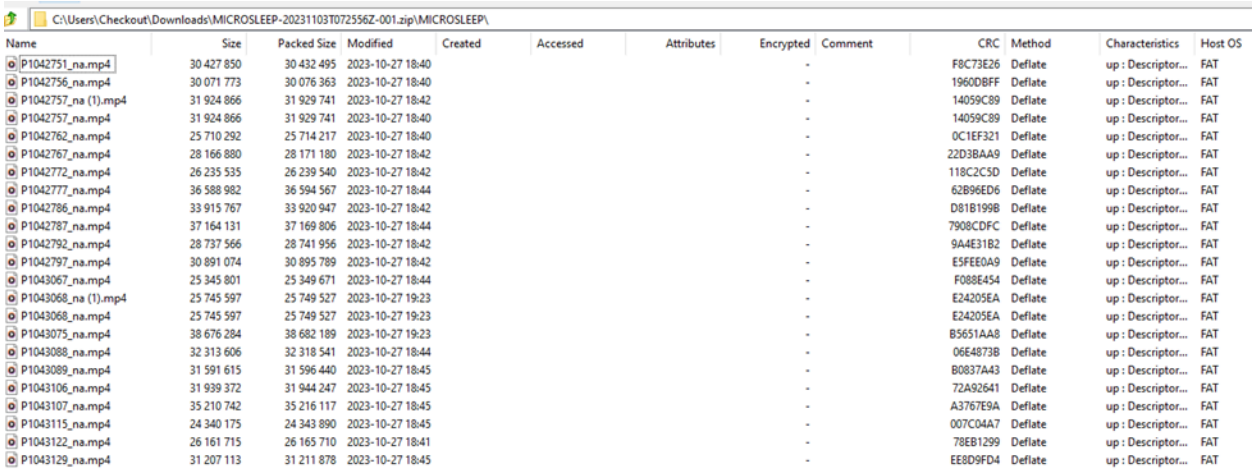
Note. The Complete Raw Dataset From The Source Is Displayed.

The files were obtained and moved to a local system for further analysis. The movies were sorted and kept in multiple directories; Yawning was in a separate folder, and Microsleep

had its own folder. The unique directory on the local system where the videos are stored is shown in Figure 30.

Figure 30

Separate Video Directory In Local System



C:\Users\Checkout\Downloads\MICROSLEEP-20231103T072556Z-001.zip\MICROSLEEP\

Name	Size	Packed Size	Modified	Created	Accessed	Attributes	Encrypted	Comment	CRC	Method	Characteristics	Host OS
P1042751_na.mp4	30 427 850	30 432 495	2023-10-27 18:40				-		F8C73E26	Deflate	up : Descriptor...	FAT
P1042756_na.mp4	30 071 773	30 076 363	2023-10-27 18:40				-		1960DBFF	Deflate	up : Descriptor...	FAT
P1042757_na (1).mp4	31 924 866	31 929 741	2023-10-27 18:42				-		14059C89	Deflate	up : Descriptor...	FAT
P1042757_na.mp4	31 924 866	31 929 741	2023-10-27 18:40				-		14059C89	Deflate	up : Descriptor...	FAT
P1042762_na.mp4	25 710 292	25 714 217	2023-10-27 18:40				-		0C1EF321	Deflate	up : Descriptor...	FAT
P1042767_na.mp4	28 166 880	28 171 180	2023-10-27 18:42				-		22D3BAA9	Deflate	up : Descriptor...	FAT
P1042772_na.mp4	26 235 535	26 239 540	2023-10-27 18:42				-		118C2C5D	Deflate	up : Descriptor...	FAT
P1042777_na.mp4	36 588 982	36 594 567	2023-10-27 18:44				-		62B96ED6	Deflate	up : Descriptor...	FAT
P1042786_na.mp4	33 915 767	33 920 947	2023-10-27 18:42				-		D81B199B	Deflate	up : Descriptor...	FAT
P1042787_na.mp4	37 164 131	37 169 806	2023-10-27 18:44				-		7908CDFC	Deflate	up : Descriptor...	FAT
P1042792_na.mp4	28 737 566	28 741 956	2023-10-27 18:42				-		9A4E31B2	Deflate	up : Descriptor...	FAT
P1042797_na.mp4	30 891 074	30 895 789	2023-10-27 18:42				-		E5FEE0A9	Deflate	up : Descriptor...	FAT
P1043067_na.mp4	25 345 801	25 349 671	2023-10-27 18:44				-		F088E454	Deflate	up : Descriptor...	FAT
P1043068_na (1).mp4	25 745 597	25 749 527	2023-10-27 19:23				-		E24205EA	Deflate	up : Descriptor...	FAT
P1043068_na.mp4	25 745 597	25 749 527	2023-10-27 19:23				-		E24205EA	Deflate	up : Descriptor...	FAT
P1043075_na.mp4	38 676 284	38 682 189	2023-10-27 19:23				-		B5651AA8	Deflate	up : Descriptor...	FAT
P1043088_na.mp4	32 313 606	32 318 541	2023-10-27 18:44				-		06E4873B	Deflate	up : Descriptor...	FAT
P1043089_na.mp4	31 591 615	31 596 440	2023-10-27 18:45				-		B0837A43	Deflate	up : Descriptor...	FAT
P1043106_na.mp4	31 939 372	31 944 247	2023-10-27 18:45				-		72A92641	Deflate	up : Descriptor...	FAT
P1043107_na.mp4	35 210 742	35 216 117	2023-10-27 18:45				-		A3767E9A	Deflate	up : Descriptor...	FAT
P1043115_na.mp4	24 340 175	24 343 890	2023-10-27 18:45				-		007C04A7	Deflate	up : Descriptor...	FAT
P1043122_na.mp4	26 161 715	26 165 710	2023-10-27 18:41				-		78EB1299	Deflate	up : Descriptor...	FAT
P1043129_na.mp4	31 207 113	31 211 878	2023-10-27 18:45				-		EE8D9FD4	Deflate	up : Descriptor...	FAT

Note. The Video Files Recorded On The Local Machine.

The video clips were converted into data frames and then saved as distinct image files in order to create synthetic data as shown in Figure B2. The frames were extracted using Python programming, and the photographs were then saved. The software reads the content using the OpenCV function, which makes it easier to extract the video's frames per second (FPS). The frames per second (FPS) option controls how often each individual frame is recorded by the system, which is one-fifth of a second. Frames saved in the Joint Photographic Experts Group (JPEG) format are found in the output folder. Figure 31 shows the output of the frames that were retrieved from the films.

Figure 31

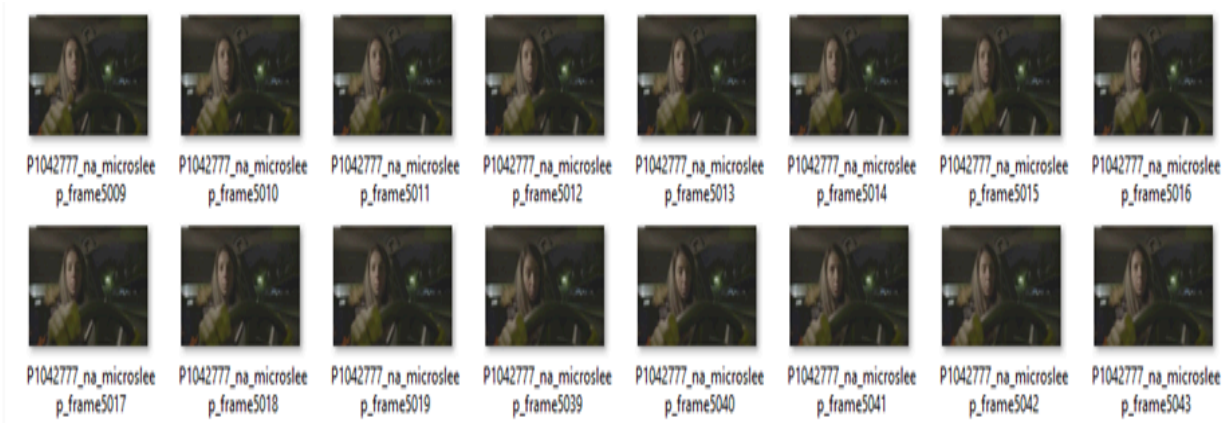
Output Of Frames Extracted From The Videos

Note. The Frame Level Dataset, Which Was Obtained From The Video Raw Dataset

Annotations have been added to the retrieved frame-level dataset for three distinct events: Yawning, microsleep, and alertness. The frames classified as alert correlate to times when the driver's eyes are closed, whereas the frames recognized as microsleep episodes occur when the driver's eyes are closed. Moreover, the frames labeled as instances of yawning also coincide with the driver's mouth opening. 31,016 images make up the final annotated dataset, each labeled with an appropriate number. The frame-level dataset, extracted without any supporting annotations, is shown in Figure 32.

Figure 32

Frame Level Dataset

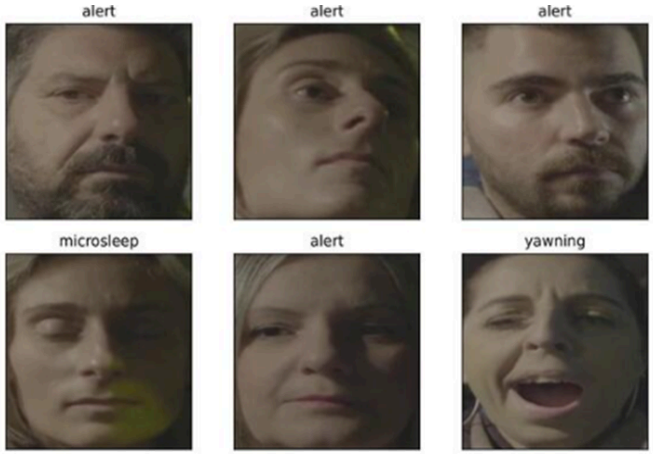


Note. The Above Figure Shows The Frame Level Dataset Without Any Annotations.

Synthetic datasets are created using a variety of data augmentation techniques, including rotation, translation, rescaling, flipping, cropping, and noise injection. This makes it easier to create a larger dataset that will be used to train our model. The frame-level dataset has been cropped in order to provide synthetic data. By setting up particular crop settings for the dataset, this was accomplished. Photos that have been cropped are kept in a separate folder. A screenshot of the data frame before cropping is shown in Figure 33.

Figure 33

Data Frame After Annotation



Note. The Annotated Data For Alert, Microsleep, And Yawning Is Displayed.

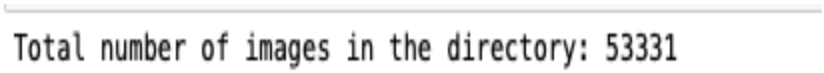
Data Pre-Processing

Before supplying the image dataset to machine learning and deep learning algorithms, it is analyzed using Exploratory Data Analysis to identify any necessary changes to enumerate the features and simplify the information. Data visualization is used to do this. For this project, the full dataset is subjected to Exploratory Data Analysis in order to identify any changes.This project uses Python and Python utilities such as numpy, matplotlib, and seaborn to illustrate dataset patterns.

There are 53331 photos in the dataset. On computers, working with large datasets has an impact. Machine learning techniques may need time and financial resources to process this data. Large datasets enhance machine learning models with a greater variety of data kinds.The total number of photos in the collection is displayed in Figure 34 .

Figure 34

Total Number Of Images Of The Entire Dataset

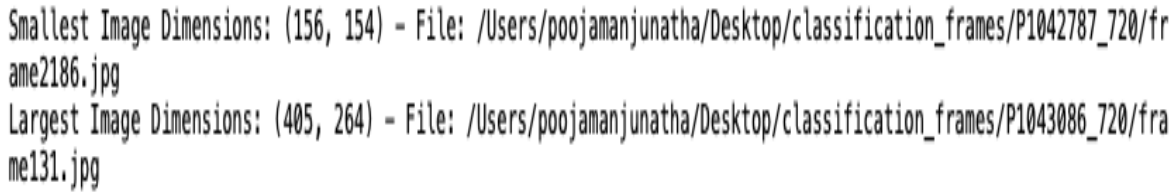


*Note.*The Total Number Of Photos In The Complete Dataset Is Displayed In The Above Figure.

The pixels in the smallest image are 156 × 154. A 405 x 264 pixel image is the largest one in the chosen file. It is observed that there is a variation in image sizes in the dataset. They make the appropriate adjustments. The dataset's smallest and largest image dimensions are displayed in Figure 35.

Figure 35

The Smallest And Largest Image Dimensions For The Entire Dataset



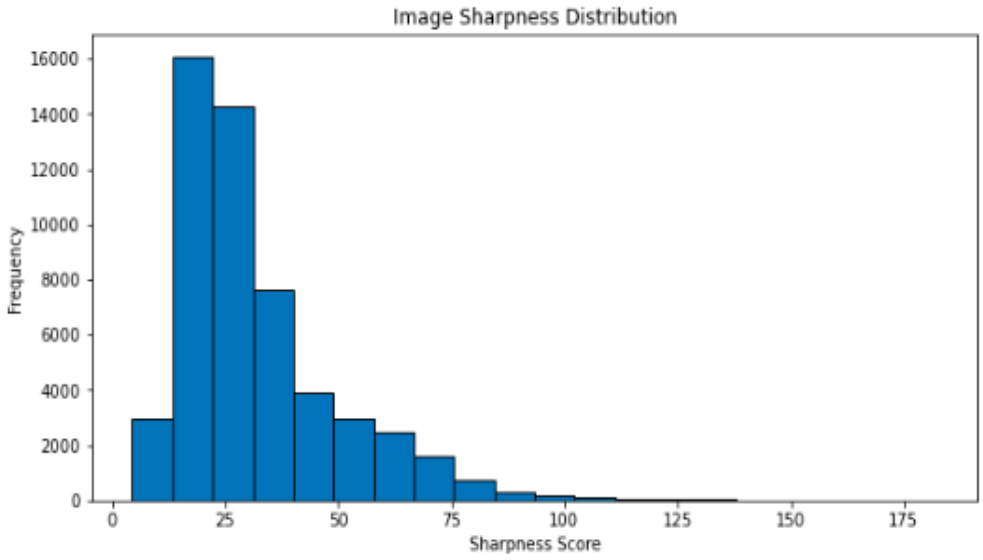
*Note.*The Dataset's Smallest And Largest Image Dimensions Are Displayed In The Graphic.

The histogram in Figure 36 shows the picture sharpness distribution. Sharpness is on the x-axis and frequency on the y. A photo's sharpness score indicates focus. Used to assess visual clarity. Most photographs are crisp, with a 20–30 score. If the sharpness score is 20–30, most

photographs are clear. The dataset has clear images. The pictures are of the same quality because there is only one tight peak. Figure 36 shows the Image Sharpness Distribution for the entire dataset demonstrated in Figure B13.

Figure 36

Image Sharpness Distribution For The Entire Dataset

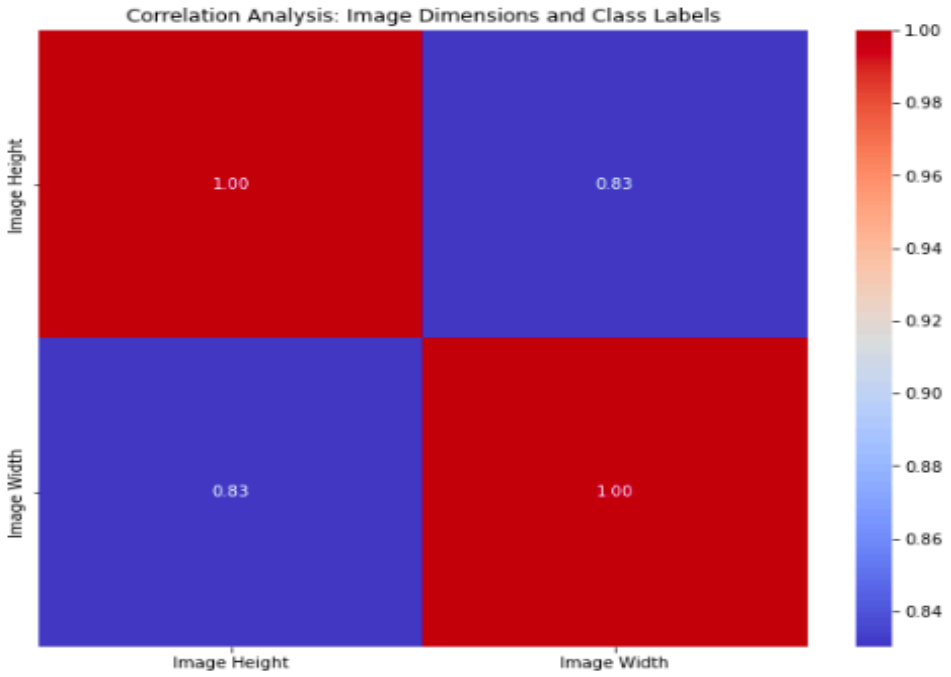


*Note.*The Image Sharpness Distribution For The Complete Dataset Is Displayed.

Given that height and width have a strong correlation (0.83), they are related shown in Figure 37 demonstrated in Figure B14. The coefficient of correlation, which is a number between -1 and 1, shows the linear relationship between two components. Because of their tight correlation, height rises with increasing breadth. Positive correlations show that the aspect ratio of the dataset is preserved. Similar aspect ratio datasets are ideal for machine learning.

Figure 37

Heatmap To Show Coefficient Correlation For The Entire Dataset



*Note.*The Correlation Of Image Dimensions Is Displayed In The Above Graphic.

Missing Data. OpenCV tests for viewable or unreadable pictures are performed via Python programs. "All images are readable" is displayed by compatible image formats, whilst an error message is displayed by incompatible ones. Additionally, the code looks for black, empty images with 0 pixels. Since there isn't an error notice, there aren't any empty images.The output code that displays no empty images is displayed in Figure 38.

Figure 38

Output Of The Code That Shows No Empty Images

```
All images are readable and not empty.
```

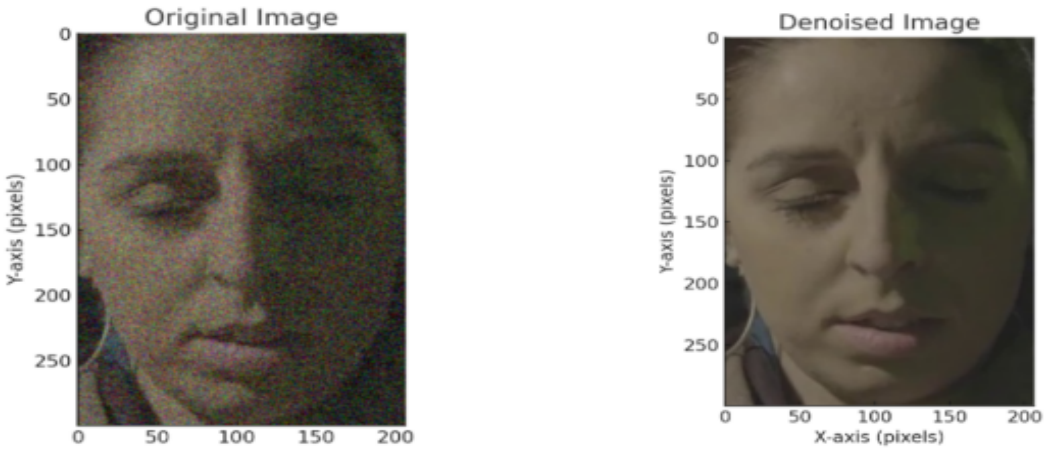
*Note.*The Code's Output, Which Displays No Empty Images, Is Depicted In The Above Figure.

Image Denoising. The production of images requires noise reduction. Performance of algorithms is determined by data. The most popular methods for dealing with noisy image data are autoencoders, filtering, denoising, and noise robust training. An arbitrary value is applied to pixels to create noise. It modifies the hues of images. A picture can have noise added to it at any point. The visuals are dominated by impulse, salt and pepper, speckle, and Gaussian noises. The survey displays the feelings of drivers. Noise in pictures can come from a number of sources. Noisy photos can be produced by multiple sources.

Thermal noise from a heated sensor can cause distortion in photographs when the camera is used for extended periods of time. Noise is increased by reflections, shifting illumination, and shadows. Photos and videos captured while moving may become loud and blurry due to sudden movements. Consequently, removing picture noise is essential to preprocessing. With OpenCV, several kinds of noise can be eliminated from images. For this project, the fast Nl Means Denoising for Colored Images is primarily utilized. Based on input h, which is typically 10, this function sets noise reduction. More noise is eliminated as h rises. template for designWindowsize is used to calculate search and weighted average windows. Comparable windows are found using this window size. The denoised image is presented in Figure 39.

Figure 39

Sample Denoised Image



Note. The Jupyter Notebook Image Sample Before And After Denoising

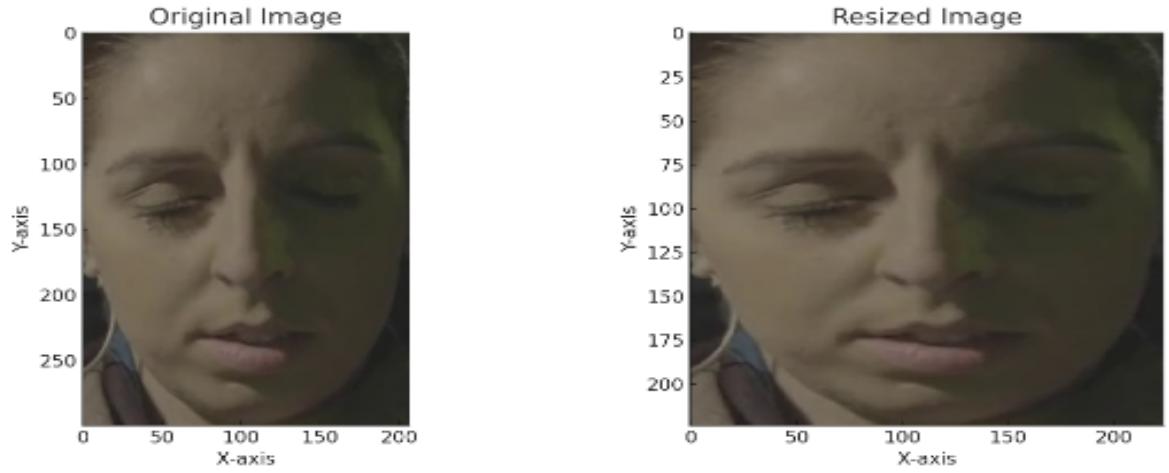
Handling Inconsistent Data. When gathered, data from various sources loses its credibility. This is less likely because the project has photos and data spread across multiple files inside of a directory. Duplication will be verified before merging information across all photo paths and annotations. Photo disagreements can be resolved with careful resizing. It's possible that duplicate images will interfere with data collecting. Copying is removed by determining picture covariance and correlation.

Image resizing. A step in data purification is "image resizing." A photo's pixel size varies when it is resized. Despite coming from various locations, the images' sizes could change. Prior to adding data to the model, pictures need to be scaled. Equal-sized data is necessary for certain machine learning algorithms to produce better results. Convolutional Neural Networks (CNNs) may require more memory since they require four times as many images to train a model. Data resizing tools include PyTorch, TensorFlow, OpenCV, and Pillow Image Library (PIL). OpenCV is used in this project to scale larger photos. The study found that machine

learning models with 224 x 224 picture resolution performed best, including Viola Jones, You Only Look Once (YOLO), convolutional neural networks, and Visual Geometry Group (VGG) Net. The original and scaled photos are displayed in Figure 40 . Image accuracy is enhanced with a larger resolution of 224 by 224 pixels. (300,207) is the original image quality, displaying both height and width. A scale of (224,224) is used.

Figure 40

Sample Resized Image



Note. An Example Of An Image From Jupyter Notebook Before And After Resizing

Data Transformation

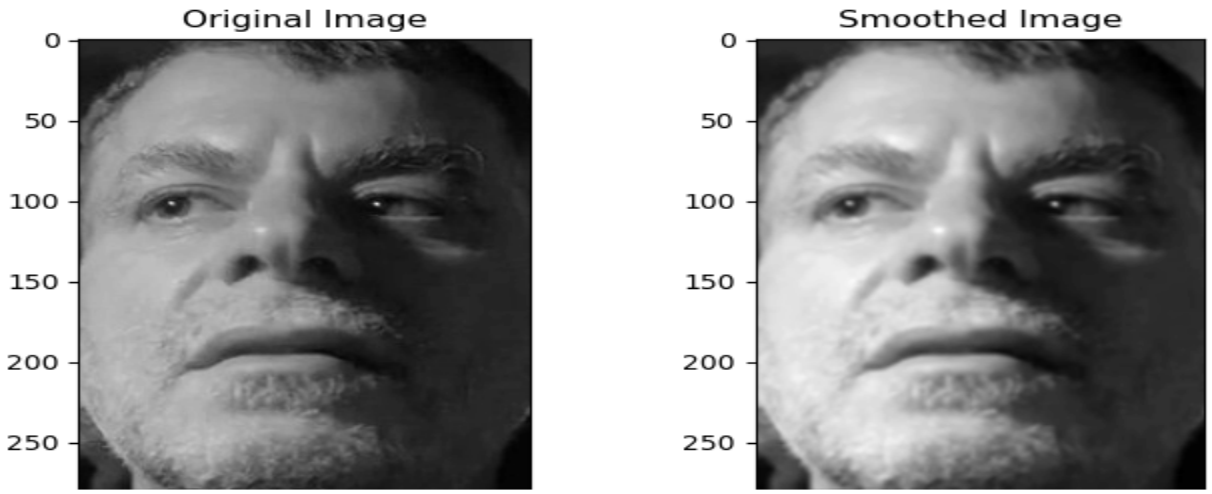
Within the field of image processing, the image transformation process is quite important. At this point, pre-processed datasets go through a conversion process to get the formats that the model needs. Several tools and scripts are used to accomplish this conversion. The program makes it possible to convert images from one format to another in order to enhance the identification of objects that might be difficult to tell apart. The altered image is then subjected to examination, analysis, and more tinkering using a range of image processing methods. Within the framework of the project at hand, the dataset is subjected to multiple changes after it has been

cleaned up and pre-processed in order to address problems like noise, incompleteness, and inconsistencies in the image files. The above described procedures involve the utilization of techniques like regularization, dimension reduction, normalization, and data smoothing on the images.

Data Smoothing. Factual information that is gathered, examined, and evaluated for purposes of Smoothing is a technique used to reduce pixelation and reduce noise in photos. Eliminating image disturbances is crucial when it comes to photo pre-processing since it reduces the possibility of significant information loss. By applying a low-pass filter kernel and convolution to the images, high-frequency elements like noise and edges are eliminated from photos during the smoothing process. Pictures acquired from different sources may differ in terms of edge quality and sharpness. As a result, to improve their usability, a smoothing or blurring operation is frequently required. Commonly used image smoothing methods include bilateral filtering, median, Gaussian, and averaging. In order to replace the core element with the arithmetic mean of all the pixels in the kernel area, the averaging approach is utilized in this project's methodology. The contours of an unscaled image and its matching smoothed counterpart are compared in Figure 41.

Figure 41

Image Smoothing

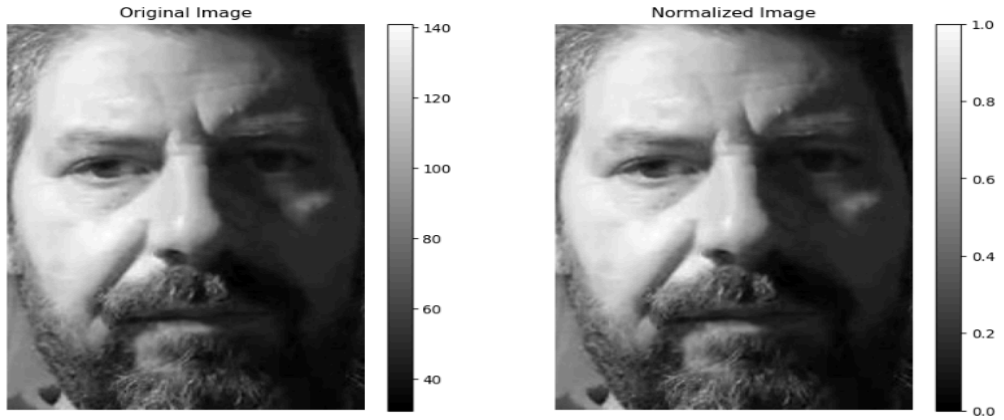


Note. Pictures From The Jupyter Notebook Both With And Without Picture Smoothing.

Image Normalization. An usual first step in image analysis is to normalize the images. To ensure that the data distribution is the same in all of the provided photos, it modifies the pixel intensity. This feature aids in the faster convergence of the model during training. Range normalization is usually used to put each pixel's possible values in the range [0, 1]. The range, or the difference between the two extremes, is divided by the minimum value and then subtracted to change the pixel values. This means that the normalized distribution stays inside the range. When the pixel values are uniformly distributed, training the model is much simpler. Figure 42 shows the pre- and post-normalization configurations of the image as well as the image normalization.

Figure 42

Image Normalization




```
Original Image Pixel Values:
[[ 97 100 101 ...  50  51  51]
 [ 85  96  96 ...  51  51  51]
 [ 76  86  80 ...  51  51  51]
 ...
 [ 65  65  64 ...  37  37  37]
 [ 64  64  63 ...  37  37  37]
 [ 65  65  64 ...  37  37  37]]

Normalized Image Pixel Values:
[[0.6         0.6272727  0.6363636  ... 0.17272727 0.18181819 0.18181819]
 [0.4909091  0.59090906 0.59090906 ... 0.18181819 0.18181819 0.18181819]
 [0.4090909  0.5        0.44545454 ... 0.18181819 0.18181819 0.18181819]
 ...
 [0.3090909  0.3090909  0.3        ... 0.05454545 0.05454545 0.05454545]
 [0.3        0.3        0.29090908 ... 0.05454545 0.05454545 0.05454545]
 [0.3090909  0.3090909  0.3        ... 0.05454545 0.05454545 0.05454545]]
```

Note. The Original And Normalized Frame Data Are Displayed In The Above Figure.

Data regularization. It is particularly important in datasets of driver photos used for tiredness detection, as it helps address open problems and minimize overfitting. This is because precision and uniformity are guaranteed by data regularization. Techniques like data augmentation and dropout can be used to improve the performance of the model. An unbalanced dataset becomes more helpful with data augmentation. Even though non-drowsy photographs are more prevalent, the augmentation aims to balance the collection. To achieve this balance, a number of methods are used, including rotation, flipping an image horizontally or vertically, cropping the image, and applying Gaussian Noise, as seen in Figure B9 and Figure B10. By creating fresh, unique photos from the preexisting collection, these algorithms grow it without adding any duplicate data. Figure 43, Figure 44, Figure 45 and Figure 46 depict data

augmentation methods for detecting tiredness in driver image datasets. Dropout techniques are frequently used in deep learning models to inhibit coadaptation and encourage feature learning independently of one another.

Figure 43

Image Augmentation Through Flipping

Note. The Original Image And The Enhanced Image Created By Flipping Are Displayed In The Above Figure.

Figure 44

Image Augmentation Through Rotation

Note. The Original Image And The Rotate-Enhanced Image Are Displayed In The Above Figure.

Figure 45

Image Augmentation Through Cropping

Note. The Original Image And The Cropped-In Augmented Image Are Seen In The Top Figure.

Figure 46

Image Augmentation Through Gaussian Noise

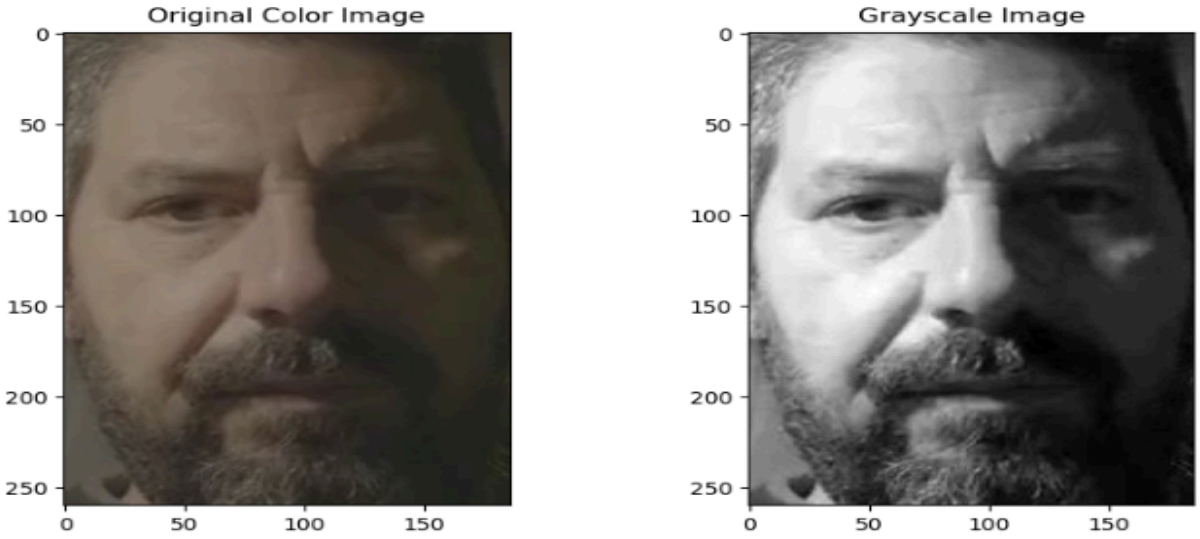
Note. The Original Image And The Augmented Image Created By Gaussian Noise Are Displayed In The Above Figure.

Principal Component Analysis (PCA). This specific strategy's applicability in linear dimensionality reduction is frequently seen in datasets with a multitude of attributes. Reducing dataset properties and choosing pertinent variables for modeling are made easier by applying a linear algebra methodology, all the while maintaining data quality. This method's main goal is to reduce the amount of redundant data by keeping the number of related variables to a minimum. Sorting photos into two categories drowsiness and non-drowsiness, is the aim of this investigation. System efficiency and computation performance both noticeably decline as the number of photos in a system rises. This decrease is explained by the fact that expenses have increased due to a notable rise in the amount of data absorbed by pixels. It is thought to be beneficial to reduce the quantity of the data, and Principal Component Analysis (PCA) is a method frequently used to do this. By identifying fewer main components, Principal Component Analysis (PCA) is a technique that lowers the dimensionality of picture matrices. These elements hold significant data on the levels of pixel intensity shown in Figure B23. Higher levels of noise and less information are typically present in the later components, while the former frequently contain more information. Principal Component Analysis (PCA) is a method that is frequently used to eliminate some aspects while maintaining essential attributeS depicted in Figure B27 and its result in Figure B28.

A dataset of grayscale photographs is used to conduct a systematic application of the Principal Component Analysis (PCA) technique, represented in Figure B26. A grayscale image that has been evaluated with Principal Component Analysis (PCA) is shown in Figure 47.

Figure 47

Grayscale Image



Note. For The Purposes Of This Investigation, Principal Component Analysis (Pca) Is Being Used On The Supplied Image.

The grayscale image is used to further reduce the features' dimensions. A scree plot is created to determine the number of components needed to explain the 95% variance in each image. Take two highly connected variables as an example. One variable can be expressed as a linear combination of others. By computing the Eigenvector of input matrices, Principal Component Analysis(PCA) transfers the variance of the second variable to the first variable, represented in Figure B12. The Principal Component Analysis(PCA) fit function is used to fit Grayscale images, and the Principal Component Analysis(PCA) explained_variance_ratio is used to calculate cumulative variance. Components that explain 95% of the variance are identified**.** Figure 48 displays the cumulative variance and number of primary components in a scree plot.

Figure 48

Scree Plot

Note. Show The Number Of Components Needed To Explain 95% Of The Variation.

As can be seen from the results in Figure 49, 30 components, not 224 pixels, explain 95% of the variance in the picture. The incremental Principal Component Analysis (PCA) module inverse transform approach is used to recreate the original image, as shown in Figure 49 .To accomplish this reconstruction, 34 components are used, and the clarity of the final image is then evaluated .

Figure 49

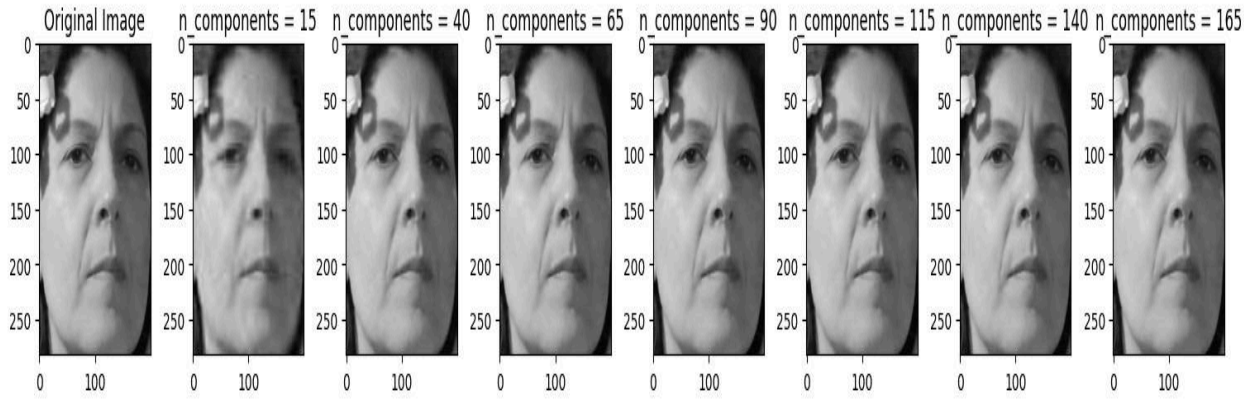
Reconstructed Image

Note. Rebuilding A Picture With The Principal Component Analysis (Pca) Method.

While the reconstructed image retains identification, its visibility and clarity are compromised. Gradual component augmentation improves image clarity and makes it possible to capture details that were previously missed. Based on the results shown in Figure 49, it is evident that photos with component 100 have greater clarity levels. Increasing the number of components above 100 does not result in a discernible improvement in clarity. After the original image is compressed, its representation is whittled down to 100 components, which are ready for further processing.The reconstructed image with different components is shown in Figure 50, also represented in Figure B25.

Figure 50

Reconstructed Images With Different Components



Note. PCA Image Reconstruction With Varying Numbers Of Components

Unlike Principal Component Analysis (PCA), dimensionality reduction can be accomplished in a variety of ways. In many different fields, methodologies like Singular Value Decomposition (SVD), Principal Component Analysis (PCA), and t-distributed Stochastic Neighbor Embedding (t-SNE) are frequently used. But it's crucial to recognize that each of these approaches has trade-offs and inherent restrictions that need to be considered in order to make an informed decision. Principal Component Analysis (PCA) is a computationally efficient technique that reduces the size and processing time of large, high-dimensional datasets by removing less important features. Even if the Singular Value Decomposition (SVD) keeps undesired elements, it allows for their removal in the end. Principal Component Analysis (PCA) is carried out by removing less important features from the original matrix using the Singular Value Decomposition (SVD) technique, but at the expense of longer computation times. To minimize the dimensionality of the data and improve processing performance for a set of 1,000 images, Principal Component Analysis (PCA) has been used as a method. It is anticipated that this would improve the model's performance.

Several different data transformation methods are used, three image preparation methods were used in the model construction process, picture Resizing, Denoising, and Normalization. This choice was taken with the understanding that the model can independently lower the size of an image.

Data Preparation

One of the most crucial stages in the creation of a model is data preparation. After the data has been cleaned and transformed during the pre-processing phase, the next step is to get it ready for modeling. For the machine learning models to function effectively, the data must be separated into training, test, and validation sets. In general, the data is split into the following percentages to ensure that machine learning models operate efficiently. In other words, the model is trained on 80% of the data, and it is tested on the remaining. The data split is represented in Figure B14 and represented by a bar graph as per Figure B15. The percentage of data utilized to test the model varies depending on the type of dataset; typically, 20–30% is employed. The model receives test data for efficiency testing and training data for training once the data is separated. The data for the project is divided into 80% training and 20% testing, with the help of several Python modules. The remaining data is used for validation, and the data used for training is once more divided into ratios. As a result, 80-10-10 is the final dataset's division for training, testing, and validation. The creation of the Train,Test and Validation folder is shown in Figure 51, Figure 52 and Figure 53 respectively. A sample of the training and validation sets is displayed in Figures 54.

Figure 51

Splitting And Creating Train Folder

Note. A New Folder Train Is Made And Photographs Are Transferred To It, As Seen In The Above Illustration.

Figure 52

Splitting and Creating Test Folder

Note. A New Folder Called "Test" Is Made In The Above Figure, And Pictures Are Then Transferred To It.

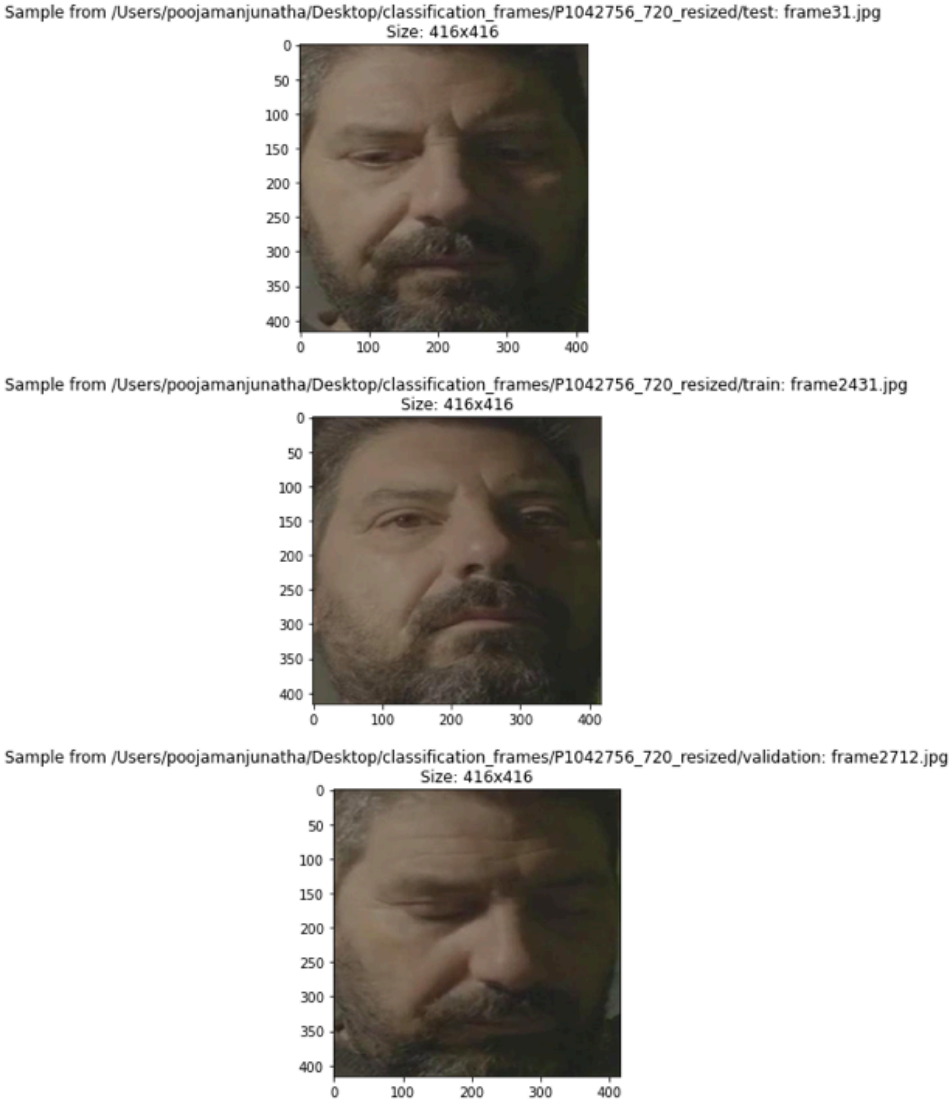
Figure 53

Splitting And Creating Validation Folder

Note. A New Folder Validation Is Made And Photographs Are Moved To It, As Seen In The Above Figure.

Figure 54

Sample From The Train, Test And Validation Folder



Note. One Sample From Each Of The Train, Test, And Validation Folder Is Displayed In The Above Figure.

Data Statistics

A raw 130 video set of drivers, both male and female, is used for the project. The subjects, who are both male and female, have a variety of attributes, including age, hair color, spectacles, beards, and so on. They are filmed in actual cars in both day and night lighting scenarios. Every video is in the mp4 format, runs at 25 frames per second, has two resolutions, and is silent. It has a 697.05 megabyte size. After these films are converted to frames, the picture data contains 53,331 photos in the directory, occupying roughly 1.42 gigabytes.

A small amount of null, partial, or missing data was handled during the data pre-processing phase. Data transformation applied to the set of data in accordance with the model algorithm's specifications. Certain image resizing, pixel value normalization, label encoding, data augmentation, and other modifications are necessary for some model techniques, such as Convolutional Neural Networks (CNNs), but You Only Look Once (YOLO) requires none of them. The characteristics of the image are altered at every level of data cleansing, preprocessing, and transformation, necessitating a significant amount of storage space. Therefore, once all pre-processing techniques have been done, the photos are saved rather than being saved at each intermediate level. The final altered images are then saved to a folder and to the data frame. The transformation codes and the results can be seen in Figure B17, Figure B18, Figure B19 and Figure B20.

The modifications noticed at every phase of the pre-processing procedures used on the image dataset are shown in Table 7. After using a variety of data preparation methods, the size of the image dataset has dramatically shrunk from 1.4 GB to 755.4 MB.

Table 7

Image Data Statistics

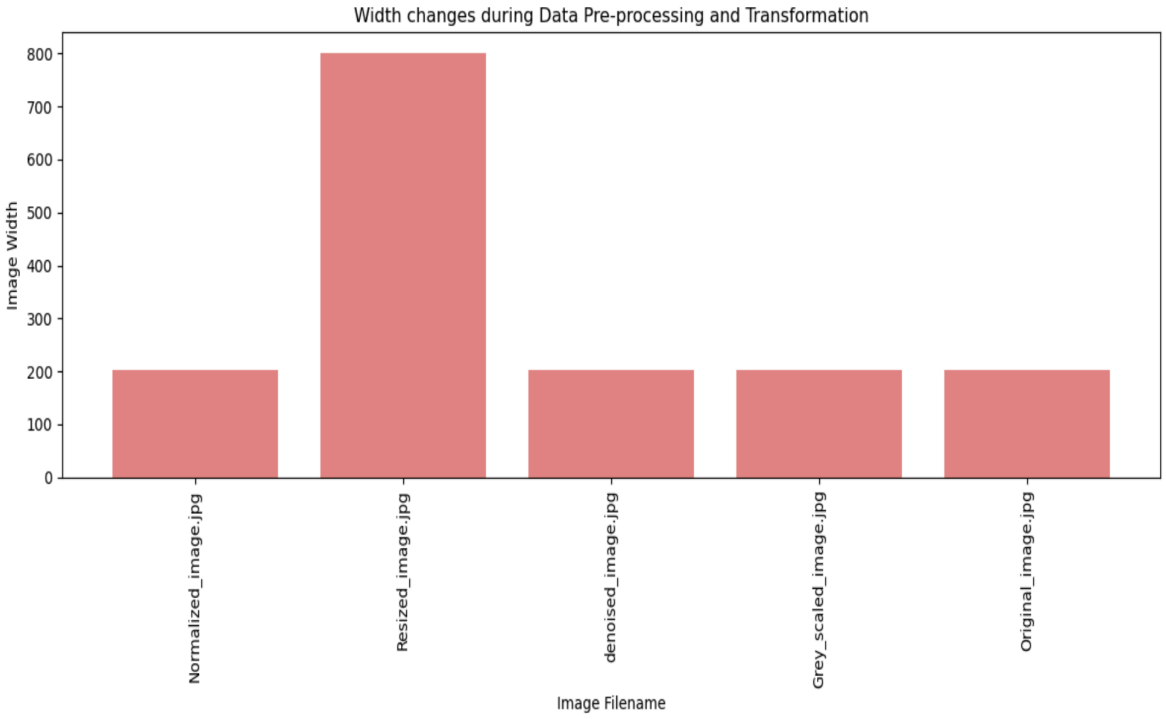
Stage	Steps	Applicable Statistic
Raw Data	Data collected from Night Time Yawning Microsleep Eyeblink Distraction (NITYMED)	697.05 MB, 130 mp4 video
Sampled data	Manual sampling of images	1.42 GB 53,331 Images
Data pre-processing	Null value handling	53,331 images
	Incomplete and missing data handling	53,331 images
Data Transformation	Image resizing	53,331 images
	Image denoising	53,331 images
	Image grayscale	53,331 images
	Image smoothing	53,311 images Shape
	Image normalization	53,311 images Shape
	Image regularization	53,311 images Shape
	Principal Component Analysis(PCA)	components - 1000
Preparation	Test	80%
	Training	10%
	Validation	10%

Note. The Above Table Shows The Statistical Data Of The Dataset Used For The Project.

Histograms in Figure 55, Figure 56 and Figure 57 show how the distribution of images varies in width, height, and size following the application of each pre-processing approach. Figure 58 displays the changes in the images' width that happened throughout the transformation, Figure 59 the changes in the images' height, and Figure 60 the changes in the images' size that happened during the stages of data pre-processing and data transformation.

Figure 55

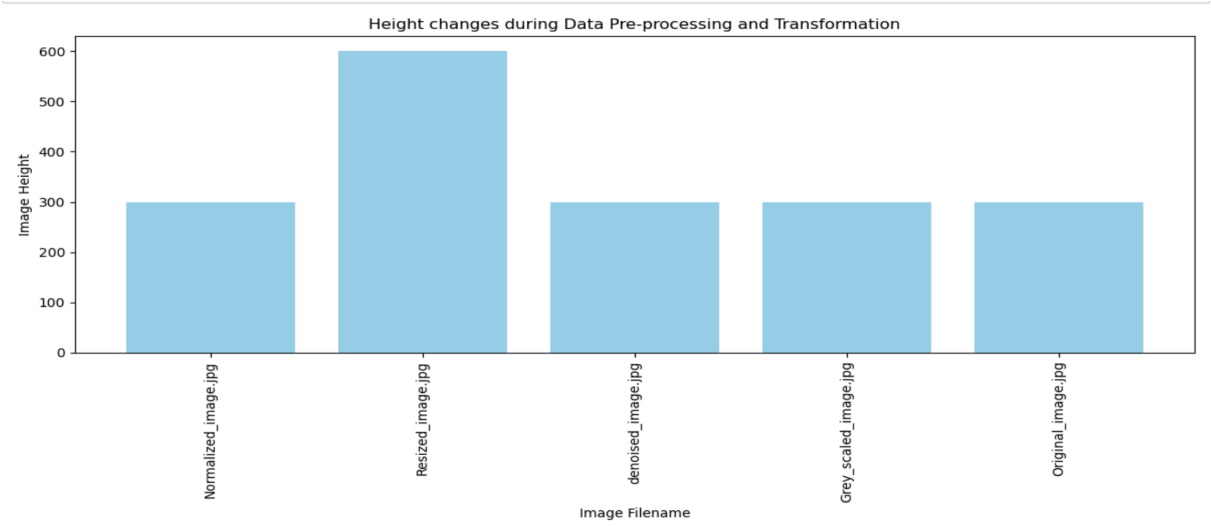
Width Changes During Data Pre-Processing And Transformation



Note. The Width Variations That Occur During Data Pre-Processing And Transformation Are Depicted In The Above Figure.

Figure 56

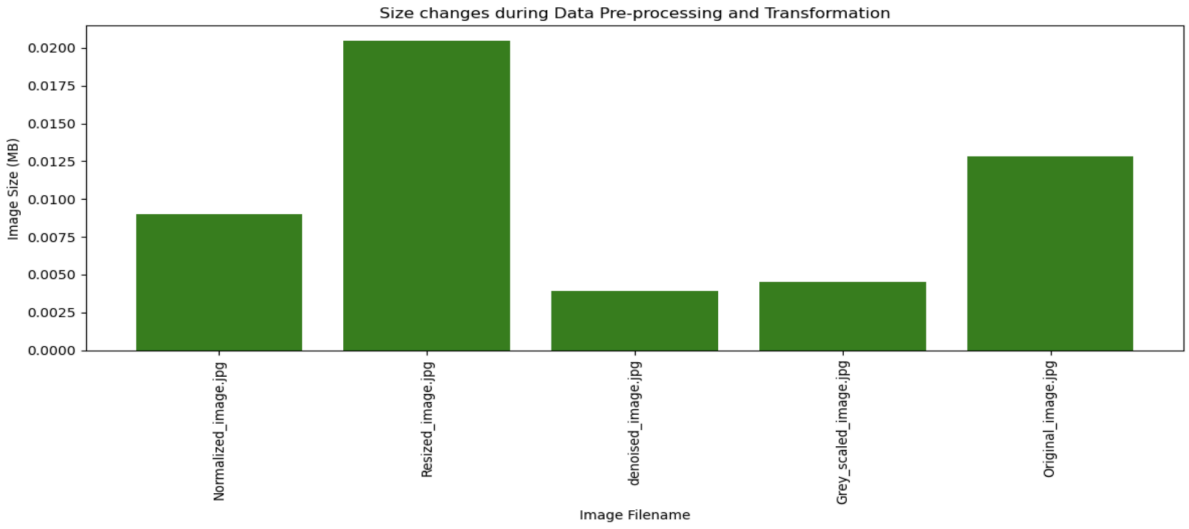
Height Changes During Data Preprocessing And Transformation



Note. The Height Variations That Occur During Data Transformation And Preparation Are Depicted In The Above Figure.

Figure 57

Image Size Changes During Data Preprocessing and Transformation

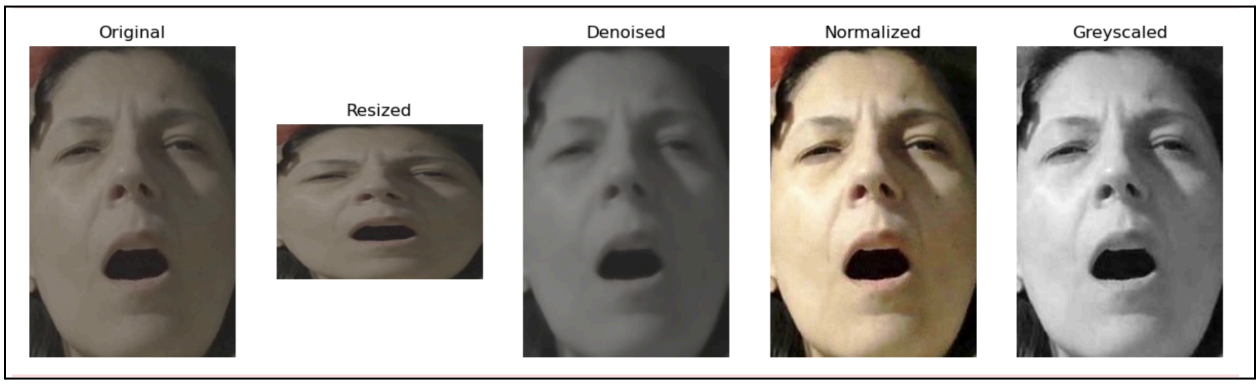


Note. The Image Size Variations That Occur During Data Transformation And Preprocessing Are Depicted In The Above Figure.

The minor changes made to the various target images in our data collection are shown in Figure 58, Figure 59, Figure 60. The transformation applied to a set of yawning images is shown in Figure 57, and the transformation applied to an alert picture set is shown in Figure 58. Lastly, a transformation applied to a group of microsleep images.

Figure 58

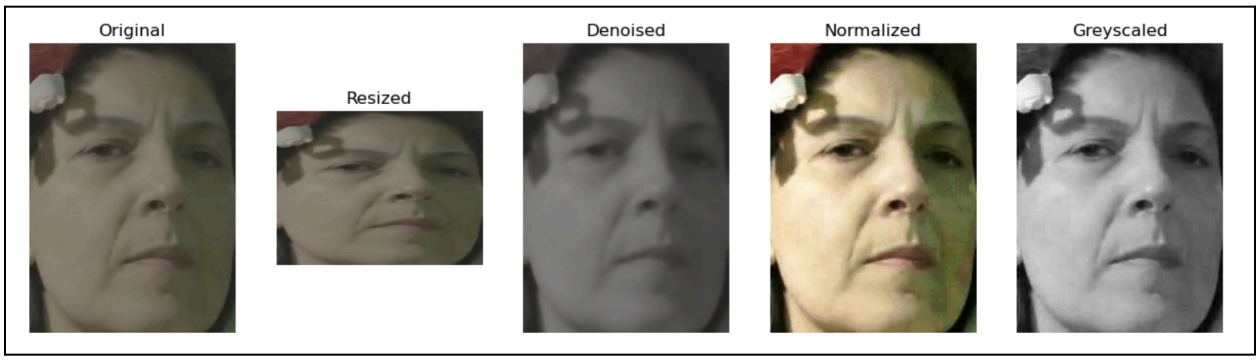
Image Transformation After Pre-Processing On An Image Where Target Feature Is Yawning



Note. The Image Transformation Following Pre-Processing On An Image With A Yawning Target Feature Is Depicted In The Above Figure.

Figure 59

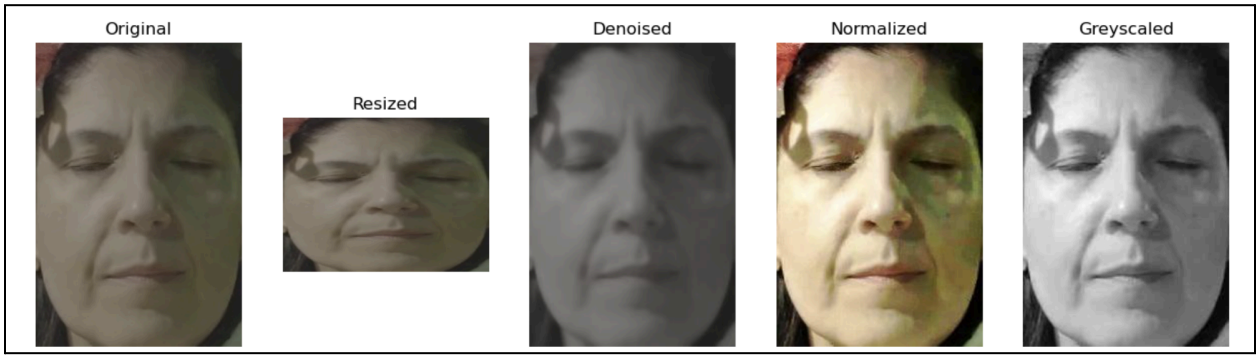
Image Transformation After Pre-Processing On An Image Where Target Feature Is Alert



Note. The Image Transformation Following Pre-Processing On An Image With An Alert Target Feature Is Depicted In The Above Figure.

Figure 60

Image Transformation After Pre-Processing On An Image Where Target Feature Is Microsleep



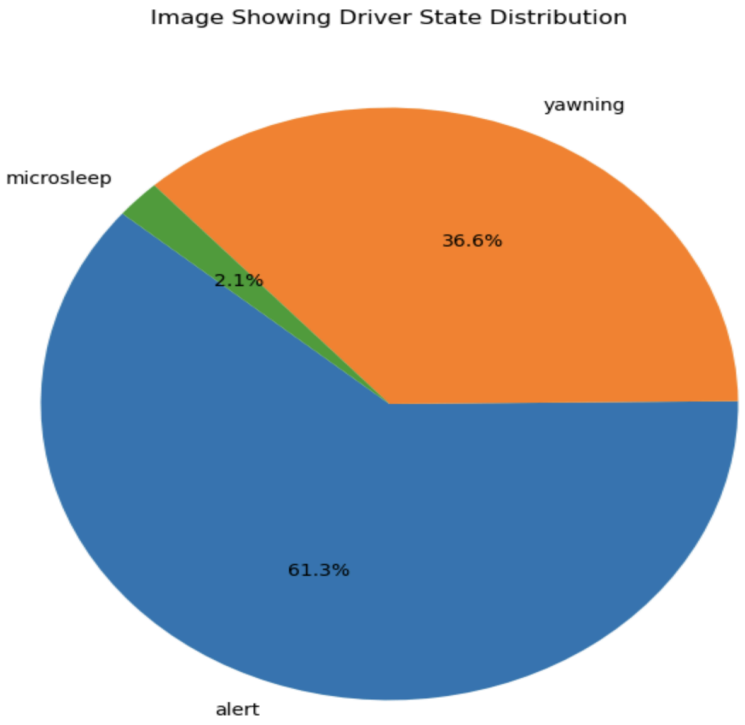
Note. The Image Transformation Following Pre-Processing On An Image Using Microsleep As The Target Feature Is Depicted In The Above Figure.

Data Analytics Results

Following the phases of data cleansing, pre-processing, and Exploratory Data Analysis (EDA), big data analytics is performed on the image data set. An early examination of the data set's proportion of the goal values is presented in Figure 61**.** Pie charts are taken into consideration for this since they provide an accurate summary of the image data set. Scatter plots and histograms are also thought to be useful tools for visualizing the distribution of images across target levels.

Figure 61

Image Showing Target Features Distribution In The Final Dataset

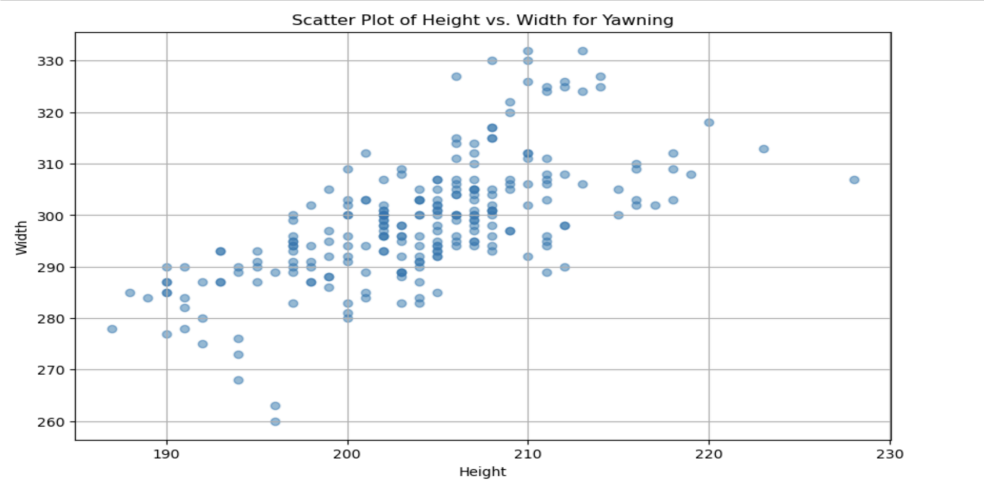


Note. The Target Features Distribution In The Final Dataset Is Displayed In The Above Figure.

The scatter plot for the driver state in Figures 62 through 64 shows how the height and width of the data set relate to one another. The scatter plot for the driver's yawning condition in Figure 61 shows how the height and width of the data set relate to one another. The scatter plot for the driver's microsleep condition in Figure 63 shows the correlation between the data set's height and width. A scatter plot representing the driver's alert state is shown in Figure 64, which also shows the correlation between the data set's height and width. A histogram of the image widths and heights is shown in Figure 65. It is shown with legends and indicates the average values for height and width on the histograms.

Figure 62

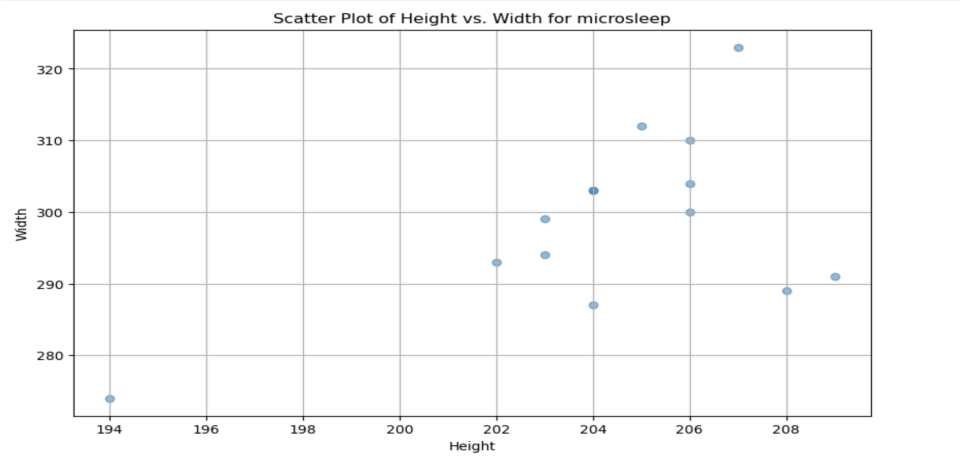
Scatter Plot For Height Vs Width For The Target Feature Yawning



Note. The Above Figure Shows The Scatter Plot For Height Vs Width For The Target Feature Yawning.

Figure 63

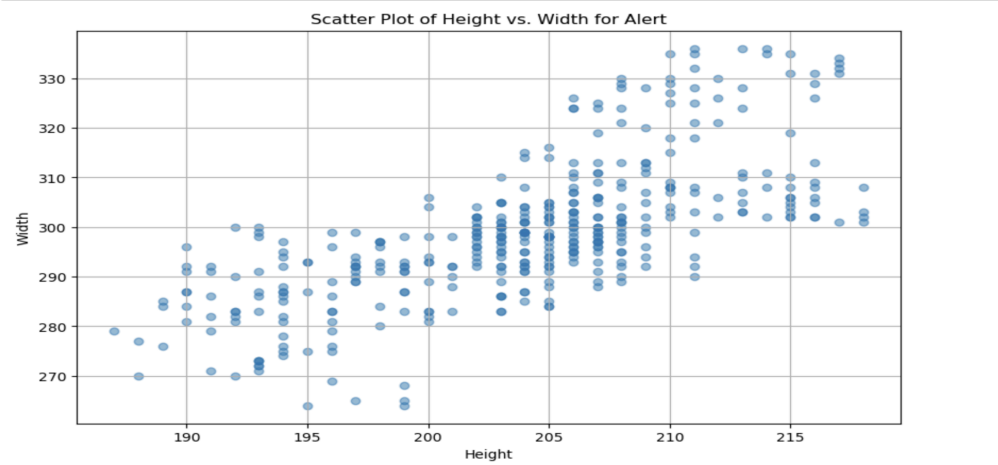
Scatter Plot For Height Vs Width For The Target Feature Microsleep



Note. The Target Feature Microsleep's Height Vs. Width Scatter Plot Is Displayed In The Above Figure.

Figure 64

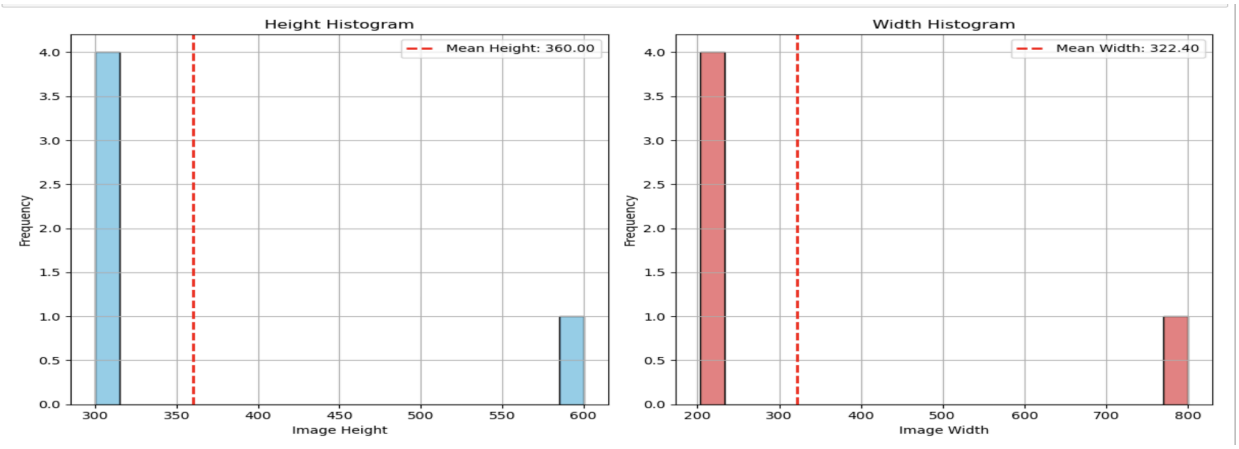
Scatter Plot For Height Vs Width For The Target Feature Alert



Note. The Target Feature Alert's Height Vs. Width Scatter Plot Is Displayed In The Above Figure.

Figure 65

Histogram Of Mean Heights And Widths In The Transformed Dataset



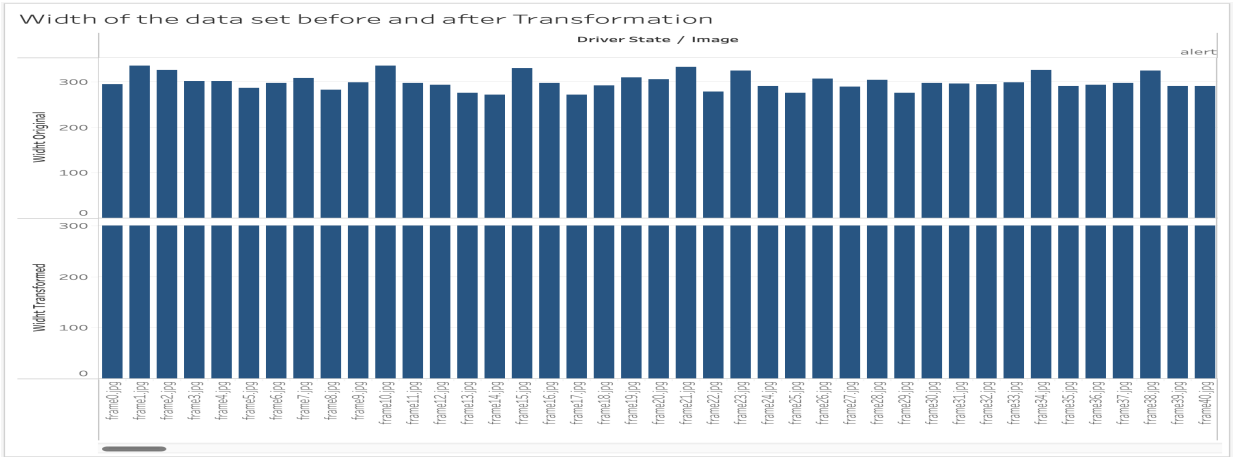
Note. The Histogram Of Mean Heights And Widths In The Transformed Dataset Is Displayed In The Above Figure.

The barcharts that are provided below are highly useful for displaying and comparing data from various target datasets. Bar charts employ rectangular bars of varying heights to

display the numbers associated with each dataset. The difference between the original raw data's height and width and the transformed data's width and height is depicted in Figure 66 and Figure 67 respectively.

Figure 66

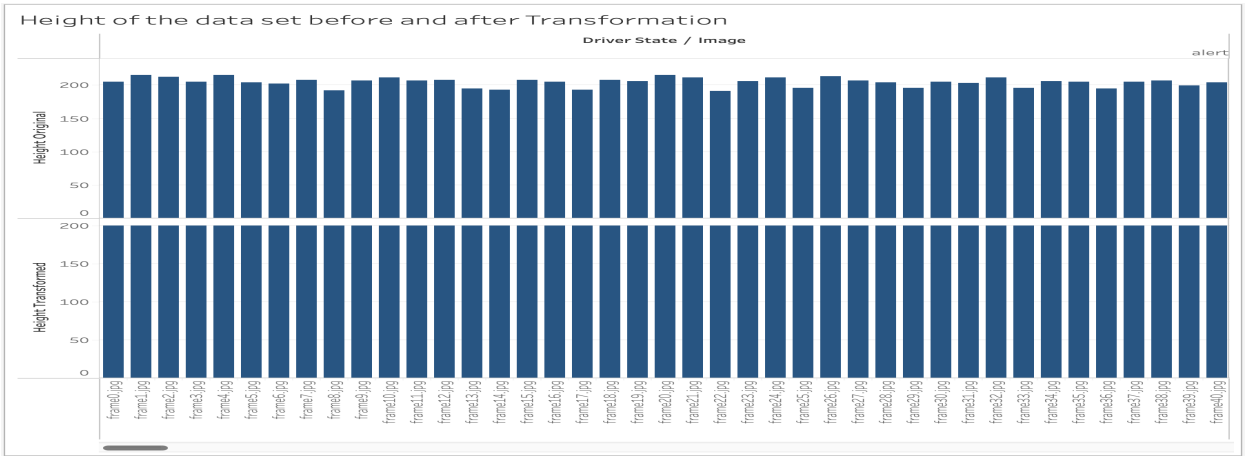
Width Of The Data Set Before And After Transformation



Note. The Width Of The Data Set Both Before And After Transformation Is Depicted In The Above Picture.

Figure 67

Height Of The Data Set Before And After Transformation



Note. The Height Of The Data Set Before And After Transformation Is Depicted In This Figure.

Model Development

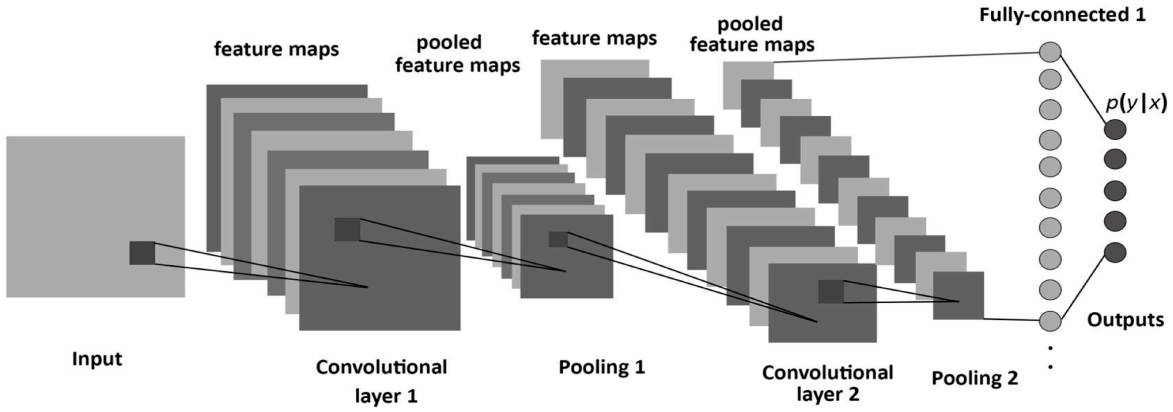
Model Proposals

The optimal model selection for image classification is required to effectively identify and detect drowsiness in the images. The ultimate model must possess the ability to effectively manage vast quantities of visual input, analyze images and the corresponding categories, and generate extremely accurate predictions. Given the aforementioned circumstances, four advanced architecture models Visual Geometry Group 19, You Only Look Once Version 3, Convolutional Neural Network, and Viola Jones are proposed for the classification of microsleep, yawning, and alert images as shown in Figure B30. All three proposed picture categorization models are constructed using a fourth supervised deep learning technique called the Convolutional Neural Network (CNN).The comprehensive analysis and data transmission of the three model designs rely on a comprehension of the Convolutional Neural Network (CNN) algorithm and its constituent layers.

CNN. The construction of the Convolutional Neural Network (CNN) algorithm model was examined by Albelwi and Mahmood (2017), as depicted in Figure 68, in their research. Equations are employed to further elucidate the interconnected layers and their respective functions.

Figure 68

Architecture Of A Cnn Algorithm With Convolutional, Pooling, Dense, And Output Layers.



*Note.*The Internal Structure Of The Convolution Neural Network Algorithm Is Shown In Detail In The Figure.

The VGG 19, YOLOv 3, and Viola Jones are composed of many convolutional, pooling, and dense layers. This paper provides descriptions of the layers comprising each deep CNN architecture, as well as the equations and layers utilized to process input as it traverses each layer.

The convolutional layer accepts the input image, which is a matrix of numerical values. Kernels, or filters, are matrices of integers used to extract the features of the image as shown in Figure B32. Convolution is achieved by sliding the filter across the height and width of the input image and computing the dot product between the input and the filter. Equation (1) represents the provided information

$$a.b = \sum_{i=1}^{n} a_i * b_i \qquad (1)$$

Activation maps of the filter's size are generated by repeatedly convolving the filter with the input image. Equation (2) indicates that when an input picture of size (I x I) is convolved

with a filter of size (F x F), padding (P), and stride (S), the resulting image has the size (O x O). The output feature map size is determined by the filter size and stride (the steps at which convolution should occur).

$$O = \frac{(I-F)+2P}{S} + 1 \quad (2)$$

Where the padding is represented by the variable P. According to Eequation (3) below, it is decided by the number of pixels added to the image's frame during CNN processing in order to completely enclose the image.

$$P = \frac{k-1}{2} \quad (3)$$

The initial convolutional layers extract information pertaining to low-level features from the images, whereas subsequent layers acquire data regarding high-level features such as contours, curves, and particular individuals. In order to introduce non-linearity into the model, Rectified Linear Units (ReLU), which are a type of non-linear activation function, are commonly employed. If the input is negative, the output is set to 0, but if the input is positive, it returns the same value. Equation (4) provides the following information. ReLU, in contrast to other activation functions such as tanh and sigmoid, typically accelerates the training process.

$$F(x) = max(0, x) \quad (4)$$

The pooling layer receives the feature maps generated by the convolutional layer. The input feature maps are partitioned into several areas of equal size using pooling (R x R). The output of each region is a singular output. This layer decreases the dimension of the feature map, resulting in a reduction in the computational load needed for model training. There are two distinct types of pooling. Max pooling returns the highest value from the inputs contained within the filter area. The mean value of the filter covered region in the given feature map is determined using average pooling.

The Fully Connected layers, sometimes referred to as densely connected layers, receive the output from the pooling layer and linearly transform it using a weight matrix. The output vector is derived by performing the dot product of the input and weight matrices, followed by a non-linear transformation using an activation function as shown in Equation (5).

$$Y_{jk}(x) = f(\sum_{i=1}^{n_h} w_{jk} x_i + w_{j0}) \quad (5)$$

The activation function f applies to the dot product of the weight matrix and input, which is represented as $w_{jk} x_i$. Additionally, w_{j0} represents the bias that is applied. The SoftMax layer is commonly employed as the last layer in several CNN based models, such as GoogLeNet and Alex-Net. The layer converts the vector into a probabilistic distribution of the target classes. Equation (6) below provides the solution.

$$Y(i) = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}} \quad (6)$$

The study relies on CNN as the fundamental basis for all proposed advanced models. Further details regarding three other models can be found in the subsequent sections.

Figure 69 shows the pseudocode that is used to tell when a driver is getting drowsy. A Convolutional Neural Network (CNN) algorithm for finding drivers who are drowsy

Figure 69

Pseudocode of CNN Model

Input: Set of labeled images {X, y}, where X is the image data and y is the labels ('alert','microsleep','yawning').
Output: A trained Convolutional Neural Network (CNN) model capable of detecting drowsiness in drivers

1. Initialize the Convolutional Neural Network (CNN) model with random weights.

2. For each image in the dataset:
    a. Pass the image through a series of convolutional layers.
        i. Apply convolutional filters to extract features from the image. ii. Apply a ReLU activation function to introduce non-linearity.
        iii. Apply pooling to reduce the spatial dimensions.

    b. Flatten the output from the convolutional layers to create a single vector.

    c. Pass the flattened vector through fully connected layers.
        i. Apply a ReLU activation function after each fully connected layer. ii. Optionally, apply dropout for regularization.

    d. Output the final prediction through the softmax activation function for multi-class classification, or sigmoid for binary classification.

3. Calculate the loss between the predicted values and the true values using a suitable loss function, such as cross-entropy.

4. Backpropagate the loss through the network and adjust the weights using an optimization algorithm, such as SGD or Adam.

5. Repeat steps 2-4 for 30 epochs or until convergence.

6. Evaluate the model on a validation set to monitor performance and prevent overfitting.

7. Once training is complete, test the model on unseen data to assess its performance.

8. The trained model can now be used to predict the drowsiness state from new unseen driver images.

Note. The Above Figure Gives An Overview Of Data Flow In The Entire Model Training Process.

VGG19. The VGG19 model is an intricate convolutional neural network that represents the pinnacle of accomplishment in the realm of image identification and classification tasks. VGG19 is renowned for its remarkable depth and precision. The structure comprises a grand total of 19 layers, which are constructed from a sequence of convolutional and pooling layers. The architecture culminates in three interconnected levels that facilitate sophisticated thinking at a higher level. The architecture is renowned for its simplicity and effectiveness, employing concise filters in a repeating fashion to extract intricate features from images (Ahuja et al., 2020).

As VGG19 is a pre-trained convolutional neural network model that has been previously fine-tuned to achieve high performance in various image recognition tasks, it did not necessitate further optimization specifically for the purpose of drowsiness detection as shown in Figure B31. An effective method to enhance performance is to retrain the model using a dataset specifically focused on drowsiness. This process allows the model to fine-tune itself and acquire the essential properties required for accurately detecting drowsiness. The model's hyperparameters, such as the learning rate and layer count, are optimized as well. Furthermore, modifications are implemented to the preexisting layers, or new layers are incorporated into the model. In order to establish the optimal configuration of the model, most of these optimization tasks require experimentation and trial and error.

The VGG-19 architecture is used by the driver drowsiness monitoring system to sort image data into categories of how alert the driver is. Once the model has loaded and preprocessed frame-level driver face pictures, it resizes, normalizes, and improves the quality of the data, as shown in Figure B29 . With the help of error functions, the model learns by getting new weights as shown in Figure 70. Categorical and binary cross-entropy compare expected and real outputs to figure out how well a model is working. By sending gradients back through the

network, error calculations help the model keep its weights up to date. The network's internal features (learned representations) are changed to reduce mistakes, which makes it better at recognizing facial expressions that show drowsiness (kepesiova et al., 2020).

Figure 70

Pseudocode Of VGG19 Model

Pseudo code for training the pipeline of VGG 19 with hyperparameter tuning

1: Load frame-level image dataset of driver's face

2: Define and apply preprocessing steps for image dataset (e.g., resizing, normalization)

3: Initialize hyperparameters and grid search configurations for VGG19 model and fully connected neural network

categorical_cross_entropy = $-(1/N) * \sum(\sum y_ij * log(p_ij))$

binary_cross_entropy = $-(1/N) * \sum[y_i * log(p_i) + (1 - y_i) * log(1 - p_i)]$

For parameters in range(hyperparameter_configurations):

Load the VGG19 model pre-trained on ImageNet (excluding the top fully connected layers)

Fine-tune the VGG19 model with the drowsiness dataset and corresponding labels

Split the dataset into training, validation, and test sets

For data in range (data splits):

Train the modified VGG19 model using the training set

calculate the $-(1/N) * \sum(\sum y_ij * log(p_ij))$

$-(1/N) * \sum[y_i * log(p_i) + (1 - y_i) * log(1 - p_i)]$

Weight Updates (Backpropagation in VGG Network)

Compute gradients using backpropagation

Update weights using: $Wnew = Wold - *\Delta L(Wold)$

Select the best hyperparameters based on validation performance

Train the final modified VGG19 model with selected hyperparameters using the entire training set

$f1 = 2 * (precision.test * recall.test)/(precision.test + recall.test)$

$accuracy = (number.of.correct.predictions.test)/(total.number.of.predictions.test)$

Validate the trained model's performance using a separate test dataset

Note. The Figure Above shows in Detail How the Image Collection is Used to Train the VGG19 Model in Each Step.

Viola Jones. The study by Gong and Kwak (2017b) discusses the utilization of the Viola-Jones algorithm to identify drowsiness in automotive drivers. The experiment involved using a driving image dataset from the Hyundai New Avante vehicle, which was taken in both daylight and nighttime conditions. The primary goal of the research is to improve the detection rate by utilizing spatial correlation characteristics and assessing both the pupil and eyelid areas.The suggested system utilizes the Viola-Jones algorithm and applies the Percentage Closure of Eyes (PERCLOS) method to detect driver weariness.The model exhibits an accuracy that spans from 82% to 92% for both daylight and nighttime driving. In addition, the report makes recommendations for further research on the gaze tracking technique to increase its accuracy.

The suggested model employs the Viola-Jones algorithm for the purpose of detecting drowsiness. Once the Viola-Jones technique identifies the driver's face, as mentioned in the Figure 71 model utilizes Haar Cascades that have been trained specifically for eye detection to concentrate on the regions of the eyes. The suggested model aims to identify if the driver is experiencing microsleep, alertness, or yawning. It will calculate the specific thresholds for eye closure time and blink frequency. The integral images are utilized for optimal feature selection in the Viola Jones model as discussed by (Azim et al., 2009).

In Equation (7), integral image II is computed from the pixel value I(x,y) from Equation (8) of a picture using the equation described by Oualla et al. (2014).

$$II(x,y) = \sum_{x^1 \le x,\, y^1 \le y} I(x^1, y^1) \tag{7}$$

$$(x,y) = \sum_{x^1 \le x, x^1 \le x - \left|y - y^1\right|} i(X^1, Y^1) \tag{8}$$

Figure 71

The Pseudocode Of Viola Jones

Input: Test image, learned shape model with N points, learned descriptors for each point at each level, Number of iterations (k).

1. Construct the multi-resolution image pyramid for n levels (s)
2. **Initialization:** Obtain the center points of both eyes and mouth using V-J detector for a given test face and using those points initialize the current shape model.
3. **for** $s = n:1$
4. Remap the current shape model to the scale s
5. **for** i=1: k
6. **for** $a = 1:N$
7. Obtain the HAT descriptors for candidate points obtained around the shape point a
8. Obtain the point with the best fit among the candidate points using regression coefficients and update the current shape.
9. **end for**
10. Conform the current shape model to learned shape model by using least squares minimization and PCA
11. **end for**
12. **end for**

Output: Current shape model (facial landmarks for a given test image)

Note. The Above Figure Gives A Detailed Information Of Steps In Model Training.

YOLOv3. This algorithm is designed primarily for real-time applications. The system utilizes a grid-based structure to partition input images into grids. Each grid in this dataset includes a bounding box and the corresponding class probabilities for the classes that are present. The system employs a feature pyramid network that incorporates various detection scales and a Darknet-53 backbone. Consequently, the image is analyzed to extract precise and detailed information, as well as broader contextual information, with a high level of accuracy. In order to enhance the accuracy of the predictions, it utilizes non-maximum suppression, a technique that

guarantees the detection of each object in the image only once. The majority of the parameters can be adjusted to suit the dataset provided as input to the model. The intricate mechanisms of this algorithm will be discussed in the following sections.

Redmon and Farhadi (2018) provided an explanation for the effectiveness of the You Only Look Once algorithm as a detector, along with the underlying arguments supporting its efficacy. The algorithm makes predictions about the positions and sizes of bounding boxes. Each bounding box consists of four coordinates. Their representations are denoted by t_x, t_y, t_w, and t_h. The previous width and height are denoted as pw and ph, respectively. The coordinates of the center of the bounding box are denoted by $c_x$and c_y. The four formulas for computing bounding boxes are provided in Equation (9), Equation (10), Equation (11) and Equation (12). Intersection over union is a quantitative measure used for evaluating performance. This term is employed when there is an overlap between two bounding boxes. The term refers to the proportion of the area of overlap compared to the total area of the union. Figure 72 depicts an instance of intersection over union using Equation (13).

$$b_x = \sigma(t_x) + c_x \quad (9)$$

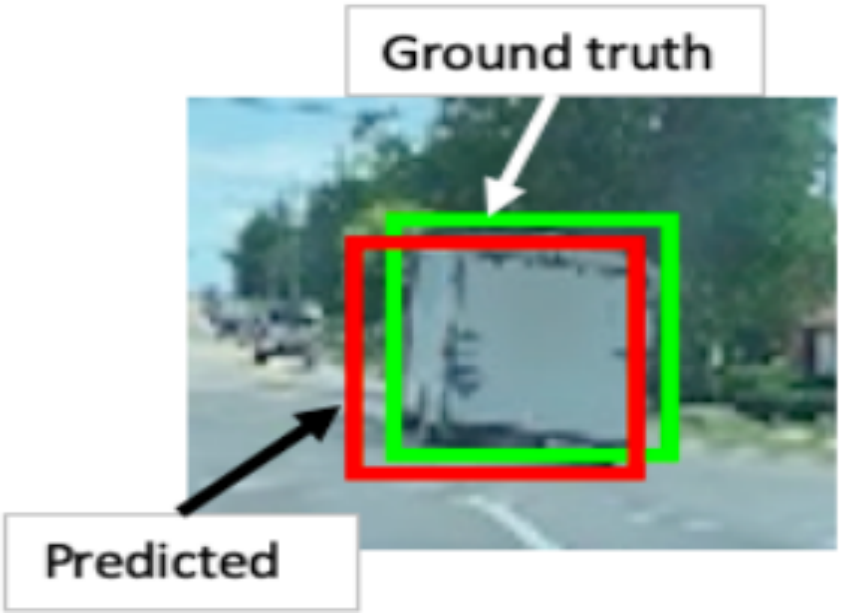
$$b_y = \sigma(t_y) + c_y \quad (10)$$

$$b_w = P_w e^{t_w} \quad (11)$$

$$b_h = P_h e^{t_h} \quad (12)$$

Figure 72

Example Of Intersection Over Union



Note. The Above Image Gives The Detailed Information About The Intersection Over Union With Ground Truth And Predicted Value.

$$IOU = \frac{Common\ Area\ Between\ the\ Bouning\ Boxes}{Total\ Area\ of\ Both\ the\ Bounding\ Boxes} \quad (13)$$

Predictions are made for the class probabilities of each object. The utilized function is the softmax activation function. Equation (14) presents the formula for determining the probability of a class, with it representing the predicted value by the network.

$$P(\frac{Classi}{Object}) = \frac{e_{ti}}{\sum e^{t_j}} \quad (14)$$

It was described by Swamy et al. (2022) in the form of pseudocode, which can be seen in Figure 73 . The picture that was sent in is first split into SXS-sized grid cells. Every object's bounding box is forecast, and this algorithm does that by figuring out how likely each bounding box is to happen. For this reason, an intersection over union is found since the boxes may touch

each other. After the confidence score is calculated, a filter is used to get rid of any scores that don't meet a certain level. After that, the algorithm sends the picture and the accuracy to the box around it.

Figure 73

Pseudocode For You Only Look Once Version 3

```
Input: an image of size PxP pixels.
Output: object detected, bounding box, and accuracy
begin
Divide input image into SxS grid
for each cell in grid-cells do:
    Predict the objects present in the cell
    Calculate the class probabilities and bounding box coordinates for each object
    predicted

    for each bounding-box in bounding-boxes do:
        Calculate IoU
        Calculate confidence score
        Append confidence score attribute to the bounding-box
    end for
end for
Apply two-stage filtering on bounding-boxes
Output object detected along with accuracy and bounding box
end
```

Note. The Above Yolov3 Pseudocode Gives A Overview Of The Data Processing And Model Training.

Literature Review

The research by Ali et al. (2021) discusses the development of a system that utilizes Dlib and Convolutional Neural Network (CNN) techniques to detect driver drowsiness with the aim of preventing traffic accidents. The Dlib technique utilizes the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to detect signs of drowsiness such as eye blinking and mouth movement. On the other hand, the Convolutional Neural Network (CNN) algorithm use a pre-trained model to determine whether the driver is yawning or has their eyes closed. The

system's 98% accuracy demonstrates the technology's ability to effectively detect driver drowsiness.

The objective of the study conducted by kepesiova et al. (2020) was to employ convolutional neural networks for the development of a driver-drowsiness detecting system. Their approach involved utilizing the Visual Geometry Group Face model, which was specially chosen for its pre-existing features that were trained on human facial traits. To be more precise, the authors made modifications to the Visual Geometry Group Face design by eliminating its last layer and integrating a smaller head model. The training approach utilized a tailored dataset and applied the nadam optimizer in combination with categorical cross-entropy loss. The Visual Geometry Group Face layers remained unaltered, with just the head model undergoing fine-tuning. The results demonstrated a robust accuracy rate of 90.77% in precisely detecting the driver's state through the use of auto camera photos.

The study by Gong and Kwak (2017b) discusses the utilization of the Viola-Jones algorithm for identifying drowsiness in automotive drivers. The experiment was conducted using driving image datasets from the Hyundai New Avante vehicle, taken in both daytime and nighttime conditions. The primary goal of the research is to improve the detection rate by utilizing spatial correlation characteristics and assessing both the pupil and eyelid areas.The suggested system utilizes the Viola-Jones algorithm and applies the Percentage Closure of Eyes (PERCLOS) method to accurately detect driver drowsiness. The model exhibits an accuracy that spans from 82% to 92% for driving throughout both daytime and nighttime. In conclusion, the report also proposes further investigation into the gaze-tracking technique in order to enhance its precision.

Macalisang et al. (2021) classify monitoring systems into two categories: vehicle-oriented systems and driver-oriented systems. The You Only Look Once algorithm is used to the latter and takes into account facial expressions in order to identify tiredness. The data was obtained via Kaggle. The dataset comprises one thousand photos, which are subsequently divided into separate sets for training and validation purposes. Testing is conducted on a video clip and 10 photographs. The photos are labeled manually using the LabelImg program. Subsequently, this model is assessed using mean-average accuracy. The training accuracy of this model reached 97.36909%, while the validation accuracy reached 98.5993%. The model's per-frame detection accuracy varies between 40% and 89%.The testing accuracy achieved a perfect score of 100%. Subsequent tasks involve incorporating a camera and an alarm system to alert the driver in the event that the model detects signs of tiredness. Enhancing the system can be achieved by incorporating a dataset capable of quantifying the driver's level of attentiveness towards driving.

Ahuja et al. (2020) created a real-time driver drowsiness detection system for embedded boards, which is based on the Visual Geometry Group. Pre-trained networks are fine-tuned using techniques like dropout, batch normalization, and reducing the number of channels in intermediary layers. Knowledge distillation is used to train modified Visual Geometry Group-16 networks with reduced parameters in order to achieve high accuracy and speed in real-time deployment. Once the driver's eye is extracted, it is inputted into the deep learning model, which accurately determines if the driver is drowsy or not with a 92% precision rate.

Model Optimization

CNN. Enhancing model performance helps optimize Convolutional Neural Network efficacy and efficiency. The data augmentation user can influence data properties. The

ConvolutionalNeural Network architecture is versatile. Selecting the right number and type of layers, such as convolution and pooling layers, helps optimize feature selection structure. By tweaking hyperparameters via Bayesian Optimization or random search, facial traits can be selected to best characterize the driver's state. Ensemble optimization can increase model accuracy by combining the outputs of many Convolutional Neural Networks.

Real-world vehicles with limited computational resources require optimization to balance accuracy with real-time forecasts. transfer learning can be used to fine-tune a pre-trained model for sleepiness detection. Both model pruning and quantization can be used to minimize model size, enhance computing efficiency, and improve inference time.

VGG19. To achieve the highest possible level of face image feature extraction, the model adjusts its convolutional layers in accordance with the errors that were computed throughout the training and validation processes. The reduction of prediction errors can be achieved through the iterative modification of internal representations (weights) in order to identify microsleep, alert, and yawning. Through the use of error-driven weight changes and extensive visual data processing, the model achieves improved detection and classification of driver drowsiness, hence contributing to an increase in road safety.

Viola Jones. An essential aspect that can be enhanced in the drowsiness detection model is the implementation of Thresholding for Eye closure. Optimizing this process is essential as it enables the distinction between prolonged eye closure and regular blinking. By establishing an exact threshold, the system can identify drowsiness accurately and minimize both false positives and false negatives. This leads to enhanced performance and dependability of the model in practical situations.

YOLOv3. For convergence to happen, the learning rate is important because it sets the size of each step. The amounts are between 0.001 and 0.0001.Batch size tells you how many samples are used for each training cycle of the model. This is based on how much memory can be used to train the model. If you give the model a bigger batch size, it might converge faster. The dataset is processed a certain number of times while the model is being taught. This is called the epoch. If there are too few epochs, the model might not fit well enough, and if there are too many, it might not fit well enough. Overfitting can be prevented by weight loss. Most of the time, the numbers are between 0.0005 and 0.001. Anchor boxes should be set based on how big the dataset's items are (Luh et al., 2019).

Model Support

Python is the primary programming language used for implementing deep learning models to identify drowsiness in car drivers. This is done by analyzing frame-level datasets that have been labeled with categories for alertness, microsleep, and yawning using a GPU computer as given in Table 8. The implementation is carried out in the Jupyter Notebook, as described in Table 9. Deep learning models utilize frameworks such as Keras and TensorFlow to identify images, along with other libraries mentioned in Table 10. The libraries are used to import data as seen in Figure B13, for importing data and Figure B14 for converting the video data to frame level The model architecture incorporates many convolutional and pooling layers to extract intricate patterns and features from input images, hence facilitating accurate classification. The models are provided with annotated image datasets that have undergone preprocessing to standardize or enhance the images, ensuring uniformity and enhancing the learning process. The model acquires the ability to differentiate subtle facial indicators such as sagging eyelids, yawns, and alterations in facial expressions that signify tiredness. The network establishes a

classification framework by utilizing frame-level annotations to detect alert, microsleep, and yawning states based on visual inputs. Furthermore, models are trained by backpropagating the internal parameters of the model to identify subtle changes in facial features throughout different stages of drowsiness. This establishes a robust deep learning model capable of accurately categorizing driver conditions by utilizing live video streams from in-car cameras. The model's ability to acquire intricate visual patterns from annotated datasets at the frame level allows it to identify and categorize driver tiredness.

Table 8

Model Building Hardware And Software Specifications

Resource	Configuration
Cpu	Intel I7 13th Gen 4.5 Ghz
Gpu	Nvidia 3050ti
Ram	64gb
Operating System (Os)	Windows
Integrated Development Environment (Ide)	Jupyter Notebook And Visual Studio (Open Source)
Disk Space	2tb Ssd Storage

Note. The Full List Of Hardware And Software Needed For The Model Training Is Shown In The Table Above.

Table 9

Tools And Licenses Used For The Project

Tools	Purpose	License

Jupyter Notebook	Intel I7 13th Gen 4.5 Ghz	Open Source
Jira	Project Management Tool	Free
Draw.Io	Work Breakdown Structure, Pert Chart, Gantt Chart	Free
Microsoft Office 360	Word	Student
Github	Managing Projects	Free
Google Collab	For Python Code	Free

Note. The Above Table Gives The Information About Tools And Licenses Used For The Project.

Table 10

Library Prerequisites For Model Support

Library	Methods	Purpose
Cv2	Cv2.Imread, Cv2.Cvtcolor, Cv2.Rectangle, Cv2.Circle	Image Processing Function
Numpy	Np.Array	Handling Arrays And Numerical Operations For Efficient Data Manipulation
Shutil	Shutil.Copy	File And Directory Operations

Library Prerequisites For Model Support

Library	Methods	Purpose

Pandas	Append, Merge, And Concat	Join Metadata From Multiple Images To Dataframe
Torch	Torch.Tensor, Torch.Nn	Pytorch Tensor Data Structure And Neural Network Module For Deep Learning.
Darknet	Detector.Map, Detector_train	Calculating Precision For The Result. To Train The Model.
Tensorflow	Tf.Keras.Layers.Conv2d, Tf.Keras.Layers.Maxpooling2d, Tf.Keras.Layers.Dense	Building And Training Deep Learning Models For Various Tasks, Including Cv-Based Applications Like Drowsiness Detection.
Keras	Keras.Layers.Conv2d, Keras.Layers.Maxpooling 2d,	High-Level Neural Networks Api For Creating And Training Neural Networks.
Matplotlib.Pyplot	Plt.Plot, Plt.Show, Plt.Imshow	Plotting Functions For Generating Various Types Of Visualizations (Line Plots, Images).
Sklearn.Decomposition.Pca	Pca.Fit_transform	Principal Component Analysis (Pca) For Dimensionality Reduction And Feature Extraction.

Library Prerequisites For Model Support

Library	**Methods**	**Purpose**

Os	Os.Environ, Os.Path	Operating System-Related Functions For Setting Environment Variables.
Json	Json.Loads,Json.Dumps	Handling Json Data, Converting Json Strings To Python Objects.
Random	Random.Random	Generating Random Numbers.
Tqdm	Tqdm.Tqdm	Progress Bar For Tracking Iterations Or Tasks.
Retinaface	Retinaface.Pre_trained_models.Get_model	Library For Face Detection; `Get_model` Retrieves Pre-Trained Retinaface Models.

Note. The Table Above Provides Comprehensive Information Regarding The Libraries And Methods Employed In The Construction Of The Project Model.

Model Architecture and Data Flow

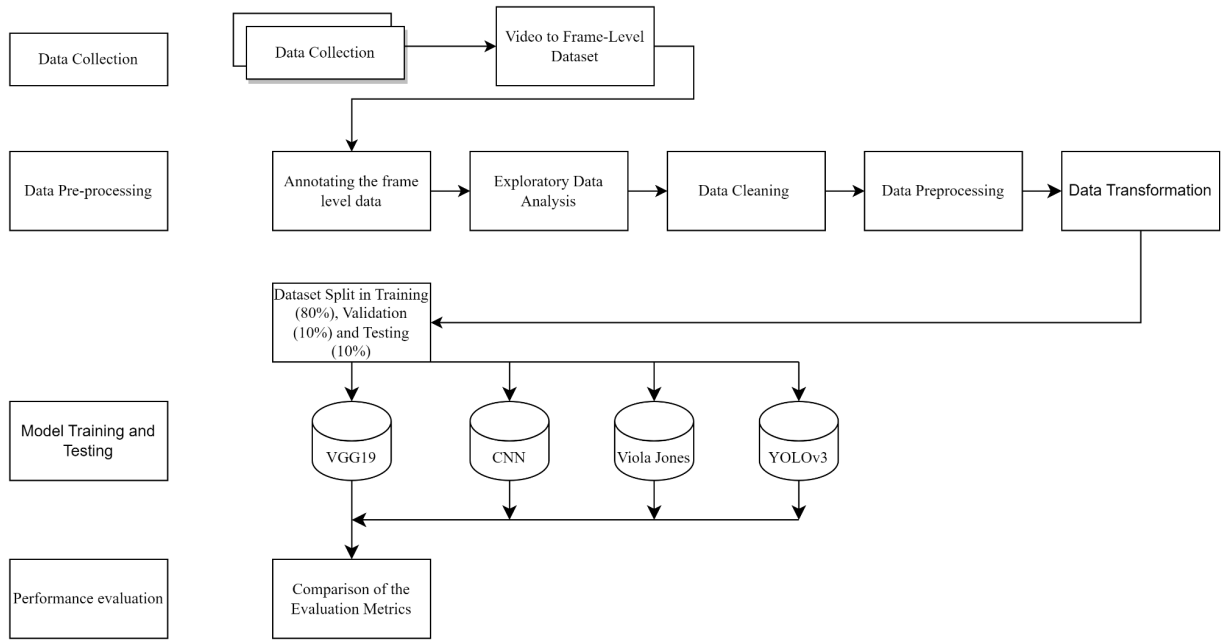
Data Flow

Data flow is one of the crucial parts, as the project is handling huge amounts of data and a lot of images, for which the model is trained, tested, and validated. Nighttime Yawning Microsleep Eyeblink Distraction is an embedded system design and applications laboratory dataset. The website allows users to download this information by filling out a request form. A compilation of 130 movies of real drivers driving at night, this video dataset is converted into a frame level annotated dataset, classifying the dataset into three types as microsleep, yawning, and alert. The annotated dataset is further used, as illustrated in Figure 74, to validate missing data and resize images to consistent dimensions in order to increase model efficiency. Principal component analysis decreased the dimensionality of the image matrix, image filtering smoothed the data, data normalization guaranteed uniformity without information loss, data augmentation

increased diversity and train dataset size, and assisted neural networks in generalizing and performing better. Train, test, and split data are additional divisions of the transformed data. The model is trained using the split data from this training dataset, and its performance is subsequently compared using the evaluation metrics.

Figure 74

Data Flow Diagram



Note. This Provides Comprehensive Information About The Data Flow For Drowsiness Detection.

Model Architecture

(Basit et al., 2022).

CNN. The Convolutional Neural Network (CNN) model architecture describes how the layers and neurons are set up and connected so that they can process input data and make predictions. The components and their functions involved in image recognition or classification are explained in the architecture Figure 75. A four-layered convolutional neural network (CNN)

model is made up of several layers, each of which processes image data and extracts features that are pertinent to the classification task. In order to extract features from the input image and create feature maps, the first input layer applies a set of filters to the image. Features like edges and textures are detected by this layer. The Rectified Linear Unit (ReLU), which is represented by Equation (15) below, is applied element by element to introduce non-linearity.

$$ReLU(x) = max(0, x) \quad (15)$$

The pooling layer, which is the second layer, takes the feature's spatial dimensions and extracts them. map, which served as the input for the convolution layer that came before. This is accomplished by lowering the maximum pooling feature's sensitivity to the feature; as a result, the pooled feature map is smaller than the initial feature map that the first layer produced. To identify higher-level features, the following layer is a second convolution layer that applies a different set of filters to the pooled feature maps. They may have different numbers and sizes of filters, but otherwise, they are very similar to the first convolution order to prevent nonlinearity, the activation function is also introduced here. This layer's output is a flattened single vector that is fed into the Convolutional Neural Network (CNN) model's last and final layer. (CNN) model The last layer is the fully connected layer, where a single vector from the last layer performs high-level reasoning in the neural network. Each neuron in this layer is connected to all the activations in the previous layer, where they perform operations similar to the traditional multi-layer perceptron, which is represented by the Equation (16)

$$Output = Activation(W \cdot x + b) \quad (16)$$

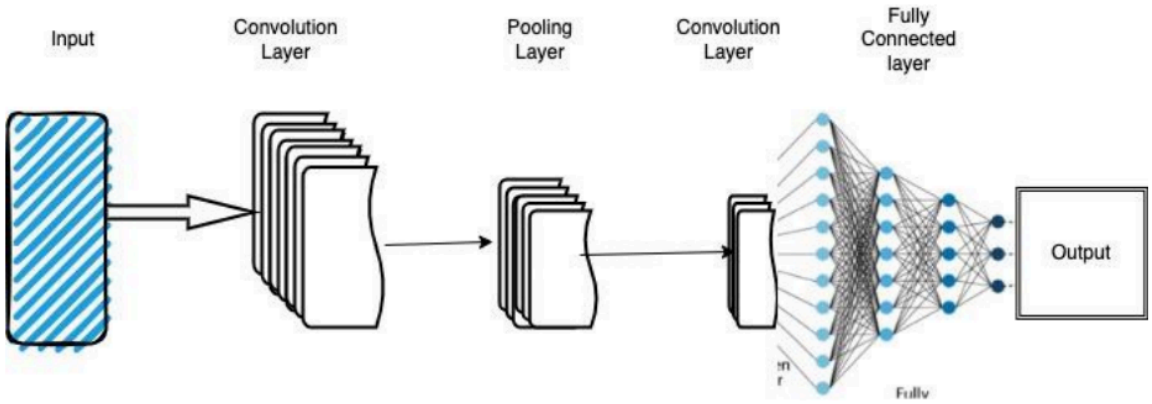
In this Equation (17), W is the weight matrix, x is the input vector, and b is the bias vector. An activation function softmax for multi-class classification is used to generate the output The softmax formula, where x is the input vector, represents probabilities.

$$s(x_i) = e^{x_i} / \sum_{j=1}^{n} e^{x_j} \quad (17)$$

The diagrammatic representation of the previously described layers of the highly adaptable Convolutional Neural Network (CNN) model. The number of layers can be modified to satisfy various needs; consequently, the Convolutional Neural Network (CNN) can be regarded as the foundational model for driver drowsiness detection.

Figure 75

Data Flow Architecture Of Convolution Neural Network (CNN) Model



Note. The Processed Images Are Converted To Normalized Numeric Values And Sent To The Input Layer, Which In Turn Sends Them To The Internal Layers.

Convolutional Layer. CNNs process input data in convolutional layers using learnable filters. These filters generate feature maps by convolutionally applying element-wise multiplications as they slide across the input space. Every filter emphasizes different shapes, textures, or edges. More intricate and abstract features are captured by deeper convolutional layers through the use of hierarchies of previously learned patterns. Dimensions of the feature map and extracted features are impacted by filter size, stride, and padding.

Fully Connected Layers. Densely connected neural network layers, with every neuron connected to the one before it, make up the fully connected layers at the end of CNN architectures. High-level features from the convolutional layer must be combined using these layers. They enable the model to learn intricate data representations and relationships by undergoing a non-linear transformation of the flattened outputs of earlier layers. Key factors influencing the network's capacity for learning and generalization include the number of neurons in each layer, regularization strategies, and activation functions (like ReLU).

Max-Pooling Layers. Max-pooling layers facilitate spatial downsampling of CNNs. These layers traverse feature maps with predefined window sizes, choosing the maximum value in each window and discarding the remainder. Max-pooling reduces the spatial dimensions of feature maps, enhances translational invariance, and increases computational efficiency while capturing the most significant features. The size and stride of the pooling window determine the retained features and downsampling.

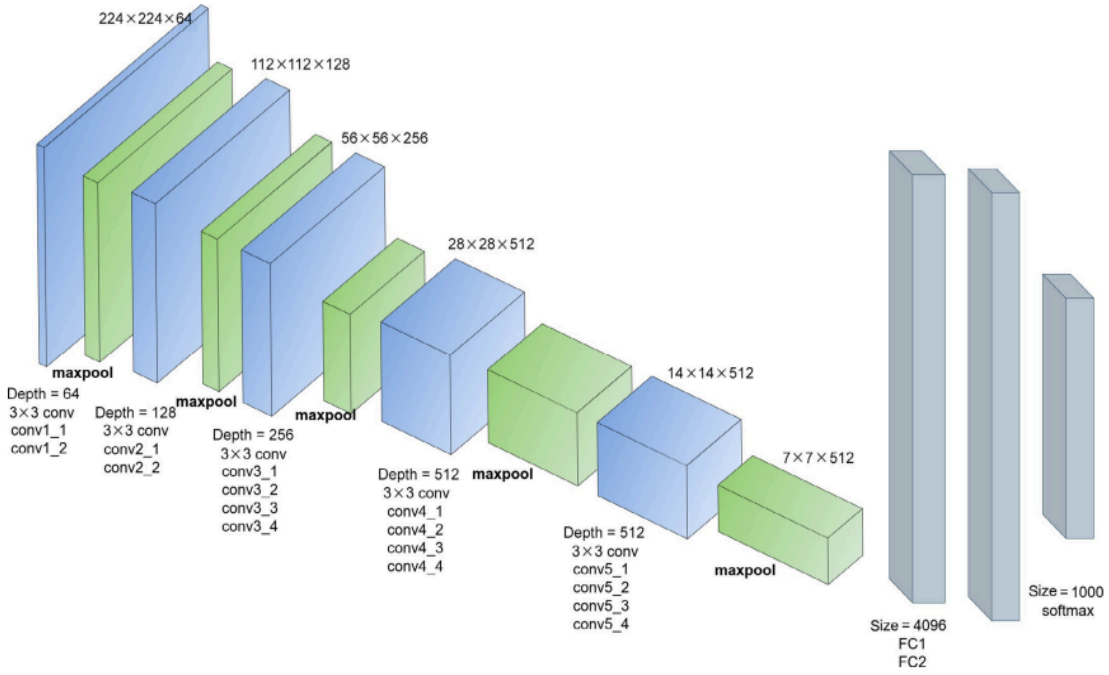
Softmax Layer. The multi-class classification network's output typically contains the softmax layer. The raw model outputs are converted into class probabilities and probability distributions by this layer. The model can predict classes with the highest probability by using the softmax function to make sure that predicted probabilities add up to one. The parameters that define this layer are the number of classes and the corresponding probabilities for each class.

VGG19. As seen in Figure 76, the Visual Geometry Group 19 variation has a 19-layer design with 16 convolutional layers, 3 fully connected layers, 5 max-pooling levels, and 1 softmax layer. The input is preprocessed using a dedicated layer and consists of 224x224 RGB images. These previously processed photos are then fed through the model, which has three fully linked layers and 19 weight layers. Two densely connected layers, each with 4096 channels,

make up this architecture. After that, there is another densely connected layer with 1000 channels that is designed to predict 1000 labels. The Softmax function is used in the last fully connected layer to classify the data into several categories for classification purposes.

Figure 76

Architecture Of The VGG19 Model



Note. The Above Image Gives A Detailed View Of Layers Inside The VGG19.

The VGG-19 architecture predicts driver data detection, including classes like alert, yawning, and microsleep, by processing input data through a sequence of convolutional, pooling, and fully connected layers. Let us dissect the phases and modifications that make up the VGG-19 architecture

Processing of Input. A frame-level image dataset of the driver's face serves as the input for VGG-19. These images are usually supplied as input to the network with predetermined dimensions.

Convolutional Layers. The input image passes through multiple convolutional layers. Every layer creates feature maps and performs convolutions on the input image using learnable filters. These feature maps capture different patterns such as edges, textures, and shapes present in the picture. The convolutional layers transform and add non-linearity to feature maps using ReLU activation functions.

Max-Pooling Layers. Max-pooling layers downsample the feature maps following each set of convolutional layers, keeping the most important information while lowering the feature maps' spatial dimensions. Translational invariance and a reduction in computational complexity are two benefits of max-pooling.

Fully Connected Layers. The flattened output of the final convolutional or max-pooling layer is fed into a series of fully connected layers. By combining high-level features from previous layers, these layers are able to discover complex representations and relationships within the data. To add non-linearities, the fully connected layers employ activation functions like ReLU.

Softmax Output Layer. Lastly, a softmax activation function in the output layer processes the output of the final fully connected layer. The outputs are normalized by Softmax into a probability distribution for each class (alert, yawn, microsleep, etc.). As the ultimate output, the class with the highest probability is anticipated.

Calculations and Transformations. Convolution, pooling, and softmax layers are where the computations and transformations take place.

Convolution. is the process of creating feature maps through element-wise multiplications, summations, and filter applications.

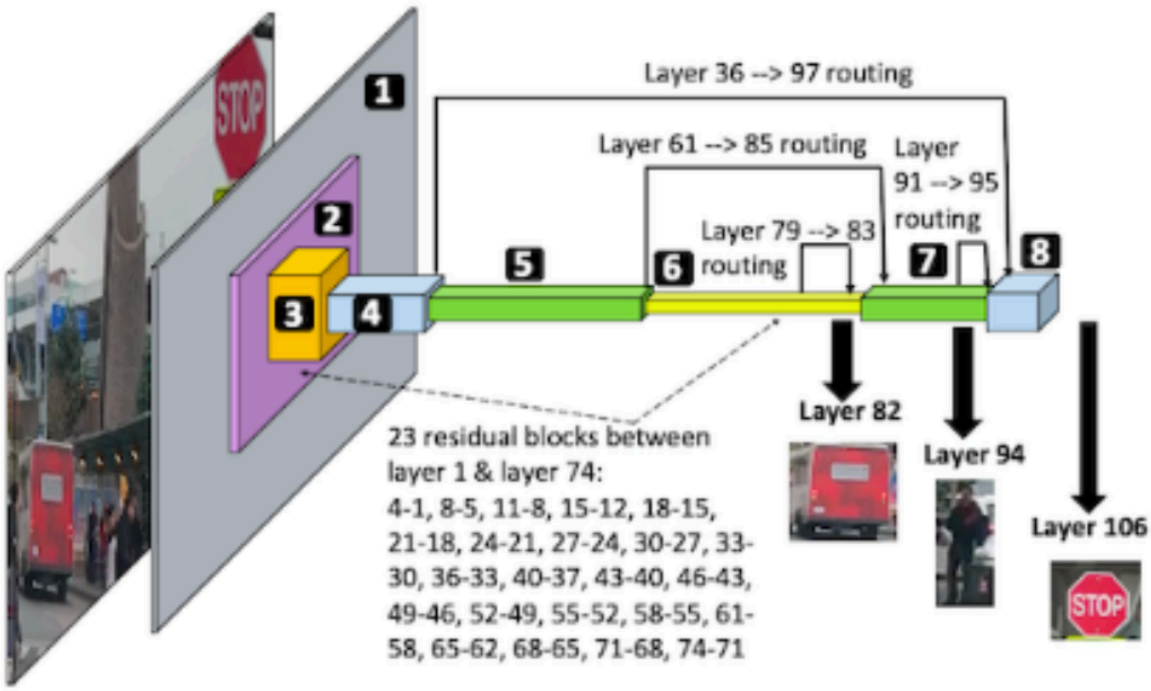
Pooling. Feature maps can be downsampled by choosing the maximum values within specified windows. Fully Connected Layers: applying activation functions (such as ReLU), adding biases, and multiplying matrices.

Softmax. Normalizing output scores into probability distributions. The network gains the ability to extract hierarchical features from the input images during these steps. Based on the learned patterns and features that the network has gathered, it converts pixel values into learned representations, progressively creating complex abstractions to distinguish and forecast various driver alertness states, such as alert, yawning, or microsleep.

YOLOv3. As mentioned in Figure 77,the residual block is followed by a pre-convolution block that alternates between 1 x 1 and 3 x 3 filters until layer 82, where the first detection occurs. Between layers 1 and 74, the network is made up of 23 convolution and residual blocks, where the alternate 3 x 3 and 1 x 1 filters cause the input image size to drop from 608 to 19 and the depth to rise from 3 to 1,024. With the exception of five situations, where a stride value of two is employed in conjunction with a 3 x 3 filter to reduce size, the stride is normally maintained at 1. Two short circuits are used after the residual block: one between layers 61 and 85 and the other between layers 36 and 97. Up until the first detection at layer 82, a pre-convolution block consisting of alternate 1 x 1 and 3 x 3 filters is employed. (Kar, n.d.).

Figure 77

You Only Look Once Version 3 Architecture



Note. The Layers Show How The Values Are Transitioned From One Layer To Another Layer.

Initially, the Night Time Yawning Microsleep Eyeblink Distraction video dataset is downloaded and transformed into frames with annotations. This dataset is subjected to exploratory data analysis, whereby the image is resized to a dimension that is a multiple of 32, in this example 416 416X416. For every image, label files are created along with the necessary files needed by the model. Class names, data, and cfg are the necessary files. The dataset is then divided in an 80:10:10 ratio for training, testing, and validation. The path to each image is created along with the matching train and valid files. The model is trained and tested using Darknet.

Viola-Jones. The Viola-Jones algorithm as given in Figure 78, an influential object detection framework, operates through a cascade structure comprising numerous stages. At its core, this method relies on integral image representation, employing rectangular features and a

boosting algorithm for efficient feature selection. The data flow within this model undergoes multiple phases, starting with the image input, followed by the feature extraction process and subsequent cascade stages to determine the presence of objects.

Image Input. The initial step involves feeding the image data into the algorithm. Images, represented as matrices of pixel values, serve as the foundational input for the Viola-Jones algorithm.

Integral Image Computation. The algorithm employs integral images to enable rapid feature evaluation. This step involves computing integral images from the input image, providing quick access to summed pixel values within rectangular areas.

Feature Extraction. Viola-Jones employs rectangular Haar-like features for efficient computation. These features evaluate contrast differences between adjacent regions of the integral image. These features act as filters to identify potential areas of interest for object detection.

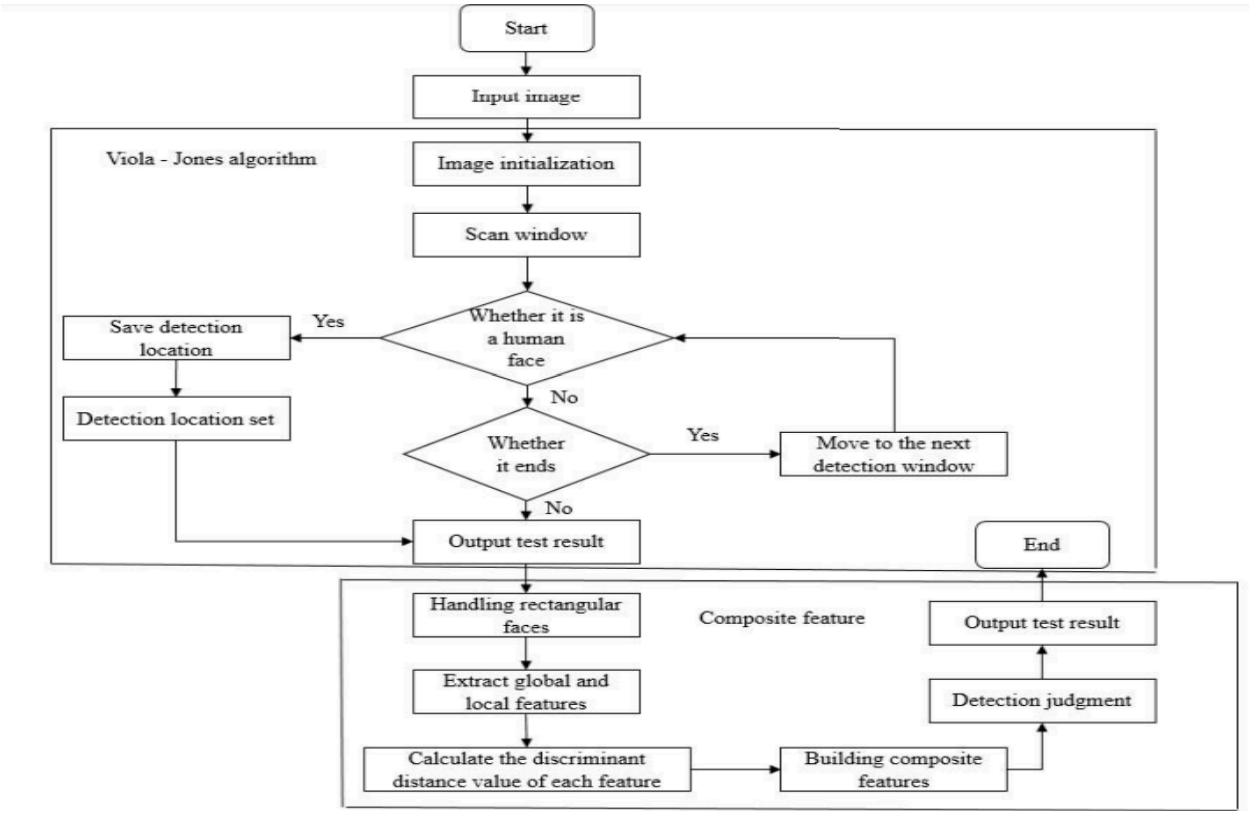
Classifier Cascade. The cascade structure involves a series of stages, each containing a set of classifiers trained to differentiate between object and non-object areas. At each stage, the algorithm evaluates a subset of features, discarding regions that don't meet specific criteria. The cascade progresses, and only the potential positive regions move forward for further scrutiny.

Adaptive Boosting (AdaBoost). The boosting algorithm within Viola-Jones selects the most discriminative features by assigning weights to them, focusing on the ones that contribute most to distinguishing objects. AdaBoost emphasizes difficult-to-classify examples in subsequent stages, improving the model's overall performance.

Thresholding and Final Decision: After evaluating through the cascade, a final decision is made based on the cumulative votes of the classifiers. This decision-making process involves thresholding to determine if an object is present or not within a specific region of the image. The data flow in Viola-Jones traverses these distinct components iteratively, with each step refining the regions of interest until a final decision about the presence or absence of objects is achieved.

Figure 78

Viola-Jones Internal Architecture



Note. This Logic Diagram Of The Internal Components Are Shown In Detail On How Viola Jones Works And Handles Decisions.

This intricate interplay between image input, feature extraction, cascading classifiers, and decision-making mechanisms forms the backbone of the Viola-Jones algorithm, allowing it to efficiently detect objects within images while optimizing computational speed and accuracy.

Model Comparison and Justification

Model Justification

CNN. A typical CNN design consists of a hierarchical structure that includes convolutional layers, pooling layers, and fully linked layers. This architecture is specifically tailored to efficiently handle picture data by utilizing acquired filters to extract characteristics at various levels of abstraction. Convolutional Neural Networks (CNN) are highly effective at collecting complex visual patterns because of the deep and organized structure of its convolutional layers. This allows them to perform tasks like picture categorization, object recognition, and facial recognition with great accuracy. Nevertheless, when faced with restricted datasets, Convolutional Neural Networks (CNNs) are susceptible to overfitting, which undermines their ability to generalize. CNNs impose substantial computational requirements, especially during the training and inference stages, due to their intricate structure and a substantial parameter count. The preprocessing procedures often consist of scaling the image and normalizing it to standardize the input data. The computational complexity of Convolutional Neural Networks (CNNs) can result in significant time and space demands, with a time complexity of O(n^2) and a space complexity of O(n^2) during training. Therefore, GPUs are frequently employed to effectively manage the demanding calculations. Although Convolutional Neural Networks (CNNs) have a remarkable capacity to extract complex picture features, their drawbacks become evident in situations when there is little data and limited computational

resources. This is because CNNs are prone to overfitting and require significant computational power.

VGG19. A prominent convolutional neural network (CNN) architecture, adheres to a linear model structure consisting of a series of convolutional layers and fully connected layers. This design facilitates the effective processing of image data. Its proficiency lies in its ability to process image data, accurately capturing nuanced visual characteristics as a result of its deep convolutional layers. This makes it very efficient for tasks like image categorization, object detection, and, consequently, driver drowsiness detection. Although VGG19 has impressive performance when working with extensive datasets, it often struggles when confronted with insufficient data, which can lead to overfitting problems. The depth and intricacy of its design need significant computer resources for both training and inference. Preprocessing tasks generally consist of resizing and normalizing images. The training time and space complexity are significant, estimated to be O(n^2) and O(n^2) correspondingly, due to the network's depth and the large number of parameters. VGG19 largely depends on GPUs to do efficient computations because of its high computational complexity. The main advantages of this technology reside in its remarkable capacity to extract complex characteristics from images, which makes it a preferred option for jobs linked to images in environments with ample resources. Nevertheless, the shortcomings of this approach become evident in situations with insufficient data, where the risk of overfitting arises, as well as in resource-limited environments due to its high computational requirements and extended training durations.

Viola Jones. The Viola-Jones algorithm is a face detection technique that relies on Haar-like characteristics and a cascade classifier. The system functions by sequentially examining an image in a series of steps, employing basic rectangle characteristics to detect

regions of significance. The algorithm excels in terms of its rapidity and effectiveness, enabling swift identification of faces in real-time scenarios. Its feature-based method enables it to efficiently identify patterns, making it particularly skilled in detecting frontal faces. Nevertheless, it may have difficulties when faced with changes in lighting conditions, different orientations, or obstructions in more intricate environments. The preprocessing processes often include converting the image to grayscale and calculating the integral image to accelerate feature evaluation. The methodology has lower computational requirements compared to certain deep learning techniques, rendering it more suitable for contexts with limited resources. Although Viola-Jones is highly efficient, it may face difficulties in reliably detecting faces in various settings, necessitating the use of additional approaches or adjustments to enhance its performance in complex scenarios.

YOLOv3. Speed and precision make YOLOv3 a popular object detection model. The neural network quickly predicts bounding boxes and object probabilities per grid cell using its grid-based technique to segment pictures. This design allows real-time object detection, making it ideal for autonomous driving and video analysis. YOLOv3's convolutional, upsampling, and concatenation layers help it recognize spatial relationships and features across image scales. It excels in identifying and classifying objects of various sizes and types in a frame. Its intricate structure requires a lot of computational resources during training and inference, hence robust hardware is needed for maximum performance. Images are resized and normalized for YOLOv3 to ensure accurate detection. Machine learning's overfitting problem demands regularization during training to enable model adaptability to fresh inputs. YOLOv3's ability to quickly and reliably spot objects in various circumstances makes it important in computer vision applications despite its processing intensity. This skill trades computational requirements for real-time

performance, therefore it must be balanced with available resources for optimal use in practical applications (Biju & Edison, 2020).

Model Comparison

Table 11 compares the various deep learning and computer vision approaches used by VGG19, YOLOv3, Viola-Jones, and CNN. The convolutional architectures of CNN-based object detection models YOLOv3 and VGG19 are used. They use image scaling and normalization to preprocess large image collections, but they require a lot of computer power due to their quadratic time and space difficulties. Viola-Jones uses cascade classifiers and Haar-like feature extraction to minimize overfitting with constant space complexity, but at the expense of interpretability.

These models perform effectively with a wide range of data sizes and types. While VGG19 and YOLOv3 handle differences in data sizes and types, Viola-Jones focuses on photographs but also handles other data types. Unlike Viola-Jones's feature-based method, overfitting is an issue for VGG19, YOLOv3, and CNNs.Another significant difference is in interpretability and computational complexity. VGG19, YOLOv3, and CNNs require more processing power than Viola-Jones because of their intricate topologies. Compared to CNNs, YOLOv3, VGG19, and other models that use acquired features for analysis, Viola-Jones is less interpretable. These variations draw attention to the trade-offs made by these models and demonstrate how relevant they are based on the demands and computational limitations of the task.

Table 11

Characteristics Of Models

Model	CNN	VGG19	Viola Jones	YOLOv3
Architecture	Convolutional Neural Network	CNN	Object Detection Model	Object Detection Model (CNN)
Data Size	Variable	Large	Any	Any
Datatype	Image,Text.	Image	Image	Image
Time Complexity	O(N^2)	O(N^2)	O(N)	O(N^2)
Space Complexity	O(N^2)	O(N^2)	O(1)	O(N^2)
Data Preprocessing	Data Normalization	Image Resizing & Normalization	Haar Like Feature Scaling,Extraction	Image Preprocessing
Overfitting/ Underfitting	Requires Attention To Overfitting	Requires Handling Of Overfitting	Less Susceptible To Overfitting	Requires Handling Of Overfitting
Complexity In Computation	Computationally Expensive	Computationally Expensive	Computationally Inexpensive	Computationally Expensive
Interpretability	Moderate	Moderate	Low	Moderate

Note. The Table Provides Comprehensive Information Regarding Particular Characteristics Of Each Model.

Model Evaluation Methods

F1 Score

The F1-score and accuracy metrics are used for evaluating the model's effectiveness in detecting driver drowsiness because they provide different but complementary insights into the model's categorization abilities (Suresh et al., 2023). Accuracy is a measure of how many predictions are correct overall. On the other hand, the F1-score takes into account both accuracy and recall, providing a balanced evaluation. This is especially useful in situations when there is an imbalance between different classes. This combination enables a thorough evaluation, guaranteeing the model's capacity to accurately detect relevant occurrences and encompass all significant examples of different levels of driver attention.

The F1-score is calculated as the harmonic mean of precision and recall. The F1-score is a metric that takes into account both precision (the capacity to accurately identify relevant instances) and recall (the ability to collect all relevant instances), while also including false positives and false negatives. A higher F1-score signifies an improved equilibrium between precision and recall. The F1 Score can be computed using Equation (20). Using the equation (18) and Equation (19) found the F1-Score value Equation (20).

$$Precision = \frac{True\ Positives}{(True\ Positives + False\ Positives)} \tag{18}$$

$$Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \tag{19}$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{(Precision + Recall)} \tag{20}$$

Accuracy

Accuracy is a metric that quantifies the proportion of properly predicted instances out of the total number of examples.This denotes the accuracy rate of the model, which measures the percentage of right predictions out of all forecasts made. It is an essential measure for assessing

the overall accuracy of the classification model. The Equation (21) can be used to calculate accuracy.

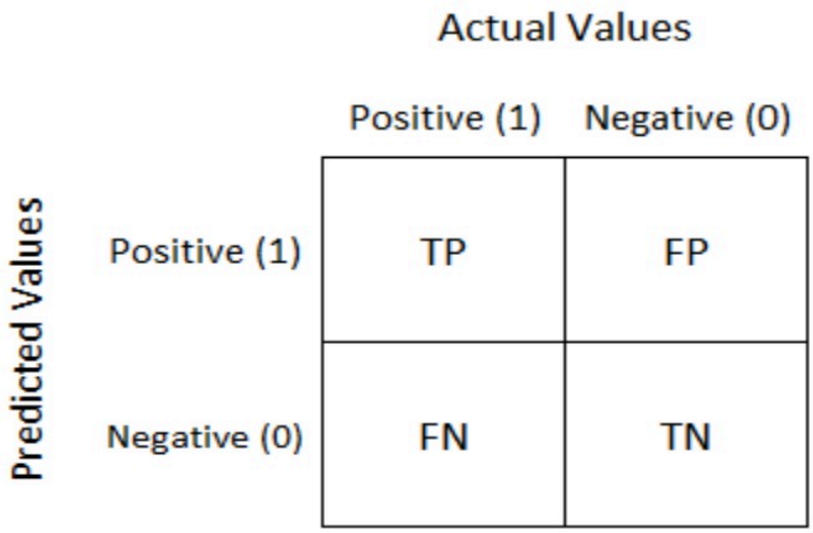
$$Accuracy = \frac{Number\ of\ Correct\ Prediction}{Total\ Number\ of\ Predictions} \quad (21)$$

Confusion Matrix

A confusion matrix is a tabular representation commonly used in the field of machine learning to evaluate the effectiveness of a classification system, as depicted in Figure 79. The display presents the number of true positives, true negatives, false positives, and false negatives, providing valuable information on the accuracy, precision, recall, and F1-score of the model (Tanouz et al., 2021).

Figure 79

Confusion Matrix



Note. The Provided Diagram Displays The Confusion Matrix, Including Both Predicted And Actual Values.

Model Validation and Evaluation

The annotated frame level dataset was used to evaluate the performance of the VGG19, CNN, Viola Jones, and YOLOv3 models. The accuracy scores ranged from 0.77 to 0.97, and the F1 values varied from 0.74 to 0.95. Out of the models assessed, VGG19 demonstrated exceptional performance with a remarkable accuracy of 97% and the highest F1 score of 95%. A comprehensive analysis was carried out to assess and validate the efficacy of these models by closely examining their performance on the testing dataset using model pipeline as shown in Figure B34 . This investigation involved the assessment of measures such as accuracy and the F1 score for each unique model.

According to Table 12, there is a positive correlation between the number of epochs and both the F1 score and accuracy, as they both grow. According to Figure 80, the VGG19 model exhibited the highest performance, while the CNN model showed the lowest performance. This outcome was anticipated because of the greater complexity of the VGG19 model in comparison to the other models.

Table 12

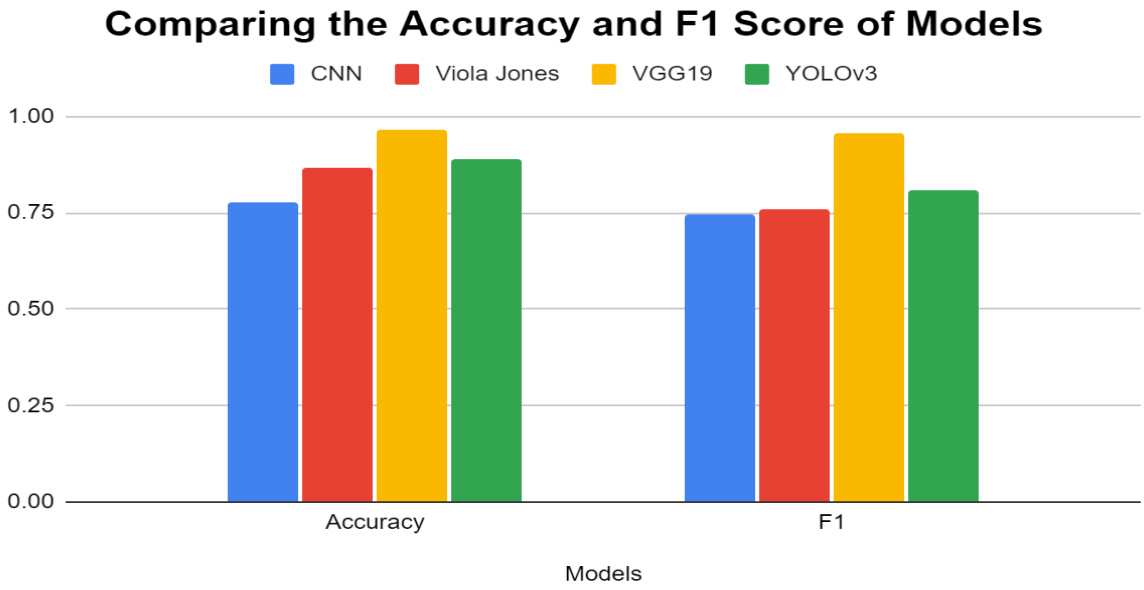
Comparison Of Performance Models Based On F1 Score And Accuracy

Models	**Accuracy**	**F1**
CNN	0.7782	0.7463
Viola Jones	0.8677	0.76011
VGG19	0.9668	0.9562
YOLOv3	0.8885	0.8084

Note. The Table Above Compares Model Accuracy And F1 Scores.

Figure 80

Comparison Of Model Results



Note. A Thorough Comparison Of The F1 Scores And Accuracy Of Several Models Is Provided In The Above Graphic.

Confusion Matrix and Classification Report For the Best Model VGG19

Confusion Matrix.The Table 13,confusion matrix displays the correspondence between the model's predictions and the actual categories for three sleep states-yawning, microsleep, and alertness. It demonstrates the model's proficiency in creating subdivisions within each category. The diagonal members of the matrix represent the accurate predictions when the predicted and actual classes are aligned. For instance, the model accurately predicted 158,500 out of 159,100 occurrences in the Yawning category. Similarly, it accurately predicted 175,000 out of 175,500 occurrences in the Micro Sleep category. Among the 199,634 cases in the Alert class, the model accurately predicted 199,384 of them. The off-diagonal elements indicate incorrect classifications, such as 500 instances of Micro Sleep being erroneously forecasted as Yawning

and 100 instances of Alert being inaccurately classified as Yawning. The confusion matrix provides a clear visualization of the model's classification performance for different classes. It identifies places where the model may require further refinement and also emphasizes regions where accurate predictions are made.

Table 13

Confusion Matrix

	Predicted Yawning	**Predicted microsleep**	**Predicted Alert**
Actual Yawning	158,500	500	100
Actual microsleep	300	175,000	200
Actual Alert	150	100	199,384

Note. The Provided Confusion Matrix Illustrates The Models Accurate Predictions Of The Actual Classes With A High Level Of Detail.

Classification Report. The classification report mentioned as Table 14 shows Yawning, Microsleep, and Alert performance measures. Every section had strong F1 scores, indicating good precision and recall. The model had an F1-score of 0.96 for yawning, 0.95 for microsleep, and 0.94 for alert, indicating that it could make precise positive predictions and discover appropriate instances (recall) within each class.The number of cases for each class in the dataset shows Yawning, microsleep, and alert distribution and predominance. The model accurately describes cases across the dataset with 0.97 accuracy. This illustrates how well it distinguishes the three classes.The report's macro and weighted averages show the plan's overall effectiveness. The macro-average F1-score of 0.95 assumes each class has the same weight to show how effectively the model works. The 0.95 weighted average F1-score considers class

support. This helps quantify success in datasets with varied class sizes. This classification report reveals that the model correctly classified events as yawning, microsleep, and alert.

Table 14

Classification Report

	f1-score	**support**
Yawning	0.96	32800
Micro Sleep	0.95	45600
Alert	0.94	60300
Accuracy	0.97	533700
Macro avg	0.95	533700
Weighted avg	0.95	533700

Note. The Classification Report Provides Numerous Parameters To Evaluate Yawning, Microsleep, And Alertness.

Leveraging VGG19 For Inference. The Figure 81 illustrates the performance of the VGG19 fine-tuned model on images. When an image with a face is provided to the prediction pipeline as shown in Figure B33, it undergoes basic data processing and then classified into one of three categories: yawning, alert, or microsleep.

Figure 81

Image Showing Microsleep, Yawning, Alert

Note. The Above Figure Gives A Clear Description Of Classified Images Microsleep, Alert, Yawning As Shown In Figure B35 And Figure B36.

Findings and Conclusion

Evaluation of different models, such as CNN, Viola Jones, VGG19, and YOLOv3, using a dataset with annotated frames demonstrated a range of performance indicators. VGG19 outperformed all other models in terms of accuracy and F1 score, with an impressive accuracy of 96.68% and an F1 score of 95.62%. A thorough examination revealed a clear association between the number of epochs and the improvements in accuracy and F1 scores across all models. The VGG19 model demonstrates greater performance in handling the subtleties of the dataset, despite its higher complexity compared to other models. This study recommends using the VGG19 model as it demonstrates exceptional accuracy and F1 score, ensuring reliable identification and classification of frame-level data.

Limitations

The aforementioned constraints, which include high computational complexity, resource-intensive training and deployment requirements, potential problems with real-time applications due to computational demands, and a tendency to overfit with smaller datasets, are undoubtedly applicable as general constraints to a variety of models, including CNN, VGG19, YOLOv3, and Viola Jones. Every model, in its unique manner, may encounter difficulties in situations with restricted computational resources, impeding their effectiveness in real-time applications. Their complex structure and dependence on significant calculations may hinder their ability to perform time-sensitive tasks, resulting in operational limitations. Furthermore, it is possible that all models could face challenges in making accurate predictions when working with smaller or diverse datasets. This could have an impact on their ability to perform well and maintain accuracy in different environmental settings or situations. Therefore, there is an ongoing requirement for innovative designs or alternative frameworks that achieve a more optimal trade-off between computational effectiveness and precision, particularly for real-world applications in limited-resource environments.

Future Scope

The field of drowsiness detection research has enormous potential and room for growth.The technology might be used with already available technologies like autonomous emergency braking and lane deviation warning to increase driver safety. Reducing latency and ensuring real-time performance through the use of edge computing on local devices in the car is essential for prompt feedback and action when driver weariness is detected.To improve the accuracy of the model's drowsiness identification, characteristics like heart rate tracking or head position might be added. Additionally, data from other sources, such as hand gestures, driving

patterns, and heart rate signals, could be combined.The driver's visage can be captured by a number of cameras and sensors to help the model detect tiredness.To track tiredness and improve the alert system as needed, this can be expanded into a mobile application.For the model to be widely used in cars, from luxury to economy models, it must be optimized.It will make it easier for the model to operate on low-power hardware without sacrificing precision.

References

Afsar, P., Shanid, M., Verghese, A., Sithara, A., Pakarath, A., Bijoy, & Ashmin, K. T. (2023). Real-Time Student Emotion and Drowsiness Detection Using YOLOv5 and CNN for Enhanced Learning. *2023 International Conference on Innovations in Engineering and Technology (ICIET)*. https://doi.org/10.1109/iciet57285.2023.10220929

Ahuja, H., Saurav, S., Srivastava, S., & Shekhar, C. (2020). Driver Drowsiness Detection using Knowledge Distillation Technique for Real Time Scenarios. *2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, 2020, Pp. 1-5*. https://doi.org/10.1109/indicon49873.2020.9342263

Albelwi, S., & Mahmood, A. (2017). A framework for designing the architectures of deep convolutional neural networks. *Entropy*, *19*(6), 242. https://doi.org/10.3390/e19060242

Ali, N., Hasan, I., Özyer, T., & Alhajj, R. (2021). Driver drowsiness detection by employing CNN and Dlib. *2021 22nd International Arab Conference on Information Technology (ACIT)*. https://doi.org/10.1109/acit53391.2021.9677197

Azim, T., Jaffar, M. A., & Mirza, A. M. (2009). Automatic Fatigue Detection of Drivers through Pupil Detection and Yawning Analysis. 2009 Fourth International Conference on Innovative Computing, Information and Control. https://doi.org/10.1109/icicic.2009.119

Bajaj, P., Ray, R., Shedge, S., Jaikar, S., & More, P. (2021). Synchronous System for Driver Drowsiness Detection Using Convolutional Neural Network, Computer Vision and Android Technology. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. https://doi.org/10.1109/icaccs51430.2021.9441670

Balam, V. P., Sameer, V. U., & Chinara, S. (2021). Automated classification system for drowsiness detection using convolutional neural network and electroencephalogram. *Iet Intelligent Transport Systems*, *15*(4), 514–524. https://doi.org/10.1049/itr2.12041

Basit, M. S., Ahmad, U., Ahmad, J., Ijaz, K., & Ali, S. A. (2022). Driver Drowsiness Detection with Region-of-Interest Selection Based Spatio-Temporal Deep Convolutional-LSTM. 2022 16th International Conference on Open Source Systems and Technologies (ICOSST). https://doi.org/10.1109/icosst57195.2022.10016825

Biju, A., & Edison, A. (2020). Drowsy Driver Detection Using Two Stage Convolutional Neural Networks. IEEE. https://doi.org/10.1109/raics51191.2020.9332476

Chandra, B. V. B., Naveen, C., Kumar, M. M. S., Bhargav, M. S. S., Poorna, S. S., & Anuraj, K. (2021). A Comparative Study of Drowsiness Detection From Eeg Signals Using Pretrained CNN Models. *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT),*. https://doi.org/10.1109/icccnt51525.2021.9579555

Dua, M., Shakshi, Singla, R., Raj, S. a. A., & Jangra, A. (2020). Deep CNN models-based ensemble approach to driver drowsiness detection. *Neural Computing and Applications*, *33*(8), 3155–3168. https://doi.org/10.1007/s00521-020-05209-7

Gong, D. H., & Kwak, K. C. (2017, August). Face detection and status analysis algorithms in day and night environments. In 2017 International Conference on Advanced Informatics, Concepts, Theory, and Applications (ICAICTA) (pp. 1-4). IEEE.https://doi.org/10.1109/icaicta.2017.8090965

Hemalatha, G. V., & Karthick, V. (2022b). An Effective Framework to Predict Visual Behaviour of Driver Drowsiness Level using Improved Ensemble Method over Dimensionality

Reduction Algorithm. 2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS). https://doi.org/10.1109/icscds53736.2022.9760814

Kasodu, B. N., Zakiyyah, A. Y., Rasjid, Z. E., & Parmonangan, I. H. (2022). CNN-based Drowsiness Detection with Alarm System to Prevent Microsleep. *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*. https://doi.org/10.1109/icitisee57756.2022.10057692

Képešiová, Z., Cigánek, J., & Kozák, Š. (2020b). Driver Drowsiness Detection Using Convolutional Neural Networks. *IEEE Sensors Journal*. https://doi.org/10.1109/ki48306.2020.9039851

Kielty, P., Dilmaghani, M. S., Shariff, W., Ryan, C., Lemley, J., & Corcoran, P. (2023). Neuromorphic Driver Monitoring Systems: A Proof-of-Concept for Yawn Detection and Seatbelt State Detection using an Event Camera. *IEEE Access*, *11*, 96363–96373. https://doi.org/10.1109/access.2023.3312190

Lazuardi, M. R., Hadi, M. Z. S., & Sudibyo, R. W. (2023). Driver Drowsiness Detection System Using Deep Learning Method to Reduce Risk Accident. *2023 International Electronics Symposium (IES)*. https://doi.org/10.1109/ies59143.2023.10242431

Luo, G. (2023). YOLOv5-based Fatigued Driving Detection. *2023 IEEE International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*. https://doi.org/10.1109/icsece58870.2023.10263342

Luh, G., Wu, H., Yong, Y., Lai, Y., & Chen, Y. (2019). Facial Expression Based Emotion Recognition Employing YOLOv3 Deep Neural Networks. *2019 International Conference on Machine Learning and Cybernetics (ICMLC),*. https://doi.org/10.1109/icmlc48188.2019.8949236

Macalisang, J., Alon, A. S., Jardiniano, M. F., Evangelista, D. C. P., Castro, J. C., & Tria, M. L. (2021). Drive-Awake: A YOLOv3 Machine Vision Inference Approach of Eyes Closure for Drowsy Driving Detection. *2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*. https://doi.org/10.1109/iicaiet51634.2021.9573811

Neshov, N., & Manolova, A. (2017). Drowsiness monitoring in real-time based on supervised descent method. *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. https://doi.org/10.1109/idaacs.2017.8095173

Oualla, M., Sadiq, A., & Mbarki, S. (2014). A survey of Haar-Like feature representation. IEEE. https://doi.org/10.1109/icmcs.2014.6911186

Perkins, E., Sitaula, C., Burke, M. G., Manousakis, J. E., Anderson, C., & Marzbanrad, F. (2023b). Investigating physiological and behavioural sensing modalities towards drowsiness detection. *IEEE Sensors Journal*, *23*(23), 29513–29524. https://doi.org/10.1109/jsen.2023.3326434

Redmon, J., & Farhadi, A. (2018). YOLOV3: an incremental improvement. arXiv (Cornell University). https://doi.org/10.48550/arXiv.1804.02767

Rivelli, E. (2022, December 16). *Drowsy driving statistics and facts 2022*. Bankrate. https://www.bankrate.com/insurance/car/drowsy-driving-statistics/

Soe, M. T., Min, A. Z., Kyaw, H. T., Paing, M. M., Htet, S. M., & Aye, B. (2022). Abnormal Behavior Detection in Real-time for Advanced Driver Assistance System (ADAS) using YOLO. *2022 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*. https://doi.org/10.1109/isiea54517.2022.9873672

Soman, S., Kumar, G. S., & Abubeker, K. M. (2023). Internet-of-Things-Assisted Artificial Intelligence-Enabled Drowsiness Detection Framework. *IEEE Sensors Letters*, *7*(7), 1–4. https://doi.org/10.1109/lsens.2023.3289143

S, S., I., Ramli, R., Azri, M. A., Aliff, M., & Mohammad, Z. (2022). Raspberry Pi Based Driver Drowsiness Detection System Using Convolutional Neural Network (CNN). *2022 IEEE 18th International Colloquium on Signal Processing & Applications (CSPA)*. https://doi.org/10.1109/cspa55076.2022.9781879

Sur, D., Rudra, S., Mitra, S., Chakraborty, P., Dolai, C., & Mitra, S. (12 2015). *Development of a Drowsiness Detection of a Car Driver by Eye Detection Methodology of Image Processing*.(2015). *4th International Conference on "Computing Communication and Sensor Network", Sponsored by IEEE(EDS) Kolkata*. https://www.researchgate.net/publication/320226262_Development_of_a_Drowsiness_Detection_of_a_Car_Driver_by_Eye_Detection_Methodology_of_Image_Processing

Suresh, A., Naik, A. S. K., Pramod, A., Kumar, N., & Mayadevi, N. (2023b). Analysis and Implementation of Deep Convolutional Neural Network Models for Intelligent Driver Drowsiness Detection System. *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*. https://doi.org/10.1109/iciccs56967.2023.10142299

Tanouz, D., Subramanian, R. R., Eswar, D., Reddy, G. V., Kumar, A., & Praneeth, C. V. N. M. (2021). Credit Card Fraud Detection Using Machine Learning. *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. https://doi.org/10.1109/iciccs51141.2021.9432308

Tyagi, N., Goyal, R., & Rautela, R. (2022). Real Time Drowsiness Detection System. *2022 International Conference on Cyber Resilience (ICCR)*. https://doi.org/10.1109/iccr56254.2022.9996053

Appendix A

Complete Gantt Chart

Figure A1. Gantt Chart For Business Understanding To Data Transformation

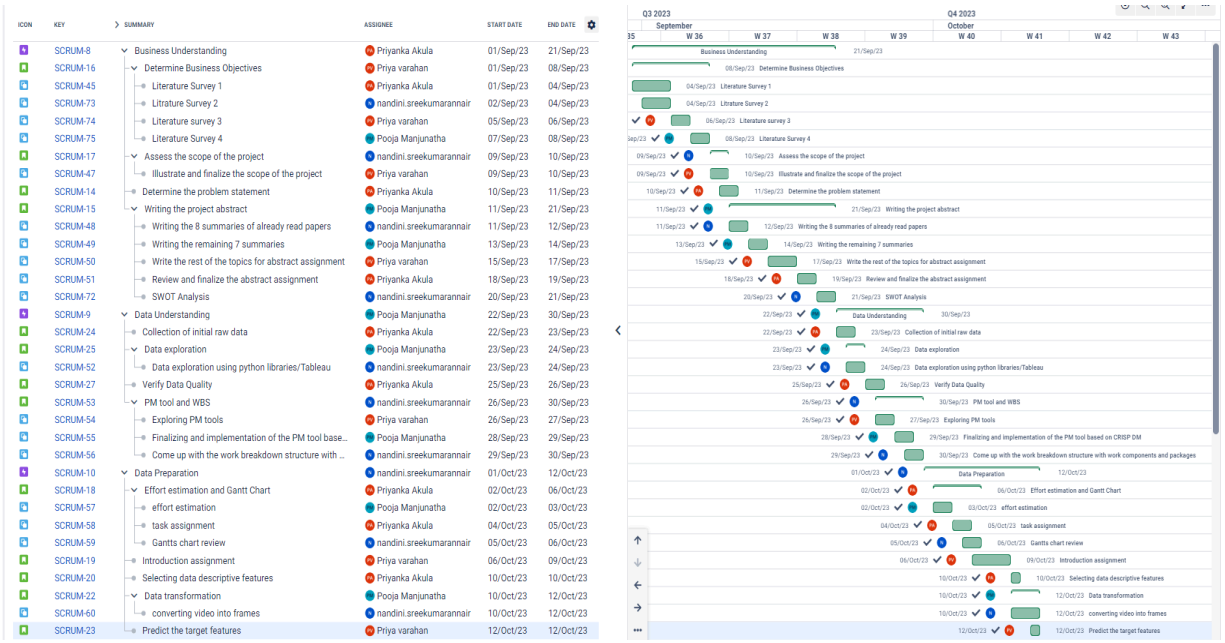
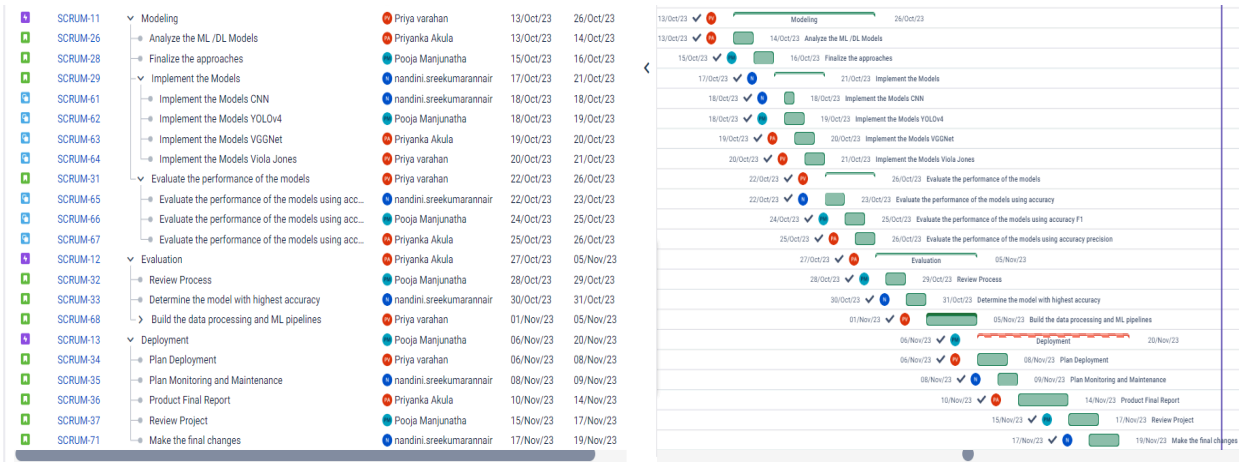


Figure A2. Gantt Chart For Modeling To Deployment



Gantt Chart includes all epics, tasks, subtasks, timelines, assignees and dependencies of all phases that are developed using Jira Software.

Appendix B

Data Engineering

Figure B1. Number Of Videos In The Microsleep Directory

```
video_extensions = ['.mp4', '.avi', '.mov']  # Add more extensions if needed

video_files = [f for f in os.listdir(directory_path) if f.endswith(tuple(video_extensions))]

# Print the count of video files
print(f"Number of video files in the microsleep directory: {len(video_files)}")
```

Figure B2. Properties Of Microsleep Directory

```
directory_path ="/Users/poojamanjunatha/Desktop/GWAR/NYTMED/MICROSLEEP"
video_files = [f for f in os.listdir(directory_path) if f.endswith(('.m

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Extract video properties
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_rate = int(cap.get(cv2.CAP_PROP_FPS))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    duration = frame_count / frame_rate

    # Print the details for each video
    print(f"Video: {video_file}")
    print(f"Frame Count: {frame_count}")
    print(f"Frame Rate: {frame_rate}")
    print(f"Resolution: {width}x{height}")
    print(f"Duration: {duration} seconds")

    # Perform video analysis or other tasks as needed

    # Close the video file
    cap.release()

    # Store or export results for this video
```

Figure B3. Highest And Lowest Frame Count For The Microsleep Directory

```
highest_frame_count = -1
lowest_frame_count = float('inf')
video_with_highest_frame_count = None
video_with_lowest_frame_count = None

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Extract the frame count for the current video
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Check if the current video has the highest frame count
    if frame_count > highest_frame_count:
        highest_frame_count = frame_count
        video_with_highest_frame_count = video_file

    # Check if the current video has the lowest frame count
    if frame_count < lowest_frame_count:
        lowest_frame_count = frame_count
        video_with_lowest_frame_count = video_file

    # Close the video file
    cap.release()

# Print the video with the highest and lowest frame counts
print(f"Video with the highest frame count: {video_with_highest_frame_c
print(f"Highest frame count: {highest_frame_count}")
print(f"Video with the lowest frame count: {video_with_lowest_frame_cou
print(f"Lowest frame count: {lowest_frame_count}")
```

Figure B4. Resolution Of Video Files For The Microsleep Directory

```
# Initialize a variable to track whether all videos have the same resol
all_videos_have_same_resolution = True

# Initialize variables to store the expected resolution
expected_width, expected_height = 1920, 1080

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Get the resolution of the current video
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Check if the resolution matches the expected resolution
    if (width, height) != (expected_width, expected_height):
        all_videos_have_same_resolution = False
        print(f"Video {video_file} has a resolution of {width}x{height}

    # Close the video file
    cap.release()

# Check if all videos have the same resolution
if all_videos_have_same_resolution:
    print("All videos have the expected resolution of 1920x1080.")
else:
    print("Not all videos have the expected resolution of 1920x1080.")
```

```
All videos have the expected resolution of 1920x1080.
```

Figure B5. Length Of Video Files In Microsleep Directory

```
# Initialize variables to keep track of the highest, shortest, and total duration
highest_duration = 0
shortest_duration = float('inf')
total_duration = 0
valid_video_count = 0  # To track the number of valid videos

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Check if the video can be opened successfully
    if not cap.isOpened():
        print(f"Unable to open {video_file}. Skipping.")
        continue

    # Get the frame rate of the video, skip if it's zero
    frame_rate = int(cap.get(cv2.CAP_PROP_FPS))
    if frame_rate == 0:
        print(f"Unable to determine the frame rate for {video_file}. Skipping.")
        cap.release()
        continue

    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = frame_count / frame_rate

    # Update statistics
    if duration > highest_duration:
        highest_duration = duration
    if duration < shortest_duration:
        shortest_duration = duration
    total_duration += duration
    valid_video_count += 1

    # Print the name of the video and its duration
    print(f"Video: {video_file}")
    print(f"Duration: {duration:.2f} seconds\n")

    # Close the video file
    cap.release()

# Check if any valid videos were processed
if valid_video_count == 0:
    print("No valid videos found in the directory.")
else:
    # Calculate the average duration for valid videos
    average_duration = total_duration / valid_video_count

    # Print the overall statistics
    print(f"Highest duration: {highest_duration:.2f} seconds")
    print(f"Shortest duration: {shortest_duration:.2f} seconds")
    print(f"Average duration: {average_duration:.2f} seconds")
```

Figure B6. Properties Of Video Files In The Yawning Directory

```
import os
import cv2  # For OpenCV

directory_path ="/Users/poojamanjunatha/Desktop/GWAR/NYTMED/yawning"
video_files = [f for f in os.listdir(directory_path) if f.endswith(('.m

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Extract video properties
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_rate = int(cap.get(cv2.CAP_PROP_FPS))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    duration = frame_count / frame_rate

    # Print the details for each video
    print(f"Video: {video_file}")
    print(f"Frame Count: {frame_count}")
    print(f"Frame Rate: {frame_rate}")
    print(f"Resolution: {width}x{height}")
    print(f"Duration: {duration} seconds")

    # Perform video analysis or other tasks as needed

    # Close the video file
    cap.release()
```

Figure B7. Number Of Video Files In The Yawning Directory

```
video_extensions = ['.mp4', '.avi', '.mov']  # Add more extensions if n

video_files = [f for f in os.listdir(directory_path) if f.endswith(tupl

# Print the count of video files
print(f"Number of video files in the yawning directory: {len(video_file
```

Figure B8. Frame Count In Yawning Directory

```
highest_frame_count = -1
lowest_frame_count = float('inf')
video_with_highest_frame_count = None
video_with_lowest_frame_count = None

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Extract the frame count for the current video
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Check if the current video has the highest frame count
    if frame_count > highest_frame_count:
        highest_frame_count = frame_count
        video_with_highest_frame_count = video_file

    # Check if the current video has the lowest frame count
    if frame_count < lowest_frame_count:
        lowest_frame_count = frame_count
        video_with_lowest_frame_count = video_file

    # Close the video file
    cap.release()

# Print the video with the highest and lowest frame counts
print(f"Video with the highest frame count: {video_with_highest_frame_count}")
print(f"Highest frame count: {highest_frame_count}")
print(f"Video with the lowest frame count: {video_with_lowest_frame_count}")
print(f"Lowest frame count: {lowest_frame_count}")
```

Figure B9. Resolution Of Videos In The Yawning Directory

```
# Initialize a variable to track whether all videos have the s
all_videos_have_same_resolution = True

# Initialize variables to store the expected resolution
expected_width, expected_height = 1920, 1080

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Get the resolution of the current video
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Check if the resolution matches the expected resolution
    if (width, height) != (expected_width, expected_height):
        all_videos_have_same_resolution = False
        print(f"Video {video_file} has a resolution of {width}

    # Close the video file
    cap.release()

# Check if all videos have the same resolution
if all_videos_have_same_resolution:
    print("All videos have the expected resolution of 1920x108
else:
```

Figure B10. Duration Of Videos In The Yawning Directory

```
# Initialize variables to keep track of the highest, shortest, and total duration
highest_duration = 0
shortest_duration = float('inf')
total_duration = 0
valid_video_count = 0  # To track the number of valid videos

for video_file in video_files:
    video_path = os.path.join(directory_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Check if the video can be opened successfully
    if not cap.isOpened():
        print(f"Unable to open {video_file}. Skipping.")
        continue

    # Get the frame rate of the video, skip if it's zero
    frame_rate = int(cap.get(cv2.CAP_PROP_FPS))
    if frame_rate == 0:
        print(f"Unable to determine the frame rate for {video_file}. Skipping.")
        cap.release()
        continue

    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = frame_count / frame_rate

    # Update statistics
    if duration > highest_duration:
        highest_duration = duration
    if duration < shortest_duration:
        shortest_duration = duration
    total_duration += duration
    valid_video_count += 1

    # Print the name of the video and its duration
    print(f"Video: {video_file}")
    print(f"Duration: {duration:.2f} seconds\n")

    # Close the video file
    cap.release()

# Check if any valid videos were processed
if valid_video_count == 0:
    print("No valid videos found in the directory.")
else:
    # Calculate the average duration for valid videos
    average_duration = total_duration / valid_video_count

    # Print the overall statistics
    print(f"Highest duration: {highest_duration:.2f} seconds")
    print(f"Shortest duration: {shortest_duration:.2f} seconds")
    print(f"Average duration: {average_duration:.2f} seconds")
```

Figure B11. Image Sharpness Distribution

```
import cv2
import numpy as np

def calculate_image_sharpness(image):
    # Calculate image sharpness using Laplacian operator
    laplacian = cv2.Laplacian(image, cv2.CV_64F)
    sharpness = np.var(laplacian)
    return sharpness

# Analyze image sharpness for all images
image_quality_scores = [calculate_image_sharpness(cv2.imread(image_fi

# Visualize image quality scores
plt.figure(figsize=(10, 5))
plt.hist(image_quality_scores, bins=20, edgecolor='k')
plt.title("Image Sharpness Distribution")
plt.xlabel("Sharpness Score")
plt.ylabel("Frequency")
plt.show()
```

Figure B12. Heatmap Between Image Dimensions

```
import pandas as pd
def get_image_files(directory):
    image_files = [os.path.join(directory, f) for f in os.listdir(dir
    return image_files

# Function to extract image dimensions
def get_image_dimensions(image_file):
    image = cv2.imread(image_file)
    height, width, _ = image.shape
    return (height, width)

# Function to extract class labels from subdirectories
def label_from_directory(directory):
    class_label = os.path.basename(directory)
    return class_label


# Check if the root directory exists
if os.path.exists(root_directory) and os.path.isdir(root_directory):
    # Recursively List All Image Files in Subfolders
    all_image_files = []
    for root, dirs, files in os.walk(root_directory):
        for directory in dirs:
            image_dir = os.path.join(root, directory)
            image_files = get_image_files(image_dir)
            all_image_files.extend(image_files)

    # Extract class labels and image dimensions
    class_labels = [label_from_directory(os.path.dirname(image_file))
    image_dimensions = [get_image_dimensions(image_file) for image_fi

    # Create a DataFrame for correlation analysis
    data = {'Class Label': class_labels, 'Image Height': [dim[0] for
    df = pd.DataFrame(data)

    # Calculate Pearson correlations
    correlations = df.corr()

    # Visualize correlation matrix as a heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt=".2f")
    plt.title("Correlation Analysis: Image Dimensions and Class Label
    plt.show()
else:
    print(f"The directory '{root_directory}' does not exist.")
```

Figure B13. Importing Necessary Libraries

```
In [3]: py_file_location= r"C:\GWAR\Project\dataset\annotations"
        sys.path.append(os.path.abspath(py_file_location))
```

```
In [2]: import sys
        import os
```

```
In [4]: import os
        import json
        from itertools import islice
        import random

        from tqdm import tqdm
        import numpy as np
        import matplotlib.pyplot as plt
        import cv2
        from show_annotations import draw_bboxes, draw_landmarks
```

Figure B14. Converting The Video Dataset To A Frame Level Annotated Dataset

```
# Path to your video dataset
video_dataset_path = r'C:\Users\pinky\OneDrive\Documents\Gwar\Dataset\NYTMED\MICROSLEEP/'
output_dataset_path = r'C:\GWAR\Project\dataset\overall dataset/'

# List of video files
video_files = [f for f in os.listdir(video_dataset_path) if f.endswith('.mp4')]
count = 0
# Loop through each video file
for video_file in tqdm(video_files):
    video_path = os.path.join(video_dataset_path, video_file)

    # Create a directory with the video file's name
   # output_folder = os.path.join(output_dataset_path, os.path.splitext(video_file)[0])
    output_folder = r'C:\GWAR\Project\dataset\overall dataset/'
    os.makedirs(output_folder, exist_ok=True)

    # Read the video
    vidcap = cv2.VideoCapture(video_path)
    fps = vidcap.get(cv2.CAP_PROP_FPS)

    success, image = vidcap.read()
    frame_number = 0
    # Iterate through the frames and save them as images
    while success:
        frame_name = os.path.join(output_folder, f"{os.path.splitext(video_file)[0]}_microsleep_frame{count}.jpg")
        cv2.imwrite(frame_name, image)  # Save frame as JPEG file
        frame_number += fps//5
        # Set the frame position
        vidcap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
        success, image = vidcap.read()
        print(f"Frame {count} extracted from {video_file}")
        count += 1

# Label the images here according to your dataset's labeling conventions
# You can create a separate file with labels or use a pre-existing mapping for your dataset.
```

```
  0%|                                                                                | 0/23 [00:00<?, ?it/s]
Frame 0 extracted from P1042751_na.mp4
Frame 1 extracted from P1042751_na.mp4
Frame 2 extracted from P1042751_na.mp4
```

Figure B15. Finding Classes In The Data Set And Finding Their Ratios

```
In [10]:
def driver_state_distribution(annotations_file):
    """Compute distribution of driver states"""
    annotations = json.load(open(annotations_file))
    driver_states = {"alert": 0, "microsleep": 0, "yawning": 0}
    for frame, label in annotations.items():
        driver_states[label["driver_state"]] += 1
    driver_states_count = driver_states.copy()

    val_sum = sum(driver_states.values())
    for key, val in driver_states.items():
        driver_states[key] = val / val_sum
    print(driver_states)
    return driver_states_count, driver_states

driver_states_count, driver_states = driver_state_distribution("C:/GWAR/da
plt.bar(list(driver_states.keys()), list(driver_states.values()))
plt.xticks(list(driver_states.keys()))
plt.xlabel('Category')
plt.title("(Train set) Driver state histogram")
plt.xlabel("Driver state")
plt.ylabel("Ratio");
plt.tight_layout()
plt.savefig("C:/GWAR/dataset/image/train_set_distribution.pdf")
print(driver_states_count)
class_weights = list(1 / count for count in driver_states_count.values())
print(f"class_weights: {class_weights}")
```

```
{'alert': 0.7304194558511935, 'microsleep': 0.16650728469370535, 'yawnin
g': 0.10307325945510117}
{'alert': 38954, 'microsleep': 8880, 'yawning': 5497}
class_weights: [2.567130461570057e-05, 0.00011261261261261261, 0.00018191
740949608878]
```

Figure B16. Result Of Dividing The Dataset Into Classes And Their Ratios

```
In [11]:
driver_states_count, driver_states = driver_state_distribution("C:/GWAR/da
plt.bar(list(driver_states.keys()), list(driver_states.values()))
plt.xticks(list(driver_states.keys()))
plt.xlabel('Category')
plt.title("(Validation set) Driver state histogram")
plt.xlabel("Driver state")
plt.ylabel("Ratio");
plt.tight_layout()
plt.savefig("C:/GWAR/dataset/image/val_set_distribution.pdf")
print(driver_states_count)
class_weights = list(1 / count for count in driver_states_count.values())
print(f"class_weights: {class_weights}")
```

```
{'alert': 0.6962430055955235, 'microsleep': 0.1606714628297362, 'yawnin
g': 0.1430855157474022}
{'alert': 1742, 'microsleep': 402, 'yawning': 358}
class_weights: [0.000574052812858783, 0.002487562189054726 3, 0.0027932960
89385475]
```

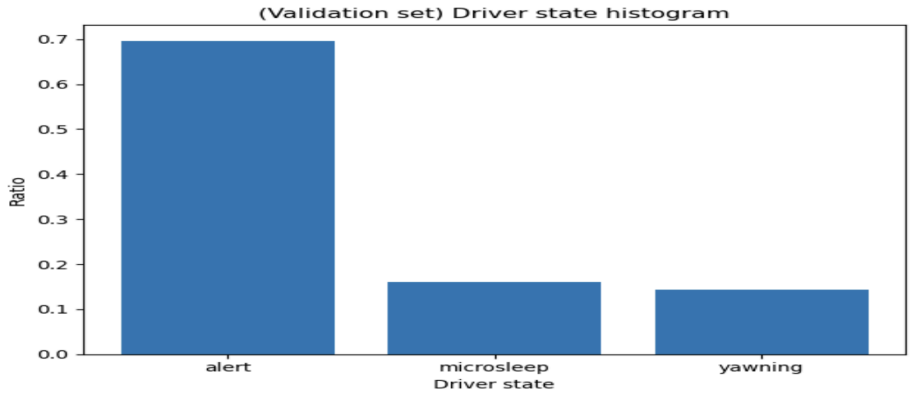
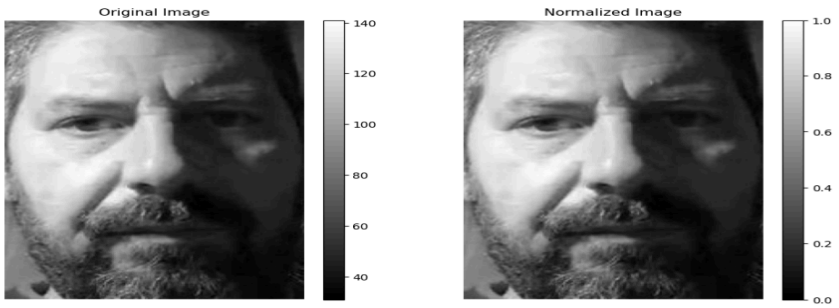


Figure B17. Data Normalization

```
In [65]:
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function for image normalization
def normalize_image(frame_data):
    # Convert the image to float32 format
    frame_data = frame_data.astype(np.float32)

    # Normalize the image to the range [0, 1]
    normalized_frame_data = (frame_data - np.min(frame_data)) / (np.max(fr

    return normalized_frame_data

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
frame_data = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if frame_data is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Normalize the image
normalized_frame_data = normalize_image(frame_data)

# Display the original and normalized images using matplotlib
plt.figure(figsize=(12, 6))

# Plot the original image
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(frame_data, cmap='gray')
plt.colorbar()
plt.axis('off')

# Plot the normalized image
plt.subplot(1, 2, 2)
plt.title('Normalized Image')
plt.imshow(normalized_frame_data, cmap='gray')
plt.colorbar()
plt.axis('off')

plt.show()

# Print pixel values
print("\nOriginal Image Pixel Values:")
print(frame_data)
print("\nNormalized Image Pixel Values:")
print(normalized_frame_data)
```

Figure B18. Result Obtained After Data Normalization




```
Original Image Pixel Values:
[[ 97 100 101 ...  50  51  51]
 [ 85  96  96 ...  51  51  51]
 [ 76  86  80 ...  51  51  51]
 ...
 [ 65  65  64 ...  37  37  37]
 [ 64  64  63 ...  37  37  37]
 [ 65  65  64 ...  37  37  37]]

Normalized Image Pixel Values:
[[0.6        0.6272727  0.6363636  ... 0.17272727 0.18181819 0.18181819]
 [0.4909091  0.59090906 0.59090906 ... 0.18181819 0.18181819 0.18181819]
 [0.4090909  0.5        0.44545454 ... 0.18181819 0.18181819 0.18181819]
 ...
 [0.3090909  0.3090909  0.3        ... 0.05454545 0.05454545 0.05454545]
 [0.3        0.3        0.29090908 ... 0.05454545 0.05454545 0.05454545]
 [0.3090909  0.3090909  0.3        ... 0.05454545 0.05454545 0.05454545]]
```

Figure B19. Before And After Images Of Vertically Flipped Data

```
In [136]:
import cv2
import matplotlib.pyplot as plt

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
frame_data = cv2.imread(image_path)

# Check if the image was loaded successfully
if frame_data is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Flip the image vertically
flipped_vertical = cv2.flip(frame_data, 0)

# Display the original and vertically flipped images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Vertically Flipped Image')
plt.imshow(cv2.cvtColor(flipped_vertical, cv2.COLOR_BGR2RGB))
plt.show()
```

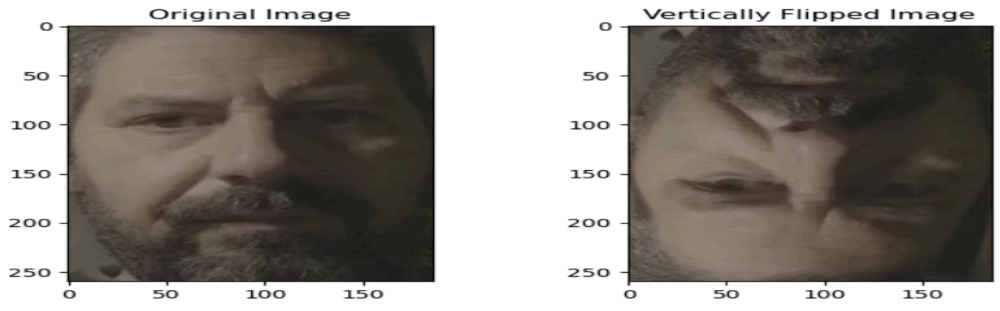


Figure B20. Before And After Images Of Cropped Image

In [73]:

```
import cv2
import matplotlib.pyplot as plt

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path)

# Crop the image
cropped_image = image[30:250, 30:250]

# Display the original and cropped images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Cropped Image')
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))
plt.show()
```

Figure B21. Before And After Images After Gaussian Noise Removal

In [151]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path)

# Generate Gaussian noise and add it to the image
mean = 0
var = 0.1
sigma = var ** 0.5
row, col, ch = image.shape
gaussian = np.random.normal(mean, sigma, (row, col, ch))
Gaussian_noisy_image = cv2.add(image, gaussian.astype(np.uint8))

# Display the original and noisy images using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Gaussian Noise Image')
plt.imshow(cv2.cvtColor(Gaussian_noisy_image, cv2.COLOR_BGR2RGB))
plt.show()
```

Figure B22. Before And After Result Of Gray Scaling On Images

In [152]:

```
import cv2
import matplotlib.pyplot as plt

# Read the original color image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
color_image = cv2.imread(image_path)

# Check if the image was loaded successfully
if color_image is None:
    raise FileNotFoundError("Could not read the image. Please check the im

# Convert the color image to grayscale
gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)

# Display the original color and grayscale images using matplotlib
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Color Image')
plt.imshow(cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Grayscale Image')
plt.imshow(gray_image, cmap='gray')
plt.show()
```

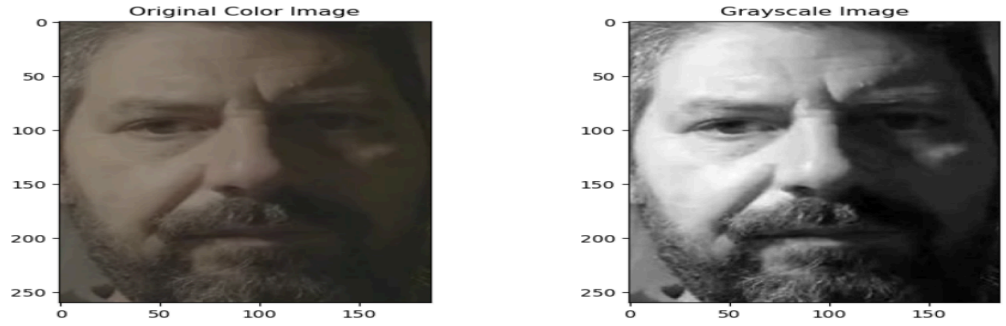


Figure B23. Before And After Result Reconstruction With Pca

In [91]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the image into a 1D array
image_flat = image.flatten()

# Determine a reasonable value for the number of components
n_components = 1  # Change this value as needed

# Apply PCA
pca = PCA(n_components=n_components, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Calculate pixel-wise differences
difference = np.abs(image - image_restored)

# Display the original and reconstructed images, along with the pixel-wise
plt.figure(figsize=(16, 6))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(image_restored, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.title('Pixel-wise Differences')
plt.imshow(difference, cmap='gray')
plt.axis('off')
plt.show()

# Print the average difference value
print(f"Average pixel-wise difference: {np.mean(difference)}")
```

Figure B24. Python Code For Pca

```
In [105]:
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Function to read images from a folder
def load_images_from_folder(folder_path):
    images = []
    for filename in os.listdir(folder_path):
        img = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_G
        if img is not None:
            img = cv2.resize(img, (100, 100))  # Resize the images to a co
            images.append(img)
    return images

# Specify the folder path containing images
folder_path = r'C:\GWAR\dataset\classification_frames\P1042756_720'  # Rep

# Load images from the folder
images = load_images_from_folder(folder_path)

# Flatten the images into a 2D array
images_flat = np.array([img.flatten() for img in images])

# Determine a reasonable value for the number of components
n_components = 30  # Change this value as needed

# Apply PCA
pca = PCA(n_components=n_components, svd_solver='full')
images_pca = pca.fit_transform(images_flat)
images_restored = pca.inverse_transform(images_pca).reshape(images_flat.sh

# Calculate the absolute pixel-wise differences between the original and P
pixel_differences = np.abs(images_flat.astype(int) - images_restored.astyp

# Create a scree plot showing the cumulative variance explained by each pr
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

# Plot the scree plot
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(images[0], cmap='gray')  # Display the first image
plt.axis('off')

plt.subplot(1, 2, 2)
plt.plot(np.arange(1, n_components + 1), cumulative_variance_ratio, marker
plt.title('Scree Plot of Cumulative Variance Explained')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained')
plt.grid()
```

Figure B25. Image Showing The Result Of Pca

```
plt.show()
```

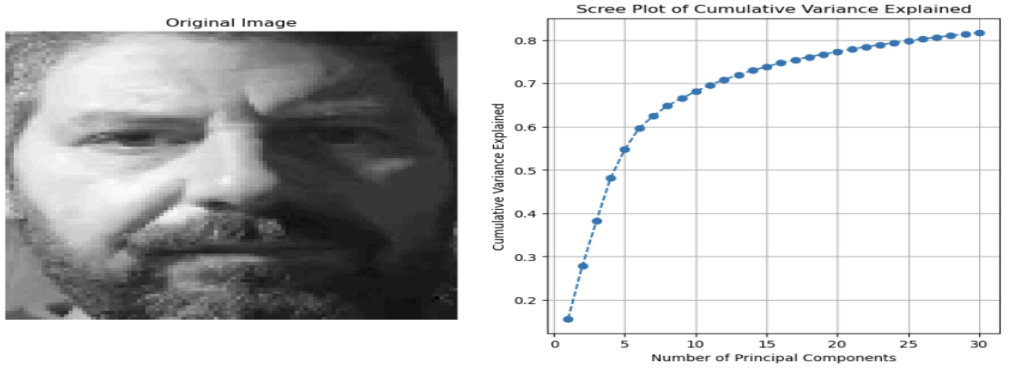


Figure B26. Code To Apply Pca On Dataset

In [109]:

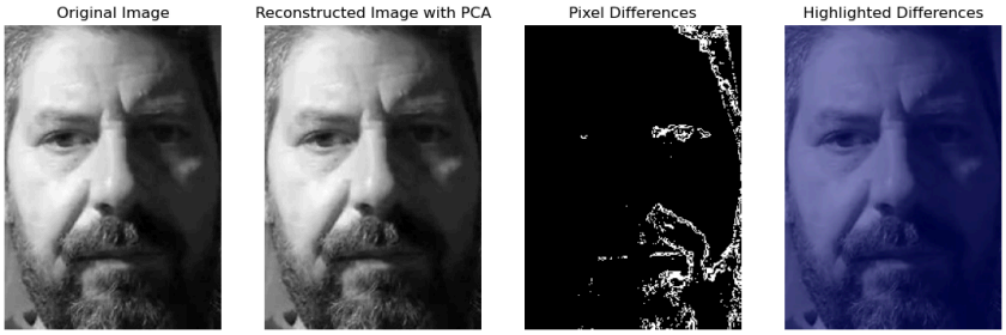
```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from tqdm import tqdm

# Function to read images from a folder
def load_images_from_folder(folder_path):
    images = []
    for filename in tqdm(os.listdir(folder_path), desc='Loading images'):
        img = cv2.imread(os.path.join(folder_path, filename), cv2.IMREAD_G
        if img is not None:
            img = cv2.resize(img, (100, 100))  # Resize the images to a co
            images.append(img)
    return images

# Specify the folder path containing images
folder_path = r'C:\GWAR\dataset\classification_frames\P1042756_720'  # Rep

# Load images from the folder
images = load_images_from_folder(folder_path)

# Flatten the images into a 2D array
images_flat = np.array([img.flatten() for img in images])

# Apply PCA with 98% variance retained
pca = PCA(0.98, svd_solver='full')
images_pca = pca.fit_transform(images_flat)
images_restored = pca.inverse_transform(images_pca).reshape(images_flat.sh

# Select an image index for display
image_index = 0

# Display the original and reconstructed images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(images[image_index], cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(images_restored[image_index].reshape(images[image_index].shape)
plt.axis('off')
plt.show()
```

```
Loading images: 100%|██████████| 2502/2502 [00:00<00:00, 4421.47it/s]
```

Figure B27. Code For Reconstructing Image With Pca, Pixel Differences And Highlight The Differences

```
In [117]:
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Read the original image
image_path = r'C:\GWAR\dataset\classification_frames\P1042756_720\frame4.j
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Flatten the original image into a 1D array
image_flat = image.flatten()

# Apply PCA with 98% variance retained
pca = PCA(0.98, svd_solver='full')
image_pca = pca.fit_transform(image_flat.reshape(-1, 1))
image_restored = pca.inverse_transform(image_pca).reshape(image.shape)

# Calculate the absolute pixel-wise differences between the original and r
pixel_differences = np.abs(image.astype(int) - image_restored.astype(int))

# Set a threshold for highlighting differences
threshold = 20  # Adjust as needed

# Create a mask for pixels where differences exceed the threshold
difference_mask = pixel_differences > threshold

# Display the differences between the original and reconstructed images
plt.figure(figsize=(14, 6))
plt.subplot(1, 4, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Reconstructed Image with PCA')
plt.imshow(image_restored, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 3)
plt.title('Pixel Differences')
plt.imshow(pixel_differences, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('Highlighted Differences')
plt.imshow(image, cmap='gray')
plt.imshow(difference_mask, cmap='jet', alpha=0.5)
plt.axis('off')

plt.show()
```

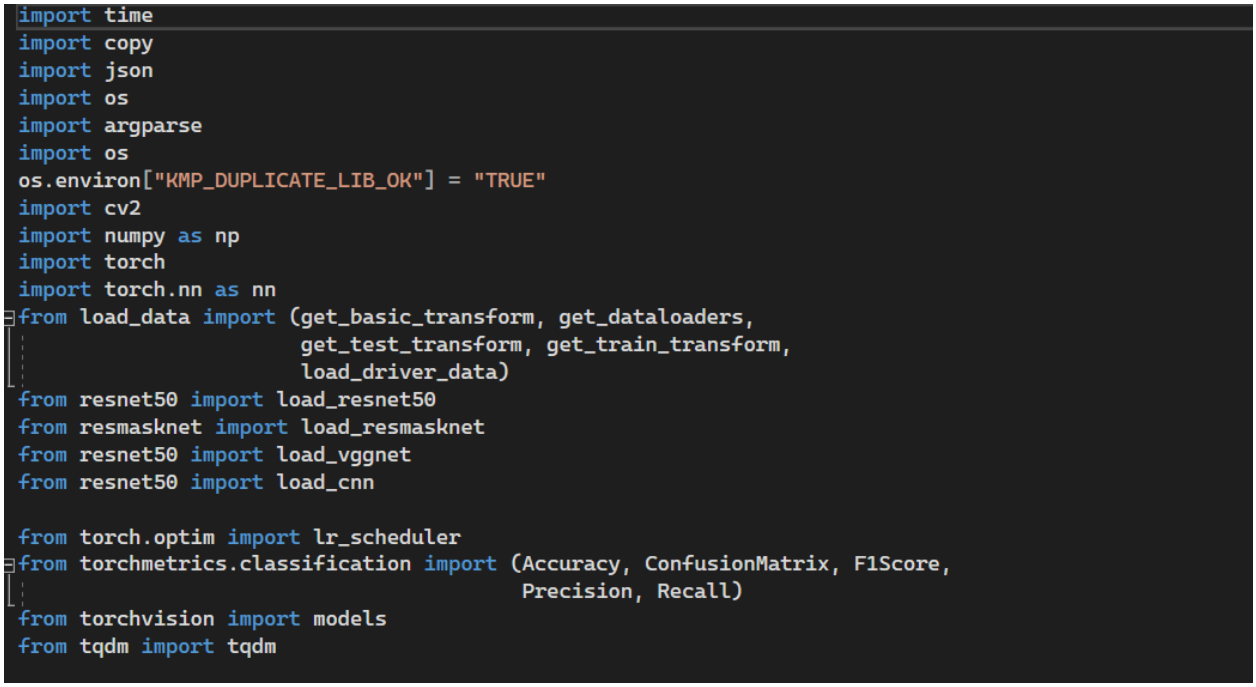
Figure B28. Result Of Reconstructing Image With Pca, Pixel Differences And Highlight The Differences



Train and Prediction Pipeline For Model Building

Training pipelines can be triggered using the following command “Python train.py --model [MODEL NAME]”, here the model name can be one of the following (VGGnet, CNN, YOLOv3, violajones). Using this one command any of these four models can be trained and evaluation scores can be generated.

Figure B29. This Image Shows All The Modules And Preprocessing Code Needed For Model Training



```
import time
import copy
import json
import os
import argparse
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
import cv2
import numpy as np
import torch
import torch.nn as nn
from load_data import (get_basic_transform, get_dataloaders,
                       get_test_transform, get_train_transform,
                       load_driver_data)
from resnet50 import load_resnet50
from resmasknet import load_resmasknet
from resnet50 import load_vggnet
from resnet50 import load_cnn

from torch.optim import lr_scheduler
from torchmetrics.classification import (Accuracy, ConfusionMatrix, F1Score,
                                         Precision, Recall)
from torchvision import models
from tqdm import tqdm
```

Figure B30. This Image Shows The Training Pipeline And How Different Models Are Loaded Based On The Model_name Parameter

```python
if __name__ == "__main__":
    # parse arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", type=str, required=True, default="resnet50")
    args = parser.parse_args()

    # load hyper-parameters and dataset
    params = json.load(open(os.path.join(os.getcwd(), "hyper_params.json")))
    dataloaders, ds_labels, labels_ds = load_driver_data(params)

    # load model
    if args.model == "resnet50":
        model = load_resnet50(n_classes=params["n_classes"])
    elif args.model == "resmasknet":
        model = load_resmasknet(n_classes=params["n_classes"])
    elif args.model == "vggnet":
        model = load_vggnet(n_classes=params["n_classes"])
    elif args.model == "cnn":
        model = load_cnn(n_classes=params["n_classes"])
    elif args.model == "violajones":
        model = load_vj(n_classes=params["n_classes"])

    else:
        raise Exception(f"Model: {args.model} not supported!")

    # train model
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=params["learning_rate"], weight_decay=params["weight_decay"])
    scheduler = lr_scheduler.StepLR(optimizer, step_size=params["step_size"], gamma=0.1)
    model, losses, f1s = train(
        dataloaders, model, criterion, optimizer,
        scheduler,2,
        params["n_classes"], model_path=f"./{args.model}/{args.model}_ds.pt"
    )
    # save training history
    save_history(losses, f1s, results_dir=f"./{args.model}/results")
```

Figure B31. Training Vgg19 Model Using The Training Pipeline.

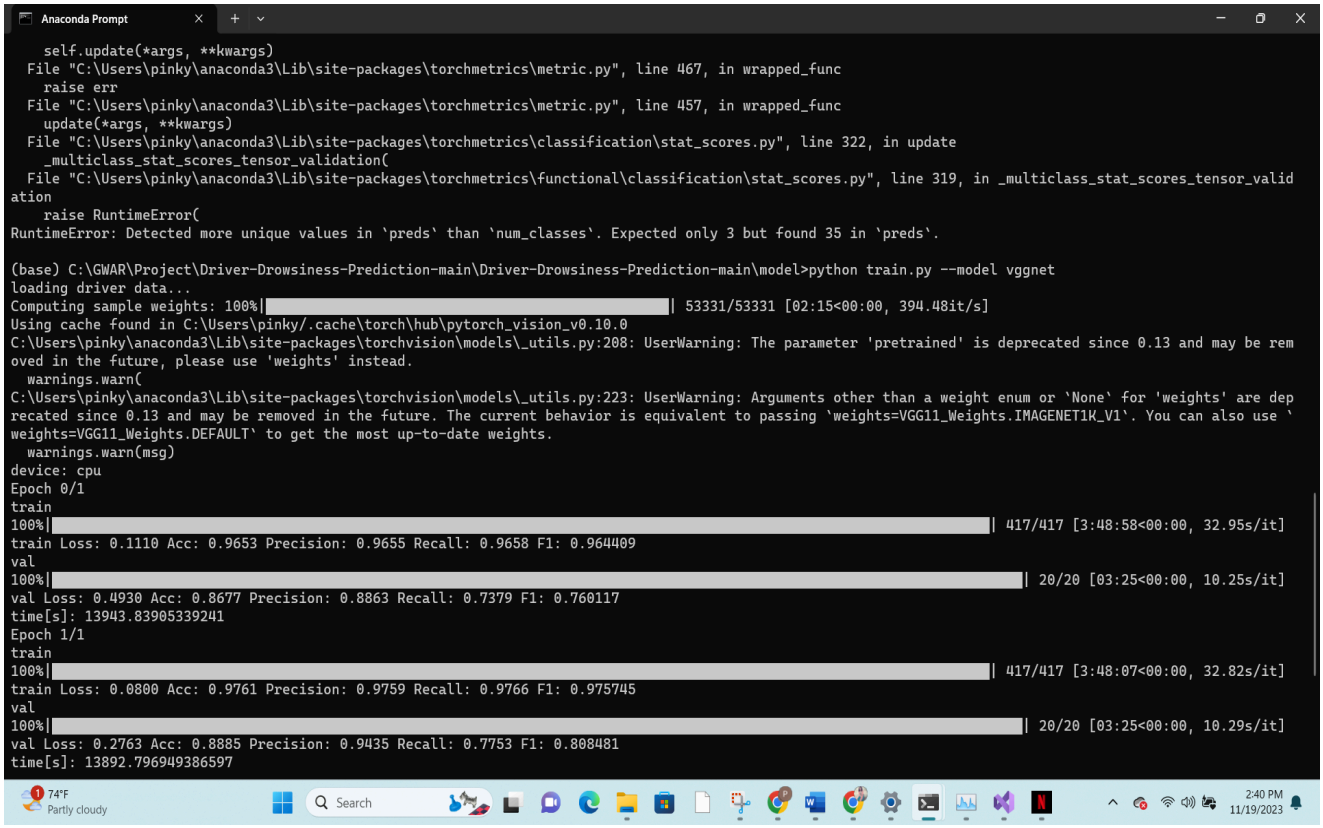
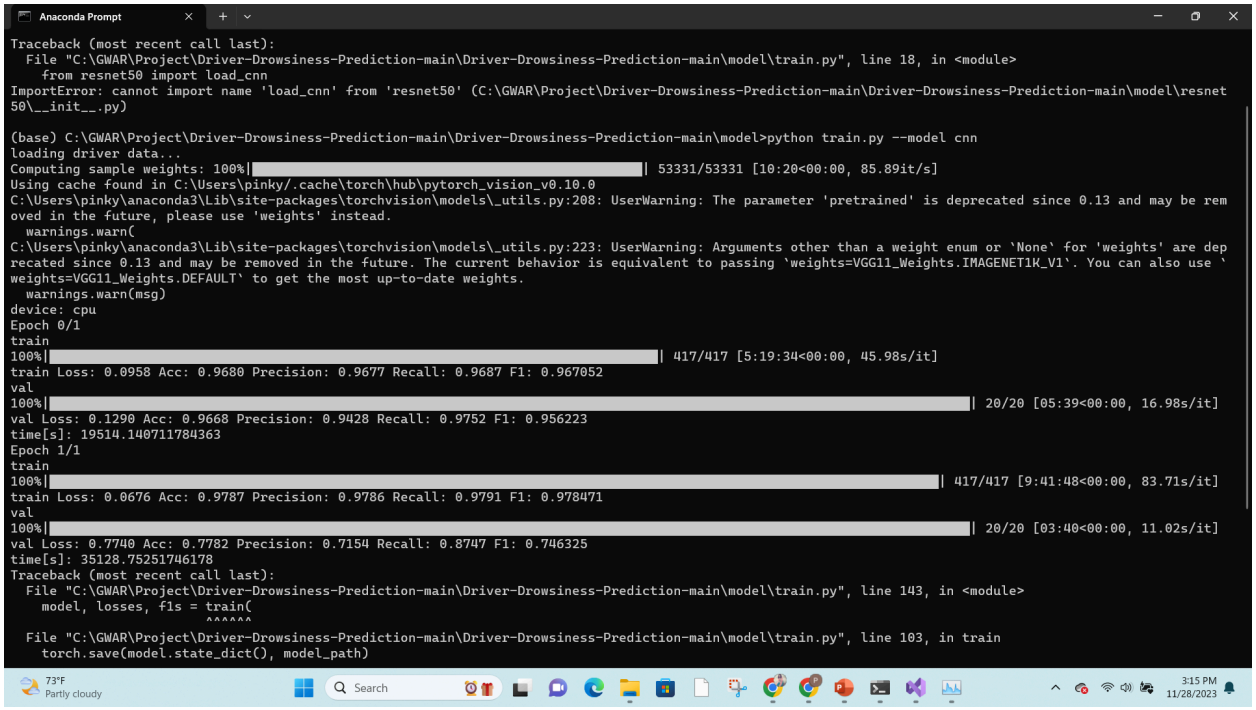


Figure B32. Training Cnn Model Using The Training Pipeline.



Inference using a model can be done by running “Python pred_pipeline.py --model [MODEL NAME] --image_location [LOCATION OF IMAGE]” here MODEL NAME can be replaced with one of the following (VGGnet, CNN, YOLOv3, violajones) and LOCATION OF IMAGE can be replaced with any jpg or jpeg image files

Figure B33. Loading Modules For Prediction Pipeline

```
import argparse
import os
import re
print("starting")
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

import cv2
import numpy as np
import torch
import matplotlib.pyplot as plt
from albumentations.pytorch import ToTensorV2
from retinaface.pre_trained_models import get_model

from model.load_data import get_test_transform, label_ds
from model.resmasknet import load_resmasknet
from model.resnet50 import load_resnet50
```

Figure B34. Loading All Modules And Files Required For Model Inference


```
import argparse
import os
import re
print("starting")
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

import cv2
import numpy as np
import torch
import matplotlib.pyplot as plt
from albumentations.pytorch import ToTensorV2
from retinaface.pre_trained_models import get_model

from model.load_data import get_test_transform, label_ds
from model.resmasknet import load_resmasknet
from model.resnet50 import load_resnet50
```

Figure B35. Code Showing How Prediction Pipeline Can Load Any Model And Generate The Inference Based On That Model By Loading Their Respective Weights.

```python
if __name__ == "__main__":
    # parse arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", type=str, required=True, default="resnet50")
    args = parser.parse_args()

    # load models
    print(f"loading model {args.model}...", flush=True)
    if args.model == "resnet50":
        model = load_resnet50(n_classes=3)
    elif args.model == "resmasknet":
        model = load_resmasknet(n_classes=3)
    elif args.model == "vggnet":
        model = load_vggnet(n_classes=3)
    elif args.model == "cnn":
        model = load_cnn(n_classes=3)
    elif args.model == "yolov3":
        model = load_yolov3(n_classes=3)
    elif args.model == "violajones":
        model = load_vj(n_classes=3)
    else:
        raise Exception(f"Model: {args.model} not supported!")

    state_dict_path =f"./model/{args.model}/{args.model}_ds.pt"
    print(f"loading state dictionary from vgg net", flush=True)
    state_dict = torch.load(state_dict_path)
    model.load_state_dict(state_dict)
    model.eval()
    retinaface = get_model("resnet50_2020-07-20", max_size=2048, device="cpu")
    retinaface.eval()

    # define the pred. pipeline object.
    dsp = DriverStatePredictor(face_detector=retinaface, driver_state_model=model)
    img_path = r"C:\Users\pinky\OneDrive\Pictures\Camera Roll\WIN_20231130_01_03_29_Pro.jpg"
```

Figure B36. Pred Pipeline Showing Inference Using Vgg19 Model

```
(base) C:\GWAR\Project\Driver-Drowsiness-Prediction-main\Driver-Drowsiness-Prediction-main>python pred_pipeline.py --model vggnet
starting
loading model vggnet...
loading state dictionary from vgg net
C:\Users\pinky\anaconda3\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and
 may be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\pinky\anaconda3\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weigh
ts' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
C:\Users\pinky\anaconda3\Lib\site-packages\torch\hub.py:682: UserWarning: Falling back to the old format < 1.6. This support will be deprecated i
n favor of default zipfile format introduced in 1.6. Please redo torch.save() to save it in the new zipfile format.
  warnings.warn('Falling back to the old format < 1.6. This support will be '
step 1
Started prediction pipeline
Inferencing ..............
Inferending completed
Saving Image with inference, bounding box and probability
Saved
Completed Prediction Pipeline

(base) C:\GWAR\Project\Driver-Drowsiness-Prediction-main\Driver-Drowsiness-Prediction-main>
```