

**ABSTRACT**

The project aimed to develop a real-time eye state monitoring system using computer vision techniques and deep learning. The system focused on detecting whether a person's eyes were open or closed, with the objective of enhancing safety and awareness in various contexts, such as driver drowsiness detection and fatigue monitoring.

The project followed a systematic approach, starting with the collection and preparation of a dataset of eye images. The dataset consisted of images of open and closed eyes, which were resized and normalized to ensure consistency and facilitate training. The dataset was then divided into training and testing sets for model evaluation.

Transfer learning was employed using the InceptionV3 model as the base model. Additional layers were added to the base model for classification purposes. By freezing the weights of the base model, the added layers were trained using the prepared dataset. The model was optimized using the Adam optimizer and categorical cross-entropy loss function.

The training process involved data augmentation techniques, including rotation, shear, zoom, and shift operations, to increase the diversity and robustness of the training data. The model was trained for multiple epochs, with a validation subset used to monitor the model's performance and prevent overfitting.

After the model training, a real-time eye monitoring system was implemented. The system utilized a webcam or video input to capture frames. Using face and eye detection algorithms, the system identified the region of interest (ROI) containing the eyes within each frame. The ROI was pre-processed, resized, and normalized before being fed into the trained model for prediction. Based on the model's output probabilities, the system determined whether the eyes were open or closed.

A picture containing text, font, logo, carmine

Description automatically generated

To enhance the usability of the system, a scoring mechanism was implemented to track the duration of closed or open eyes. An alarm sound was triggered if the eyes remained closed for an extended period, serving as an alert for potential drowsiness or fatigue.

The project's outcome demonstrated the successful development and implementation of a real-time eye state monitoring system. However, the system's performance is contingent on the quality of the training data, the effectiveness of the deep learning model, and the accuracy of the face and eye detection algorithms. Further improvements could be made by incorporating more diverse and representative datasets, fine-tuning the model architecture, and optimizing the detection algorithms.

Overall, the project highlights the potential of computer vision and deep learning techniques in addressing eye state monitoring challenges and offers a foundation for further research and application in various domains requiring enhanced safety and attention tracking.

A picture containing text, font, logo, carmine

Description automatically generated

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Name** | **Page No** |
| 1 | Introduction | 04 – 09 |
| 2 | Review of Literature | 10 – 13 |
| 3 | Existing System (Research Gap) | 14 – 16 |
| 4 | Proposed System (Research Objectives) | 17 – 38 |
| 5 | System Design | 39 – 41 |
| 6 | Results | 42 – 44 |
| 7 | Conclusion | 45 – 46 |
| 8 | References | 47 |

A picture containing text, font, logo, carmine

Description automatically generated

LIST OF FIGURES:-

5.1 USE CASE DIAGRAM…………………………………………………………… 39

5.2 ACTIVITY DIAGRAM…………………………………………………………… 40

5.3 BLOCK DIAGRAM……………………………………………………………….. 41

A picture containing text, font, logo, carmine

Description automatically generated

**INTRODUCTION**

1.1 Introduction

Driving while fatigued or drowsy is a dangerous behaviour that poses a significant risk to road safety. The consequences of falling asleep at the wheel can be devastating, leading to severe accidents, injuries, and even loss of life. As such, the development of effective solutions to prevent drowsy driving incidents has become a critical focus in the field of transportation safety.

The Sleep Alert System for Drivers is an innovative project aimed at addressing this urgent issue. This system leverages advanced technology to detect signs of driver fatigue and provide timely alerts, enabling drivers to take necessary measures to prevent accidents due to drowsiness.

This project encompasses the integration of various components, including camera-based detection systems, data analysis algorithms, and real-time alert mechanisms. By continuously monitoring the driver's behaviour, such as eye movements, the Sleep Alert System can accurately identify early signs of sleeping or drowsiness.

Once the system detects potential sleeping/drowsiness, it triggers an immediate alert to the driver, effectively notifying them of their deteriorating state. The alert is in the form of auditory signals, ensuring that the driver is adequately notified and prompted to take appropriate action.

The Sleep Alert System for Drivers goes beyond mere detection and alerting. It also improves driver safety. Through data analysis and machine learning techniques, the system can identify if the driver is sleeping or is about to sleep with fatigue, with the help of auditory signals it helps the drivers to make informed decisions about their rest breaks and overall well-being.

A picture containing text, font, logo, carmine

Description automatically generated

1.2 Necessity

Sleepy/drowsy driving poses a significant threat to road safety, making the development and implementation of effective countermeasures an urgent necessity. The Sleep Alert System for Drivers addresses this pressing need by providing a proactive and reliable solution to detect and prevent sleepiness/drowsiness-related accidents.

One of the primary reasons for the necessity of this system is the alarming statistics surrounding drowsy driving incidents. Road safety is a significant public health issue, and a cause of injuries and fatalities. According to a report by the **Ministry of Road Transport and Highways Transport Research Wing**, road accidents claimed 1,53,972 lives and harmed. 3,84,448 people in 2021. Unfortunately, the age range that is most severely hit by road accidents is 18 to 45 years old, which accounts for almost 67 percent of all accidental deaths. These staggering numbers highlight the critical importance of tackling this issue and implementing preventive measures.

Furthermore, the consequences of sleepy/drowsy driving accidents are often severe. When drivers fall asleep at the wheel, they lose control of their vehicles, leading to high-speed collisions, rollovers, or collisions with other vehicles or objects. Such accidents can cause life-changing injuries and tragic loss of life. By implementing the Sleep Alert System, the aim is to prevent these accidents before they occur, saving lives and reducing the devastating impact on individuals, families, and communities. Another key factor that emphasizes the necessity of the Sleep Alert System is the inherent limitations of human judgment and self-awareness.

Sleepiness/drowsiness can creep in without warning, and drivers often underestimate their level of fatigue. Many individuals mistakenly believe that they can push through their tiredness or rely on brief moments of microsleep to continue driving. However, these risky behaviours significantly increase the likelihood of accidents. The Sleep Alert System acts as an objective and reliable monitor, detecting signs of sleepiness/drowsiness that may go unnoticed by the

A picture containing text, font, logo, carmine

Description automatically generated

driver and providing timely alerts, ensuring that drivers are aware of their fatigue levels and can take appropriate action to prevent accidents.

Moreover, the Sleep Alert System is not limited to long-haul truck drivers or commercial vehicle operators. Fatigue can affect anyone, including those commuting to work, individuals on long road trips, or drivers working irregular shifts. This system has the potential to benefit all drivers, regardless of their driving patterns or vehicle type. By providing a universal solution to address drowsy driving, the Sleep Alert System ensures that road safety is improved for everyone.

1.3 Objective

The objective of the Sleep Alert System for Drivers is to enhance road safety by effectively detecting and preventing drowsy driving incidents. This innovative system is designed to achieve the following key objectives:

1. Early Detection of Sleepiness: The primary objective of the Sleep Alert System is to detect signs of sleepiness/drowsiness in drivers as early as possible. By monitoring indicators such as eye movements, the system can accurately identify the onset of fatigue. This early detection allows for triggering alarms and timely intervention and prevents the progression of drowsiness, reducing the risk of accidents.
2. Timely Alerts and Warnings: Once the system detects signs of drowsiness, its second objective is to promptly alert the driver. The alerts is in the form of auditory signals, effectively notifying the driver of their deteriorating state and prompting them to take immediate action. By providing timely warnings, the system aims to prevent accidents by ensuring that drivers remain vigilant and responsive while on the road.

A picture containing text, font, logo, carmine

Description automatically generated

1. Universal Application: The objective of the Sleep Alert System is to have a universal application across various driving contexts. It is designed to benefit not only commercial drivers but also individuals driving for personal or recreational purposes. Whether it is a long-haul trucker, a commuter, or a family on a road trip, the system aims to enhance safety for all drivers, regardless of their driving patterns or vehicle types.

1.4 Theme

**Theme: "Vigilance for Safer Journeys"**

The theme for the Sleep Alert System for Drivers project is "**Vigilance for Safer Journeys**." This theme encompasses the core objective of the system, which is to enhance road safety by promoting driver vigilance and preventing sleepy/drowsy driving incidents.

The theme highlights the importance of remaining vigilant while on the road, as drowsiness can significantly impair a driver's ability to react and make critical decisions. By emphasizing vigilance, the project aims to create awareness among drivers about the dangers of driving while fatigued and the need for proactive measures to ensure safer journeys.

Furthermore, the theme reflects the role of the Sleep Alert System in promoting vigilance. Through its advanced technology, the system enables drivers to stay alert and attentive by detecting signs of drowsiness and providing timely alerts. By emphasizing the importance of vigilance, the project encourages drivers to prioritize their well-being and take necessary actions to prevent accidents caused by fatigue.

A picture containing text, font, logo, carmine

Description automatically generated

The theme also conveys the project's commitment to fostering a culture of safety on the roads. By promoting vigilance and safer journeys, the Sleep Alert System aims to contribute to a positive shift in driver behaviour and attitudes towards drowsy driving. It encourages drivers to be proactive in managing their fatigue, making informed decisions about rest breaks, and adopting strategies to prevent accidents.

1.5 Intended Audience

The intended audience for this document are the development team, the project evaluation jury, and other tech-savvy enthusiasts who wish to further work on the project.

1.6 Problem Definition

The Sleep Alert System for D rivers tackles the problem of drowsy driving, which poses a major risk to road safety. The objective is to detect signs of driver fatigue early and provide timely alerts to prevent accidents caused by drowsiness.

Drowsy driving is a widespread issue as drivers often underestimate its dangers and fail to recognize their own fatigue until it's too late. This lack of awareness increases the risk of accidents and endangers lives.

The problem arises from drivers' difficulty in accurately perceiving their level of fatigue and ignoring warning signs. This leads to attention lapses, slower reactions, and even microsleep episodes, impairing driving abilities and raising accident risks.

A picture containing text, font, logo, carmine

Description automatically generated

To solve this, the Sleep Alert System integrates sensors, data analysis, and alerts to objectively detect fatigue. Challenges include developing accurate algorithms, seamless sensor integration, and attention-grabbing alerts.

By addressing these challenges, the system aims to mitigate drowsy driving risks, prevent accidents, and save lives. It empowers drivers to prioritize their well-being, stay alert, and make informed decisions about driving and rest breaks.

A picture containing text, font, logo, carmine

Description automatically generated

**REVIEW OF LITERATURE**

2.1 Technologies Used

Here is a small description of all the libraries we used from the starting to the end of the project.

2.1.1 JUPYTER LAB-

Jupyter Lab is a powerful interactive development environment (IDE) that allows data scientists, researchers, and developers to create and share documents containing live code, equations, visualizations, and narrative text. It provides a flexible and user-friendly interface for working with Jupyter notebooks, which combine executable code with rich media outputs. Jupyter Lab offers a range of features, including a file explorer, code editor, and interactive consoles, enabling seamless collaboration, data exploration, and prototyping. Its extensibility and support for multiple programming languages make it a preferred choice for data analysis, machine learning, and scientific computing workflows.

2.1.2 PYTHON-

Python is a widely used, versatile programming language known for its simplicity and readability. Created by Guido van Rossum, Python offers a clean and concise syntax, making it easy to learn and understand. It supports multiple programming paradigms, has an extensive standard library, and a vibrant community that contributes to its vast ecosystem of tools and frameworks. Python is widely adopted for web development, data analysis, scientific computing, and artificial intelligence applications. With cross-platform compatibility and integration capabilities, Python remains a popular choice for developers seeking productivity and flexibility in their coding endeavours.

A picture containing text, font, logo, carmine

Description automatically generated

* + 1. PYTHON MODULES/LIBRARIES-
  1. For Data Pre-processing-
     1. OS module - The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules.
     2. Shutil module - The Shutil module allows you to do high-level operations on a file, such as copy, create, and remote operations. It falls within the umbrella of Python's basic utility modules.
     3. glob module - The glob module, which is short for global, is a function that is used to search for files that match a specific file pattern or name. It can be used to search CSV files and for text in files.
     4. Random - The Python Random module is a built-in module for generating random integers in Python.
  2. For Model Training-
     1. Tensorflow - TensorFlow is a Python library for fast numerical computing created and released by Google.
     2. Keras - Keras is a high-level, deep learning API developed by Google for implementing neural networks. Inceptionv3 - Inception v3 is an image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset.

A picture containing text, font, logo, carmine

Description automatically generated

* + 1. Keras model - Keras is a neural network Application Programming Interface (API) for Python that is tightly integrated with TensorFlow, which is used to build machine learning models. Keras' models offer a simple, user-friendly way to define a neural network, which will then be built for you by TensorFlow.
    2. tensorflow.keras.layers - The output of one layer will flow into the next layer as its input
    3. ImageDataGenerator - The ImageDataGenerator class allows your model to receive new variations of the images at each epoch.

* 1. For Model Implementation –
     1. Cv2 - OpenCV is a Python library that allows you to perform image processing and computer vision tasks. It provides a wide range of features, including object detection, face recognition, and tracking.

* + 1. Load\_Model – A library from tensorflow.keras.models. model\_loads is an open-source Python package for pytorch load models easy.
    2. NumPy - NumPy (Numerical Python) is an open-source Python library that is used in almost every field of science and engineering. It is the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems.
    3. Mixer – A library from pygame. The Mixer is a helper to generate instances of Django or SQLAlchemy models. It's useful for testing and fixture replacement.

A picture containing text, font, logo, carmine

Description automatically generated

2.1.4 OPERATING SYSTEM -

Windows 11

2.2 Hardware Requirements

* + 1. Basic with laptop hardware
    2. Webcam

A picture containing text, font, logo, carmine

Description automatically generated

**EXISTING SYSTEM**

* 1. DMS by TESLA

Tesla, a leading electric vehicle manufacturer, incorporates a drowsiness alert system in its vehicles to enhance driver safety and prevent accidents caused by drowsy driving. The system, known as Driver Monitoring System (DMS), utilizes advanced sensor technology and artificial intelligence to detect signs of driver fatigue or inattention.

The DMS in Tesla vehicles primarily relies on a combination of cameras and sensors strategically placed within the cabin. These sensors continuously monitor the driver's behaviour, including eye movements, head position, and facial expressions. The system tracks the driver's attention and analyses patterns to determine if there are signs of drowsiness or distraction.

When the DMS detects potential signs of drowsiness or driver inattention, it issues visual and audible alerts to grab the driver's attention and prompt them to focus on the road. These alerts can take the form of visual cues on the instrument cluster or center display, as well as audio warnings.

In addition to the real-time alerts, Tesla's DMS also provides recommendations for the driver to take a break and rest if signs of fatigue persist. These recommendations aim to ensure the driver's well-being and minimize the risks associated with drowsy driving.

Tesla's DMS continually evolves through over-the-air software updates, allowing for refinements and enhancements to its algorithms and detection capabilities. This adaptive approach enables the system to become more accurate and reliable over time, adapting to different driving conditions and individual drivers' behaviours.

A picture containing text, font, logo, carmine

Description automatically generated

The integration of the drowsiness alert system in Tesla vehicles is part of the company's commitment to driver safety and their vision of advancing autonomous driving technologies. By actively monitoring driver attention and intervening when drowsiness or distraction is detected, Tesla aims to reduce the likelihood of accidents caused by driver fatigue, ultimately contributing to safer roads for all users.

* + 1. Drawbacks of the Existing System

While the Drowsiness Alert System in Tesla vehicles offers significant benefits in terms of driver safety, there are a few potential drawbacks associated with its implementation:

* 1. False Alarms: The DMS may occasionally generate false alarms, triggering alerts when the driver is not actually fatigued or inattentive. Factors like poor lighting conditions, wearing sunglasses, or certain facial expressions may be misinterpreted by the system, leading to unnecessary warnings and potential driver distraction.
  2. Limited Effectiveness for Certain Individuals: The DMS may not be as effective for individuals with unique facial features or those who wear glasses or sunglasses. These factors could hinder the system's ability to accurately track eye movements and facial expressions, reducing its overall effectiveness in detecting drowsiness or distraction.
  3. Reliance on Visual Cues: The DMS primarily relies on visual cues, such as eye movements and facial expressions, to determine driver alertness. However, this approach may overlook certain signs of drowsiness, such as changes in body posture or decreased steering responsiveness. Relying solely on visual cues may limit the system's ability to comprehensively detect driver fatigue in all situations.

A picture containing text, font, logo, carmine

Description automatically generated

* 1. Overreliance on Technology: The presence of a drowsiness alert system might inadvertently lead some drivers to become overly reliant on the technology, assuming it will always detect fatigue and prevent accidents. This overreliance could potentially result in complacency and a decrease in personal vigilance, undermining the intended purpose of the system.
  2. Lack of Customization: The DMS in Tesla vehicles may lack customization options for individual drivers. Factors such as different driver behaviours, sleep patterns, or fatigue tolerance levels may not be fully considered or adjustable in the system, leading to alerts that may not align with each driver's specific needs.
  3. Limitations in Challenging Conditions: The effectiveness of the DMS might be compromised in challenging driving conditions, such as heavy rain, fog, or glare. Reduced visibility or environmental factors may impact the system's ability to accurately detect drowsiness or driver inattention, potentially increasing the risk of false negatives.
  4. Cost: The high cost of manufacturing and unavailability of the product also plays an important role in the unavailability of the product to the consumer.
  5. DMS in INDIA

The only vehicle in India which is about to release the sleep alert system is the XUV 700 by the **Mahindra automobiles.** There is no otherautomobile industry whichhad yet released this alert system.

A picture containing text, font, logo, carmine

Description automatically generated

**PROPOSED SYSTEM**

4.1 Objectives

The proposed system aims to address the limitations of existing drowsiness alert systems by incorporating advanced technologies and innovative features to enhance driver safety and prevent accidents caused by drowsy driving.

4.1.1 Key Features

1. Early Identification of Fatigue: The primary goal of the Sleep Alert System is to detect the early signs of fatigue or drowsiness in drivers. Through continuous monitoring of factors such as eye movements and other relevant indicators, the system can accurately recognize the initial stages of fatigue. This early identification enables the system to issue timely alarms and interventions, preventing the progression of drowsiness and reducing the likelihood of accidents.
2. Timely Alarms and Notifications: Once the system detects drowsiness, its secondary objective is to promptly notify the driver. This is achieved with audible alerts that effectively inform the driver about their deteriorating state and urge them to take immediate action. By providing timely warnings, the system aims to avert accidents by ensuring that drivers remain attentive and responsive while operating their vehicles.
3. Personalized and Adaptive Functionality: The system will be customizable to accommodate individual driver preferences and unique physiological characteristics. It will adapt to different driving styles, sleep patterns, and tolerance levels for fatigue, providing tailored alerts and interventions that align with each driver's specific needs.

A picture containing text, font, logo, carmine

Description automatically generated

1. Machine Learning Algorithms: The proposed system will employ machine learning algorithms to analyse the collected data and identify patterns indicative of drowsiness. These algorithms will continuously adapt and learn from driver behaviour, leading to improved accuracy and customized detection of drowsiness based on individual driver profiles.
2. Real-Time Alerts and Interventions: When signs of drowsiness are detected, the system will provide immediate and customizable alerts to the driver. These alerts may include visual cues on the dashboard, audible warnings, or haptic feedback through the steering wheel or seat. Additionally, the system may suggest appropriate interventions, such as recommending rest breaks, playing soothing music, or adjusting cabin temperature to promote driver alertness.

4.1.2 Updates to Be Made:

1. Multi-Sensor Integration: The system will integrate multiple sensors, such as cameras, infrared sensors, and physiological monitors, to gather comprehensive data about the driver's behaviour, vital signs, and external conditions. This multi-sensor approach ensures a more accurate and reliable assessment of driver drowsiness.
2. Personalized and Adaptive Functionality: The system will be customizable to accommodate individual driver preferences and unique physiological characteristics. It will adapt to different driving styles, sleep patterns, and tolerance levels for fatigue, providing tailored alerts and interventions that align with each driver's specific needs.

A picture containing text, font, logo, carmine

Description automatically generated

1. Data Logging and Analysis: The proposed system will log and analyse data related to driver drowsiness, providing valuable insights for both the driver and fleet managers. This data can be used to identify patterns, trends, and potential areas for improvement in driver behaviour, sleep patterns, or work.
2. Integration with Vehicle Control Systems: The proposed system will have the capability to interface with vehicle control systems, such as adaptive cruise control or lane-keeping assist. When drowsiness is detected, the system can automatically engage safety features or modify driving settings to ensure a safer driving experience, such as reducing speed, increasing following distance, or providing gentle steering corrections.

By incorporating these advanced features, the proposed drowsiness alert system aims to significantly enhance road safety and minimize the risks associated with drowsy driving. It seeks to provide timely and personalized interventions, improve detection accuracy, and promote driver well-being during long journeys or challenging conditions.

4.2 METHODOLOGY

4.2.1 Main Code Explanation

In the first module i.e., Sleep Alert System. This system detects whether the driver is drowsy or not by considering eyes and facial landmarks. The first step to it is to perform face recognition in order to detect blink of an eye, where the system recognizes the driver's eyes and then calculates the speed of eye blink and provides an appropriate output to whether trigger an alarm or not.

A picture containing text, font, logo, carmine

Description automatically generated



This code snippet demonstrates the implementation of a face and eye detection system using the HAAR cascade classifier and a pre-trained deep learning model.

1. cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml'): This line initializes a CascadeClassifier object from OpenCV, specifically using the haarcascade\_frontalface\_default.xml file. This file contains the pre-trained model for detecting frontal faces.
2. cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_eye.xml'): This line initializes another CascadeClassifier object, using the haarcascade\_eye.xml file. This file contains the pre-trained model for detecting eyes within the detected faces.
3. model = load\_model(r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\model.h5'): This line loads a pre-trained deep learning model using the Keras library's load\_model function. The model.h5 file represents the trained model for eye recognition, trained on the MRL Eyes dataset.

By combining the face cascade classifier, eye cascade classifier, and the deep learning model, this code sets up the necessary components for detecting faces and eyes in an image or video stream. These detections can be utilized for various applications such as facial recognition, gaze tracking, or driver drowsiness detection.

A picture containing text, font, logo, carmine

Description automatically generated

A screenshot of a computer screen

Description automatically generated with medium confidence

A picture containing text, font, logo, carmine

Description automatically generated

The provided code demonstrates a Sleep Alert System using face and eye detection, along with sound alerts. Here's a breakdown of the code:

1. `mixer.init()`: Initializes the sound mixer.

2. `sound = mixer.Sound(r"C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\emergency-alarm-with-reverb-29431.mp3")`: Loads an emergency alarm sound file.

3. `cap = cv2.VideoCapture(r"C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\test123.mp4")`: Opens a video file for processing.

4. `Score = 0`: Initializes a score variable.

5. The code enters a while loop to process each frame of the video.

6. `ret, frame = cap.read()`: Reads the next frame from the video.

7. `gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)`: Converts the frame to grayscale.

8. `faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=3)`: Detects faces in the grayscale frame using the pre-trained face cascade classifier.

A picture containing text, font, logo, carmine

Description automatically generated

9. `eyes = eye\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=1)`: Detects eyes in the grayscale frame using the pre-trained eye cascade classifier.

10. The code iterates through the detected faces and eyes, drawing rectangles around them on the frame.

11. Preprocessing and prediction steps are performed for each detected eye:

- The eye region is extracted and resized to a standard size (80x80 pixels).

- The pixel values of the eye image are normalized.

- The eye image is reshaped to match the input shape expected by the model.

- The model predicts whether the eye is open or closed.

12. Based on the model predictions, the code updates the frame with text indicating whether the eyes are open or closed and the current score.

13. If the eyes are predicted to be closed for a certain number of frames (Score > 15), the emergency alarm sound is played.

14. If the eyes are open, the score is decreased (Score -= 1), and if the score goes below 0, it is reset to 0.

15. The processed frame is displayed using `cv2.imshow()`.

A picture containing text, font, logo, carmine

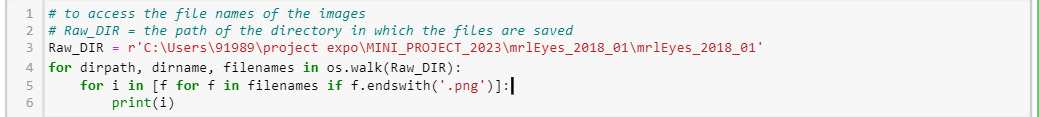
Description automatically generated

16. The code waits for a 'q' key press to break the loop and exit the program.

17. `cap.release()` releases the video capture object.

18. `cv2.destroyAllWindows()` closes all windows.

4.2.2 Data Pre-Processing Code Explanation



The provided code snippet uses the `os.walk()` function to traverse through a directory and its subdirectories, and then prints the names of all the PNG image files found.

Here's a breakdown of the code:

1. `Raw\_DIR = r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\mrlEyes\_2018\_01'`: Defines the directory path where the code will start traversing.

A picture containing text, font, logo, carmine

Description automatically generated

2. `os.walk(Raw\_DIR)`: The `os.walk()` function generates the file names in a directory tree by iterating over the directory tree root (`Raw\_DIR`), its subdirectories, and the files within them.

3. `for dirpath, dirname, filenames in os.walk(Raw\_DIR)`: This line sets up a loop that iterates over each subdirectory (`dirpath`), its immediate child directories (`dirname`), and the filenames within each subdirectory (`filenames`).

4. `for i in [f for f in filenames if f.endswith('.png')]:`: This nested loop iterates through the filenames in the current subdirectory and filters out only the files that have the '.png' extension.

5. `print(i)`: This line prints each filtered filename (i.e., the names of the PNG image files) found in the directory tree.

A screenshot of a computer program

Description automatically generated with low confidence

The code provided performs file operations using the `os.walk()` function and the `shutil.copy()` function. It copies PNG image files from a source directory to different destination directories based on their filenames.

Here's a breakdown of the code:

1. `for dirpath, dirname, filenames in os.walk(Raw\_DIR)`: This line initiates a loop that iterates over each subdirectory (`dirpath`), its immediate child directories (`dirname`), and the filenames within each subdirectory (`filenames`) in the specified directory tree (`Raw\_DIR`).

A picture containing text, font, logo, carmine

Description automatically generated

2. `for i in tqdm([f for f in filenames if f.endswith('.png')])`: This nested loop filters out only the PNG image files from the filenames within the current subdirectory. The `tqdm()` function wraps the loop to provide a progress bar for the file copying process.

3. `if i.split('\_')[4]=='0':`: This conditional statement checks if the fifth element of the split filename (using the '\_' delimiter) is equal to '0'. It identifies if the image file represents closed eyes.

4. `shutil.copy(src = dirpath + '/' + i , dst = r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\Prepared\_Data\Close Eyes')`: This line copies the image file from the source directory (`dirpath + '/' + i`) to the destination directory (`r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\Prepared\_Data\Close Eyes'`). The `shutil.copy()` function is used for the file copy operation.

5. `elif i.split('\_')[4]=='1':`: This conditional statement checks if the fifth element of the split filename is equal to '1'. It identifies if the image file represents open eyes.

6. `shutil.copy(src = dirpath + '/' + i , dst = r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\Prepared\_Data\Open Eyes')`: This line copies the image file from the source directory to a different destination directory (`r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\Prepared\_Data\Open Eyes'`).

A picture containing text, font, logo, carmine

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

The provided code defines a function `split\_files` that splits files from a source directory into train and test directories based on a given split ratio.

Here's a breakdown of the code:

1. `def split\_files(source\_dir, train\_dir, test\_dir, split\_ratio):`: This line defines the `split\_files` function that takes four arguments: `source\_dir` (the source directory containing the files to be split), `train\_dir` (the destination directory for train files), `test\_dir` (the destination directory for test files), and `split\_ratio` (the ratio to split the files).

A picture containing text, font, logo, carmine

Description automatically generated

2. `os.makedirs(train\_dir, exist\_ok=True)` and `os.makedirs(test\_dir, exist\_ok=True)`: These lines create the train and test directories if they don't already exist using the `os.makedirs()` function. The `exist\_ok=True` argument ensures that the directories are created only if they don't exist.

3. `file\_list = os.listdir(source\_dir)`: This line retrieves a list of file names in the source directory using the `os.listdir()` function.

4. `random.shuffle(file\_list)`: This line shuffles the order of the file names randomly using the `random.shuffle()` function.

5. `split\_index = int(len(file\_list) \* split\_ratio)`: This line calculates the index at which to split the file list based on the split ratio. It multiplies the length of the file list by the split ratio and converts it to an integer using `int()`.

6. `for file\_name in file\_list[:split\_index]:`: This loop iterates through the file names up to the split index, representing the train files.

7. `source\_path = os.path.join(source\_dir, file\_name)`: This line constructs the source file path by joining the source directory path with the current file name using `os.path.join()`.

8. `dest\_path = os.path.join(train\_dir, file\_name)`: This line constructs the destination file path for the train files by joining the train directory path with the current file name.

A picture containing text, font, logo, carmine

Description automatically generated

9. `shutil.move(source\_path, dest\_path)`: This line moves the file from the source path to the destination path using `shutil.move()`.

10. The second loop starting with `for file\_name in file\_list[split\_index:]:` follows the same logic as the previous loop but iterates from the split index to the end of the file list, representing the test files. The files are moved from the source directory to the test directory.

11. The code outside the function defines the source directory, train directory, test directory, and split ratio.

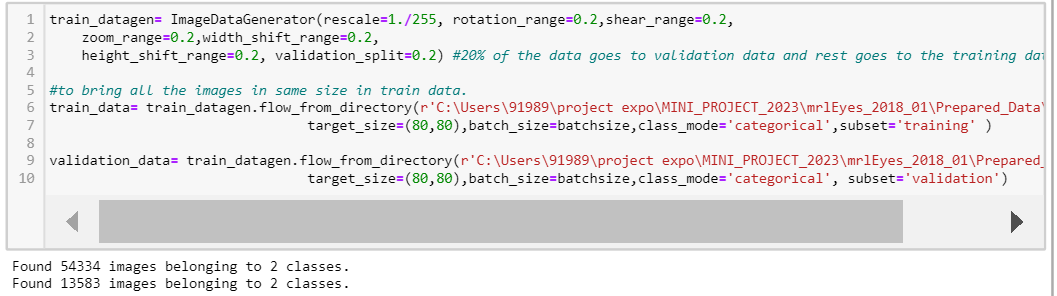
12. `split\_files(source\_directory, train\_directory, test\_directory, split\_ratio)`: This line calls the `split\_files` function with the provided arguments to perform the file splitting operation.

Note:-For splitting the closed eyes in test and train part the same code with different directory is used in our model.

A picture containing text, font, logo, carmine

Description automatically generated

4.2.3 Model Training Using InceptionV3 Code Explanation



The provided code sets up an image data generator using `ImageDataGenerator` from Keras for data augmentation and preprocessing. It also creates data generators for the train and validation datasets. Here's a breakdown of the code:

1. `train\_datagen = ImageDataGenerator(rescale=1./255, rotation\_range=0.2, shear\_range=0.2, zoom\_range=0.2, width\_shift\_range=0.2, height\_shift\_range=0.2, validation\_split=0.2)`: This line creates an instance of `ImageDataGenerator` with various augmentation and preprocessing parameters. These parameters include rescaling the pixel values to a range of 0 to 1, rotating the images by a maximum of 0.2 radians, shearing the images by a maximum of 0.2 radians, zooming the images by a maximum factor of 0.2, shifting the width and height of the images by a maximum of 0.2, and setting the validation split to 0.2 (20% of the data will be used for validation).

A picture containing text, font, logo, carmine

Description automatically generated

2. `train\_data = train\_datagen.flow\_from\_directory(...)`: This line creates a data generator for the training dataset using the `flow\_from\_directory` method of the `ImageDataGenerator` class. It specifies the directory of the training data, the target size of the images (80x80 pixels in this case), the batch size, the class mode as 'categorical' (indicating categorical labels), and the subset as 'training' (to select the training subset based on the validation split).

3. `validation\_data = train\_datagen.flow\_from\_directory(...)`: This line creates a data generator for the validation dataset using the same `flow\_from\_directory` method. It specifies the directory of the training data, the target size, batch size, class mode, and subset as 'validation' (to select the validation subset based on the validation split).

A screenshot of a computer

Description automatically generated with medium confidence

The provided code sets up an image data generator for the test dataset using `ImageDataGenerator` from Keras. Here's an explanation of the code:

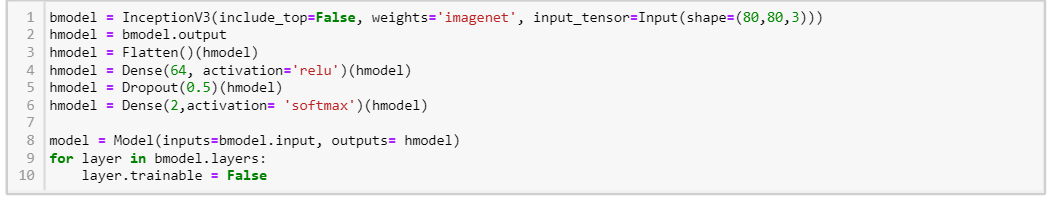
1. `test\_datagen = ImageDataGenerator(rescale=1./255)`: This line creates an instance of `ImageDataGenerator` for the test dataset. The only preprocessing step applied here is rescaling the pixel values to a range of 0 to 1 by dividing them by 255. This ensures that the pixel values are normalized before being used for testing or prediction.

A picture containing text, font, logo, carmine

Description automatically generated

2. `test\_data = test\_datagen.flow\_from\_directory(...)`: This line creates a data generator for the test dataset using the `flow\_from\_directory` method of the `ImageDataGenerator` class. It specifies the directory of the test data, the target size of the images (80x80 pixels in this case), the batch size, and the class mode as 'categorical' (indicating categorical labels).

The purpose of this code is to set up a data generator that will generate batches of test images with the specified preprocessing steps. The `flow\_from\_directory` method automatically reads the images from the given directory, resizes them to the target size, applies the rescaling, and organizes the data into batches. This allows for efficient testing or evaluation of a machine learning model on the test dataset.



The provided code defines a convolutional neural network (CNN) model using the InceptionV3 architecture as a base. Here's an explanation of the code:

1. `bmodel = InceptionV3(include\_top=False, weights='imagenet', input\_tensor=Input(shape=(80,80,3)))`: This line initializes the base model using InceptionV3 architecture. The `include\_top=False` argument means that the fully connected layers on top of the InceptionV3 network will not be included. The `weights='imagenet'` argument loads the pre-trained weights from the ImageNet dataset. The `input\_tensor=Input(shape=(80,80,3))` specifies the input shape of the model as an 80x80 RGB image.

A picture containing text, font, logo, carmine

Description automatically generated

2. `hmodel = bmodel.output`: This line retrieves the output of the base model. In this case, it will be the feature maps obtained from the InceptionV3 layers.

3. `hmodel = Flatten()(hmodel)`: This line adds a `Flatten` layer to convert the multi-dimensional feature maps into a one-dimensional vector. This is necessary to connect the CNN part to the fully connected layers.

4. `hmodel = Dense(64, activation='relu')(hmodel)`: This line adds a fully connected layer with 64 units and a ReLU activation function.

5. `hmodel = Dropout(0.5)(hmodel)`: This line adds a `Dropout` layer with a dropout rate of 0.5. Dropout is a regularization technique that randomly drops a fraction of the connections during training, which helps prevent overfitting.

6. `hmodel = Dense(2,activation= 'softmax')(hmodel)`: This line adds the final fully connected layer with 2 units and a softmax activation function. This layer outputs the predicted probabilities for the two classes (closed eyes and open eyes).

7. `model = Model(inputs=bmodel.input, outputs= hmodel)`: This line creates the final model by specifying the inputs and outputs. The input is set as the input of the base model, and the output is set as the output of the last added layer.

A picture containing text, font, logo, carmine

Description automatically generated

8. `for layer in bmodel.layers: layer.trainable = False`: This loop freezes the weights of the layers in the base model so that they are not updated during training. By setting `trainable = False`, the pre-trained weights are kept fixed, and only the weights of the newly added layers are trained.

The purpose of this code is to create a model architecture that combines the InceptionV3 base model with additional layers for fine-tuning and classification. The InceptionV3 model is used as a feature extractor, and the additional layers are added to adapt the extracted features to the specific task of detecting closed and open eyes.

A screenshot of a computer

Description automatically generated with low confidence

The provided code sets up callback functions to be used during the training of a neural network model. Here's an explanation of the code:

1. `checkpoint = ModelCheckpoint(r'C:\Users\91989\project expo\MINI\_PROJECT\_2023\mrlEyes\_2018\_01\model2.h5', monitor='val\_loss', save\_best\_only=True, verbose=3)`: This line creates a `ModelCheckpoint` callback. It specifies the filepath where the best model weights will be saved during training. The `monitor='val\_loss'` argument means that the validation loss will be monitored. The `save\_best\_only=True` argument ensures that only the best model weights based on the validation loss will be saved. The `verbose=3` argument sets the verbosity level to provide detailed output.

2. `earlystop = EarlyStopping(monitor='val\_loss', patience=20, verbose=3, restore\_best\_weights=True)`: This line creates an `EarlyStopping` callback. It stops the training

A picture containing text, font, logo, carmine

Description automatically generated

process if the monitored validation loss does not improve for a specified number of epochs (in this case, 20). The `verbose=3` argument sets the verbosity level. The `restore\_best\_weights=True` argument restores the weights of the model to the ones obtained at the best epoch.

3. `learning\_rate = ReduceLROnPlateau(monitor='val\_loss', patience=3, verbose=3)`: This line creates a `ReduceLROnPlateau` callback. It reduces the learning rate when the monitored validation loss has stopped improving for a specified number of epochs (in this case, 3). The `monitor='val\_loss'` argument means that the validation loss will be monitored. The `verbose=3` argument sets the verbosity level.

4. `callbacks=[checkpoint, earlystop, learning\_rate]`: This line combines the created callbacks into a list. During the training process, these callbacks will be executed at specific points, such as after each epoch or when certain conditions are met. The `callbacks` list is passed as an argument to the training function to enable the desired callback functionalities.

The purpose of these callbacks is to enhance the training process and improve the performance of the model. The `ModelCheckpoint` callback saves the best model weights, allowing you to restore them later. The `EarlyStopping` callback stops the training if the model does not improve, preventing overfitting. The `ReduceLROnPlateau` callback dynamically adjusts the learning rate, which can help the model converge faster and achieve better performance.

A picture containing text, font, logo, carmine

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

The provided code compiles and trains a neural network model using the Keras framework. Here's an explanation of the code:

1. `model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics=['accuracy'])`: This line compiles the model. It specifies the optimizer to be used (in this case, Adam optimizer), the loss function (categorical cross-entropy), and the metrics to be evaluated during training (accuracy). Compiling the model prepares it for the training process.

2. `model.fit\_generator(train\_data, steps\_per\_epoch=train\_data.samples//batchsize, validation\_data=validation\_data, validation\_steps=validation\_data.samples//batchsize, callbacks=callbacks, epochs=50)`: This line starts the training process. It uses the `fit\_generator` function to train the model on the training data (`train\_data`) and validate it on the validation data (`validation\_data`). The `steps\_per\_epoch` argument specifies the number of steps (batches) per epoch during training, which is calculated by dividing the total number of training samples by the batch size. The `validation\_steps` argument is similar but for the validation data. The `callbacks` argument is set to the list of callbacks created earlier, which will be executed during training. Finally, the `epochs` argument determines the number of training epochs (in this case, 50).

During training, the model will iterate through the training and validation data in batches, computing the gradients and updating the weights based on the specified optimizer and loss function. The provided callbacks will be executed at specific points during training, such as after

A picture containing text, font, logo, carmine

Description automatically generated

each epoch, to perform actions such as saving the best model weights, early stopping, and adjusting the learning rate. The training process continues for the specified number of epochs.

A picture containing text, font, logo, carmine

Description automatically generated

**SYSTEM DESIGN**

5.1 USE CASE DIGRAM

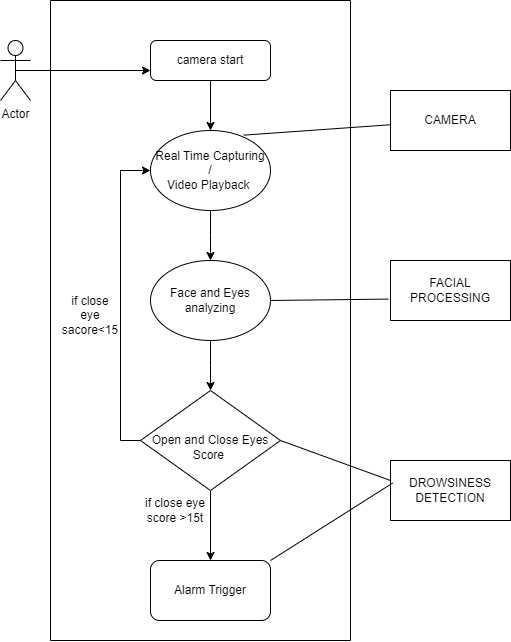


FIGURE 5.1:- USE CASE DIAGRAM

A picture containing text, font, logo, carmine

Description automatically generated

5.2 ACTIVITY DIAGRAM

A white rectangles with black text

Description automatically generated with low confidence

FIGURE 5.2:- ACTIVITY DIAGRAM

A picture containing text, font, logo, carmine

Description automatically generated

5.3 BLOCK DIAGRAM

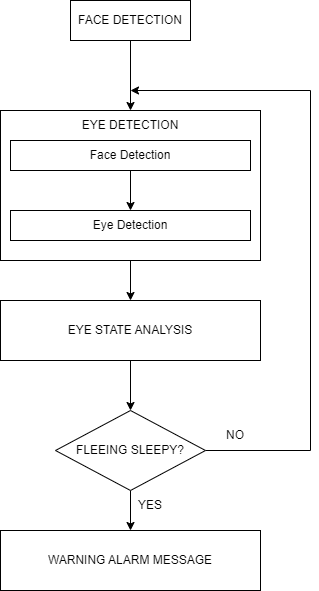


FIGURE 5.3:- BLOCK DIAGRAM

A picture containing text, font, logo, carmine

Description automatically generated

**RESULTS**

6.1 RESULT OF THE MODEL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eyes State | Prediction[0][0] | Prediction[0][1] | Score | Action |
| CLOSED | >0.20 | - | Increase Score by 1 |  |
|  |  |  |  | Play emergency alarm sound |
| OPEN | - | >0.90 | Decrease Score by 1 |  |
|  |  |  | <0 | Reset Score to 0 |

Explanation:

- The code captures video frames from a source (specified by `cap`) and processes them.

- It detects faces and eyes within each frame using Haar cascades (`face\_cascade` and `eye\_cascade`).

- For each detected eye region, it performs the following steps:

- Preprocesses the eye region by resizing it to (80, 80) and normalizing pixel values.

- Feeds the preprocessed eye region to the `model` for prediction.

- Checks the prediction values (`prediction[0][0]` and `prediction[0][1]`) to determine if the eyes are closed or open.

A picture containing text, font, logo, carmine

Description automatically generated

- Based on the prediction, it updates the score and displays the corresponding text on the frame using OpenCV functions.

- If the score exceeds a threshold (`Score > 15`) and the emergency alarm sound file is provided (`sound`), it plays the sound.

- The frame with the annotations is displayed using OpenCV's `imshow` function.

- The loop continues until the user presses the 'q' key to exit.

- Finally, the video capture is released (`cap.release()`) and OpenCV windows are closed (`cv2.destroyAllWindows()`).

Note:- That the behavior of this code heavily depends on the specific model used for eye state classification, the quality of the trained model, and the accuracy of the face and eye detection using Haar cascades .

A picture containing text, font, logo, carmine

Description automatically generated

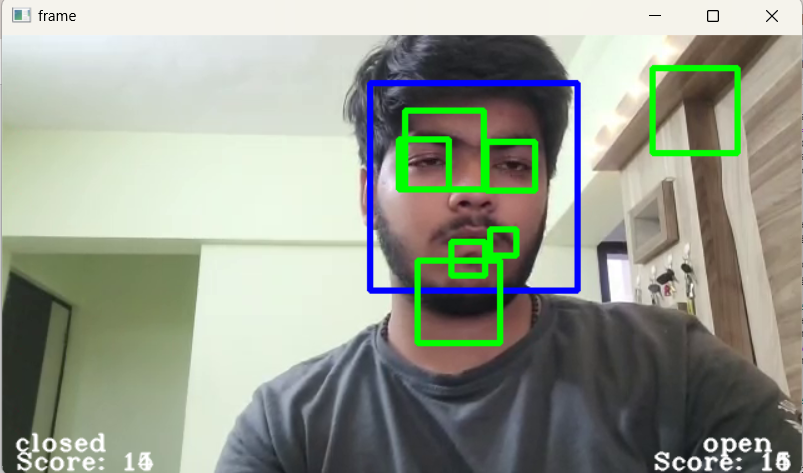
6.2 Screen Shot of The Model

6.2.1 Photo of A Non-Sleepy Person:-

A person taking a selfie

Description automatically generated

6.2.2 Photo of A Sleepy Person:-



A picture containing text, font, logo, carmine

Description automatically generated

**CONCLUSION**

In conclusion, the project focused on developing a system to detect and monitor the state of a person's eyes using computer vision techniques. The main goal was to classify whether the eyes are open or closed in real-time.

The project involved several key steps. First, a dataset of eye images was prepared by collecting and pre-processing images of open and closed eyes. The dataset was then split into training and testing sets to train a deep learning model.

An InceptionV3 model was chosen as the base model and customized by adding additional layers for classification. Transfer learning was utilized by freezing the weights of the base model while training the added layers. The model was trained using the prepared dataset and evaluated using validation data.

Once the model was trained, it was integrated into a real-time eye monitoring system. The system used a webcam or video input to capture frames, detect faces and eyes within the frames, and feed the detected eye regions into the trained model for prediction. Based on the predictions, the system determined whether the eyes were open or closed and maintained a score to track the duration of closed or open eyes. An alarm sound was triggered if the eyes were closed for an extended period.

The project demonstrated the application of computer vision and deep learning techniques to tackle the problem of eye state monitoring. The system could have potential applications in areas such as driver drowsiness detection, fatigue monitoring, and attention tracking in various industries.

A picture containing text, font, logo, carmine

Description automatically generated

However, it is important to note that the accuracy and performance of the system heavily depend on the quality of the training data, the effectiveness of the deep learning model, and the accuracy of the face and eye detection algorithms. Further improvements can be made by incorporating more diverse and representative datasets, fine-tuning the model architecture, and optimizing the detection algorithms.

Overall, the project provides a foundation for developing an eye state monitoring system and highlights the potential for using computer vision techniques to enhance safety and awareness in various contexts.

A picture containing text, font, logo, carmine

Description automatically generated

**REFRENCES**

1. COMPUTATIONALLY EFFICIENT FACE DETECTION; B. SCHLKOPF-A. BLAKE, S. ROMDHANI, AND P. TORR.
2. MRL Eye Dataset | MRL. (n.d.). <http://mrl.cs.vsb.cz/eyedataset>
3. Libraries & extensions. (n.d.). TensorFlow. <https://www.tensorflow.org/resources/libraries-extensions>

1. India, T. O. (2023, January 19). Driver fatigue-led road accidents: Approaches India could take to prevent these. The Times of India. <https://timesofindia.indiatimes.com/auto/policy-and-industry/driver-fatigue-led-road-accidents-approaches-india-could-take-to-prevent-these/articleshow/97048300.cms>
2. Kashikar, G. (2021, August 20). Mahindra shows off driver drowsiness detection feature for the XUV700. CarTrade. <https://www.cartrade.com/news/mahindra-shows-off-driver-drowsiness-detection-feature-for-the-xuv700/#:~:text=Mahindra%20shows%20off%20driver%20drowsiness%20detection%20feature%20for%20the%20XUV700,-Gajanan%20Kashikar&text=Mahindra%20has%20been%20revealing%20one,provided%20in%20the%20upcoming%20XUV700>.

1. Log in. (n.d.). NVIDIA Developer. <https://developer.nvidia.com/rdp/cudnn-download>